NISTIR 7744

Improving the Efficiency of Markov Chain Analysis of Complex Distributed Systems

Christopher Dabrowski
Fern Hunt
Katherine Morrison

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 7744

Improving Efficiency of Markov Chain Analysis of Complex Distributed Systems

Christopher Dabrowski
Advanced NetworkingTechnology 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

Katherine Morrison
University of Nebraska-Lincoln
Department of Mathematics
Avery Hall
Lincoln, NE 68588

November 2010

U.S. Department of Commerce
Gary Locke, Secretary

National Institute of Standards and Technology
Patrick D. Gallagher, Director

Improving Efficiency of Markov Chain Analysis of Complex Distributed Systems

Abstract: In large-scale distributed systems, the interactions of many independent components may lead to emergent global behaviors with unforeseen, often detrimental, outcomes. The increasing importance of distributed systems such as clouds and computing grids will require analytical tools to understand and predict, complex system behavior to ensure system reliability. In previous work, we described how a piecewise homogeneous Discrete Time Markov chain representation of a computing grid can be systematically perturbed to predict situations that lead to performance degradations. While the execution time of this approach compared favorably with detailed large-scale simulation, a sizable number of perturbations of the model were needed to identify scenarios in which system performance degraded. Here, we evolve our original approach and describe two novel methods for quickly identifying portions of the Markov chain that are sensitive to perturbation. The first method involves finding minimal s-t cut sets, consisting of state transitions that disconnect all paths in a Markov chain from the initial to a desired end state. By perturbing state transitions in the cut set, it is possible to quickly identify scenarios in which system performance is adversely affected. We show this method can be applied to larger Markov models than the approach described in our earlier work. 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 to predict 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 needed. We argue that these methods provide a basis for creating practical tools for analysis of complex systems behavior in distributed systems.

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

Contents

1. Introduction

In recent years, the advent of large-scale distributed systems, such as computing grids and commercial cloud systems[1] 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 [1]. The rapid growth and increasing economic importance of these systems [2, 3] 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 [4–6] and failure scenarios [1] 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 [7] 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 described 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 to system-wide failures. This allowed Markov chain analysis to be used to predict 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. We showed that the computational cost of this approach was reduced in comparison with detailed large-scale simulation or testbeds. 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, here 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 minimal s-t 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 [7], 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.

[1] Any mention of commercial products in this document is for information only; it does not imply recommendation or endorsement by NIST.

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 by the perturbation algorithm in [7], 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 at reduced computation cost. Results from application of both methods are detailed for all cases and corroborated by earlier results obtained by applying the perturbation algorithm. We show that the two general approaches presented in this report can be equally effective, but more efficient, than our previous Markov chain analysis using the perturbation algorithm. We also present results indicating scalability of the graph-theoretic method.

The plan of this report 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 work [7], focusing on use of Markov chain concepts to model dynamic systems. Most importantly, Sec. 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 report. Section 4 describes how minimal s-t 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 Sec. 4, a simple method for identifying minimal s-t cut sets is discussed and examples are provided. Section 5 presents the results of using the methods described in Sec. 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 Sec. 3 as well as with more detailed large-scale simulation. The comparison shows that identification of minimal s-t cut sets is 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 s-t 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 1.1. Summary of sections of report.

Section	**Topics**
Section 2	Review of previous work on use of Markov chain analysis to study dynamic systems.
Section 3	Use of Markov chain models to represent dynamic systems. Review of perturbation algorithm in [7] for finding areas of Markov chains that are sensitive to perturbation.
Section 4	Use of minimal cut set analysis to find state transitions in Markov chains that are sensitive to perturbation.
Section 5	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.
Section 6	Description of minimal cut set analysis method for large, complex Markov chains and presentation of preliminary results
Section 7	Presentation of theoretical method, in which the Spectral Expansion Theorem is used to analyze the eigensystem of a Markov chain.

2. Previous Work

The methods described in this report should be distinguished from the well-known use of Discrete Time Markov chains (DTMCs) for providing quantitative measures of system performance and reliability, which we review in [7]. Of this work, most closely related are [8, 9], in which a feedback control loop process is used to moderate delay in a network, where delay characteristics are modeled using a Markov chain. Instead of measuring reliability, we use 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. This section summarizes previous work on using Markov chains to study dynamic system behavior that focus on the main topic of this report—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 [10]. Since then, various lumping approaches have been proposed, including [11, 12] who use model structure and symmetry to reduce size, [13] who rely on group-theoretic concepts for size reduction. Other approaches for reducing model size have been based on stochastic activity nets [14], stochastic colored nets [15], use of reward variable structures to identify symmetries [16], and use of eigenvector equivalence classes to partition a Markov state space into lumps [17]. 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 report).

In the last three decades, perturbation analysis of discrete time Markov chains has been the topic of significant theoretical [18, 19] and computational study [20, 21]. However, much of this work 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 [22, 23] 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 [23, 24] and extended by [25, 26] 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 [27–29] including large-scale models [30]. In [29], 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

[31], distances between stationary distributions of perturbed Markov chains were calculated using graph-theoretic techniques. 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 s-t cut set identification [32–35] described below. Minimal cut set identification methods have long been used for analysis of VLSI designs, network systems, and design of various other distributed systems. For example, in [36], minimal cut sets of avionics system component graphs were used to identify the shortest sequence of individual component failures. 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. Finally, we mention maximum flow algorithms [37–39], which identify minimum cut sets between two vertices in a graph on the basis of maximum flow and minimum capacity. These algorithms have also been used for many practical problems and could potentially be used to identify critical state transitions, as the approach described here does. However, we regard maximum flow algorithms as a distinct approach from Markov chains which do not employ flows. Therefore, flow-based algorithms merit separate treatment, perhaps as future work.

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 [40], both of which were developed by observing the operation of large-scale simulations. We then present an overview of the perturbation algorithm described in [7] 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 Secs. 4–7.

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 [1, 6] that simulates 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 Fig. 3.1, represent actions taken by the grid system to process a task as described in [7]. 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.

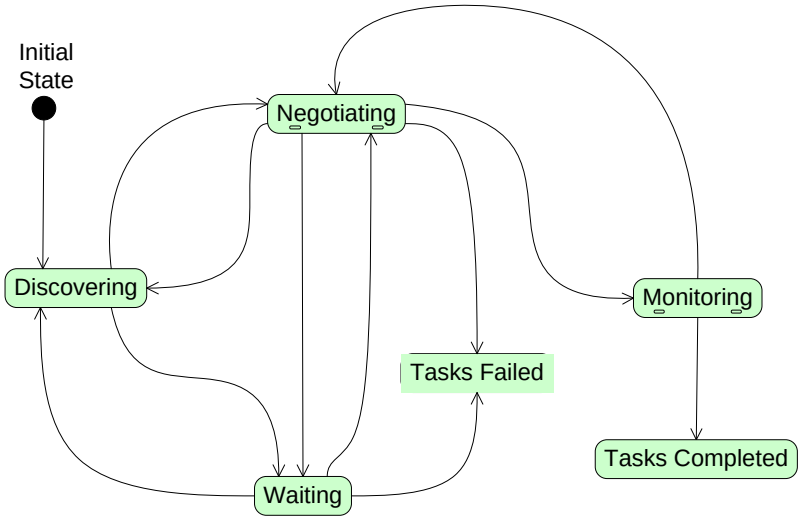

Figure 3.1. State model of grid computing system.

To convert the state model in Fig. 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 transitions from 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).

$$p_{ij} = \frac{f_{ij}}{\sum_{k=1}^{n} f_{ik}} \tag{3.1}$$

Repeating computation (1) for all $i, j = 1 \ldots n$ 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 Eq. (3.1) for each period. This subdivision enabled the Markov model to capture changes in system behavior over time, or to be considered as *piecewise homogenous* [41]. A time period duration of 2 h, or 7200 s was chosen as the duration of a subdivided time period. Thus, a simulated 8 h 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 Fig. 3.2(a). Following this, we repeated these observations for the large-scale grid simulation over a 640 h period, which resulted in 321 time period TPMs, summarized in Fig. 3.2(b). (Note: The complete set of 5 time period TPMs for the 8 h duration appear in [42], while the complete set of 321 TPMs can be obtained from [43] or upon request from the authors.)

(a)

	Initial	Wait	Disc	Ngt	Mon	Comp	Fail
Initial	0.9697	0	0.0303	0	0	0	0
Waiting	0	0.8363	0.0673	0.0918	0	0	0.0046
Disc	0	0.0355	0.6714	0.2931	0	0	0
Ngt	0	0.4974	0.0182	0.2882	0.1961	0	0.0001
Mon	0	0	0	0.0003	0.9917	0.0080	0
Comp	0	0	0	0	0	1.0	0
Fail	0	0	0	0	0	0	1.0

(b)

	Initial	Wait	Disc	Ngt	Mon	Comp	Fail
Initial	0.9997	0	0.0003	0	0	0	0
Wait	0	0.6292	0.0252	0.3441	0	0	0.0015
Disc	0	0.0766	0.6133	0.3101	0	0	0
Ngt	0	0.0378	0.0015	0.0637	0.8710	0	0.0259
Mon	0	0	0	0.0004	0.9883	0.0113	0
Comp	0	0	0	0	0	1.0	0
Fail	0	0	0	0	0	0	1.0

Figure 3.2 (a, b). Summary TPM for the grid computing system: (a) over 8 h duration (plus a 2 h clean-up period), and (b) over 640 h 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 h 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_{period} = 7200 s has $G = d_{period}/h$ or 85 time steps). To advance the system state one time step, a vector v_{m-1}, which represents the system state at time step *m-1*, is multiplied by the TPM $\mathcal{P}_{tp(m)}$ for the applicable time period *tp(m) =integral value ((m-1)/G)* + 1 to produce a new system state v_m. This operation is represented by Eq. (3.2) below,

$$v_m = v_{m-1} \cdot \mathcal{P}_{tp(m)}. \tag{3.2}$$

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, Eq. (3.2) is repeated for $G \times k$ time steps until the end

of the total simulated duration to produce the end state vector $v_{(G \times k)}$. The result of this process, which we refer to as a *Markov simulation* is shown in Fig. 3.3(a) for the 8h simulated duration and in Fig. 3.3(b) for 640 h duration which approached steady-state conditions. In Fig. 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.

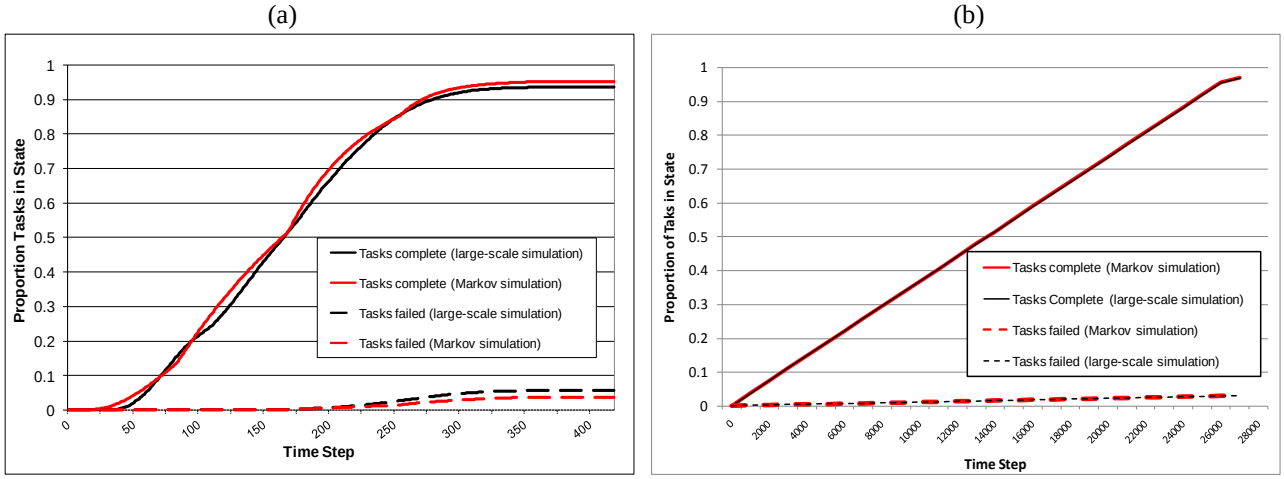

Figure 3.3 (a, b). Comparison between large-scale and Markov simulations of the change in the proportion of tasks completed and tasks failed in grid computing system for: (a) 8 h time duration (plus a 2 h clean-up period) in which Markov simulation covered 421, 85 s time steps; and (b) 640 h duration in which Markov simulation covered > 27000 time steps.

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 [40] that simulates the performance of a network using alternative congestion control algorithms. The large-scale model simulates in detail how different congestion control algorithms affect 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 Sec. 3.1 for the grid system Markov model. Figure 3.4 shows the Abilene system Markov model as a state diagram for a single data flow. 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 Avoidance* (NCA), *Alternate Congestion Avoidance* (ACA), and *Slow Start* (SS). Flows may re-enter any of these three states according to criteria described in [40] until they complete (or fail). Flows may fail from the *Connecting* state, NCA, or SS, which we do not show in the figure for the sake of simplicity. Like the grid system model, the Abilene system Markov model is an absorbing chain.

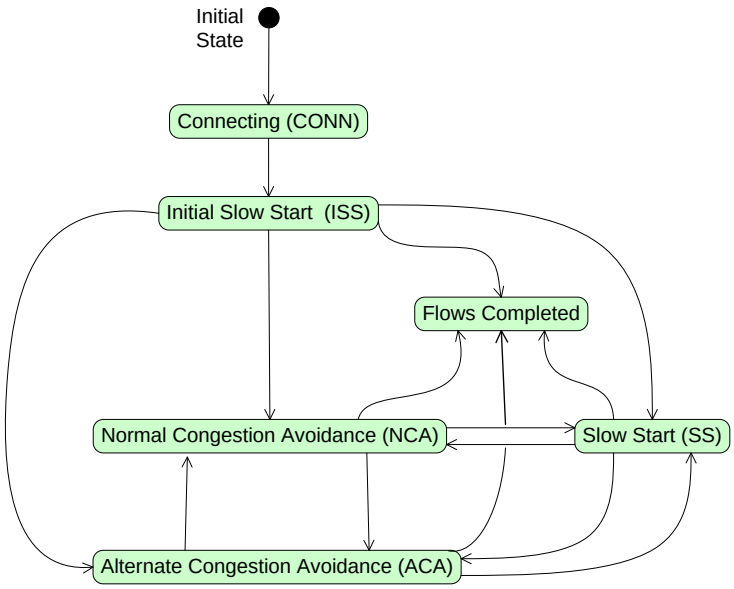

Figure 3.4. State model of Abilene network system

To obtain probabilities of transition, the procedure described in Sec. 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 piecewise homogenous by subdividing the 1500 s 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 Fig. 3.5. The Markov simulation of the Abilene system and its comparison with the related large-scale simulation is shown in Fig. 3.6.

	Initial	Conn	ISS	NCA	ACA	SS	Comp	Fail
Initial	0.999834	0.000167	0	0	0	0	0	0
Conn	0	0.816360	0.183574	0	0	0	0	0.000067
ISS	0	0	0.887686	0.029721	0.005570	0.006451	0.070572	0
NCA	0	0	0	0.963576	0.000132	0.014755	0.021526	0.000014
ACA	0	0	0	0.009739	0.853708	0.033566	0.102988	0
SS	0	0	0	0.096942	0.003969	0.794193	0.104896	0.000002
Comp	0	0	0	0	0	0	1.0	0
Fail	0	0	0	0	0	0	0	1.0

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).

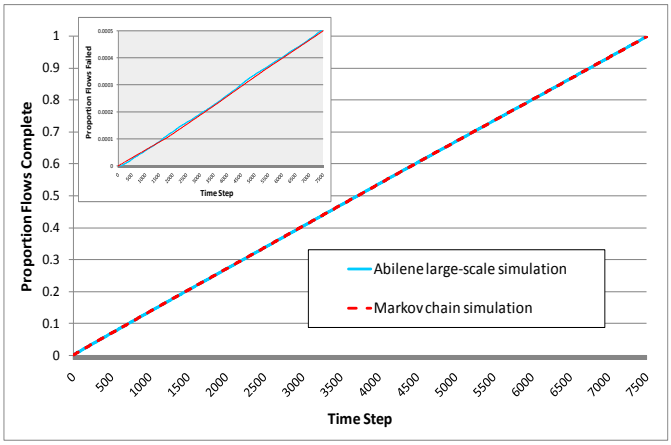

Figure 3.6. Comparison between Abilene network large-scale simulation and Markov simulation of comparative progress of flows completed and flows failed (see inset) over 1500 s duration. Results show that the Markov simulation provides a very close approximation of the large-scale simulation.

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 [7] 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. In this section, we provide an overview of the algorithm described in [7].

The perturbation algorithm executes 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 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* $c^{\uparrow}$. The primary increase column $c^{\uparrow}$ corresponds to a state being transitioned into from the state corresponding to the row *r*. During the procedure, the transition probability of $c^{\uparrow}$ is incrementally raised. To offset this increase, a row element corresponding to a different state is selected as a *primary sink column*, $c^{\downarrow}$. The row element $c^{\downarrow}$ is decreased by a portion of the increase to $c^{\uparrow}$, where the portion is determined by the weight $w \leq 1$. The remainder of the increase to $c^{\uparrow}$ is offset by decreasing any remaining non-zero elements of *r* by amounts that are proportional to their non-zero values. In this way the changes $c^{\uparrow}$, $c^{\downarrow}$, 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 $c^{\uparrow}$ 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*, $d^{\uparrow}$. The row element $d^{\uparrow}$ is raised by the amount m_{sec} to a predetermined maximum. In the perturbation of the secondary row, the decrease is distributed proportionally to all other row elements having non-zero transition probabilities. A set of values $\{r, c^{\uparrow}, c^{\downarrow}, w, s, d^{\uparrow}, m_{sec}\}$ 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 $c^{\uparrow}$ and $d^{\uparrow}$ are raised respectively. These decisions define the extent and granularity of the perturbation (note, v_s is used to determine m_{sec} mentioned above). The algorithm proceeds by enumerating all feasible perturbation combinations. For each combination, an iteration is performed to raise $c^{\uparrow}$ and $d^{\uparrow}$ 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 for which systematic changes to transition probabilities lead to performance degradations.

Fig. 3.7 illustrates an example of one such drastic performance degradation identified by the perturbation algorithm in both the 8 h and 640 h cases. These figures show the impact of perturbing a single combination

(blue curves) in which lowering the probability of transition to 0 for *Negotiating* → *Monitoring* causes the proportion of tasks completed to also fall 0. In both the 8 h and 640 h cases, the Markov simulation predictions are borne out when the large-scale simulation, which is also altered to behave aberrantly (red curve) so that negotiations that would otherwise succeed instead fail and task execution is prevented. In these experiments, the large-scale simulation served as a proxy for a real-world system.

(a) (b)

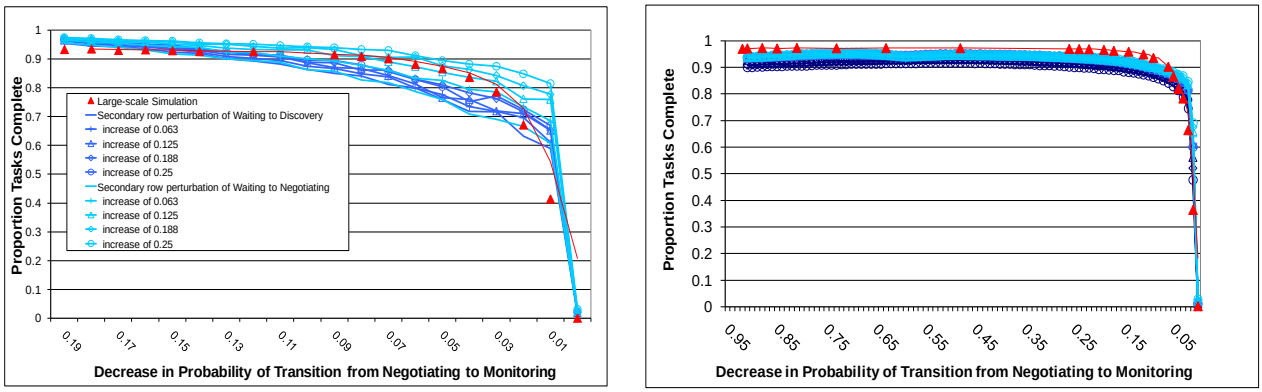

Figure 3.7 (a,b). Perturbation of *Negotiating* State (r=4) in grid system Markov chain model to predict effect of reducing probability of transition from *Negotiating* → *Monitoring* in (a) the 8 h case and (b) the 640 h case for the grid computing system. Proportion of tasks completed in the large-scale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

Although Fig. 3.7 shows that the decrease in proportion of tasks completed is a straight forward consequence of this perturbation, there is a less obvious persistence in the high rate of tasks completed as the probability of transition from *Negotiating* → *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 proportion of tasks completed declines sharply. This pattern is in fact predicted by the Markov simulation.

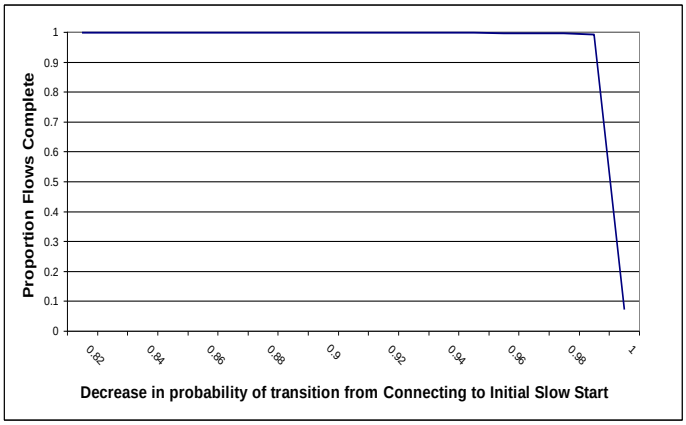

Figure 3.8. Perturbation of *Connecting* State (r=2) in the Abilene Markov chain model to predict effect of the reducing the probability of transition to 0 for *Connecting* → *Initial Slow Start* ($c^{\downarrow}$ = 3, w = 1), while raising probability of *Connecting* self-transition ($c^{\uparrow}$ = 2). No secondary perturbation row was chosen. The effect of this perturbation on proportion of flows completed is shown. Note: the probability of *Connecting* self-transition is increased to 1, making this a trap state situation as well.

For the Abilene network system, Fig. 3.8 shows a similar trend in the proportion of flows completed when the probability of transition from *Connecting* → *Initial Slow Start* is reduced to 0. Here, the perturbation also causes the proportion of 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* → *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 $O(ab^2(b-1))$ perturbation combinations. In this formula, b is a constant that indicates the expected number of $c^{\uparrow}$ (or $d^{\uparrow}$) column in the row being perturbed, or the *branching factor*. The constant a is the number of weight values that the primary sink element may take on. If L/v_r 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

$$E = O\,(L_r/v_r \cdot (L_s/v_s + 1) \cdot a(b^3 - b^2)), \tag{3.3}$$

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

$$E = O\,(L_r/v_r \cdot (L_s/v_s + 1) \cdot a(b^3 - 2b^2 - b)). \tag{3.4}$$

The complete exploration of a TPM having a order d is thus $O\,(d * E)$, 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 h scenario required approximately 56 min, while the 640 h scenario required 15.4 h. 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 h scenario, large-scale simulation needed about two orders of magnitude more time; in the 640 h 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 Sec. 5). For the Abilene system problem, execution of the perturbation algorithm to completely perturb rows 2 to 6 of the related Markov chain TPMs required 27.3 h, 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 h! In Sec. 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 might 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 report, in which we describe our work on two methods for this purpose. In Secs. 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 Sec. 7, we show how the Spectral Expansion Theorem can be used as an alternative to Markov simulations described earlier by the evaluation of analytical formulae. Further, 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 s-t cut sets between an initial state and desired absorbing state. The minimal s-t cut sets consist of critical state transitions, which if perturbed, are likely to lead to system performance degradations. In Sec. 5, we see that this method is capable of finding the same areas of performance degradation as the perturbation algorithm overviewed in Sec. 3, 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.

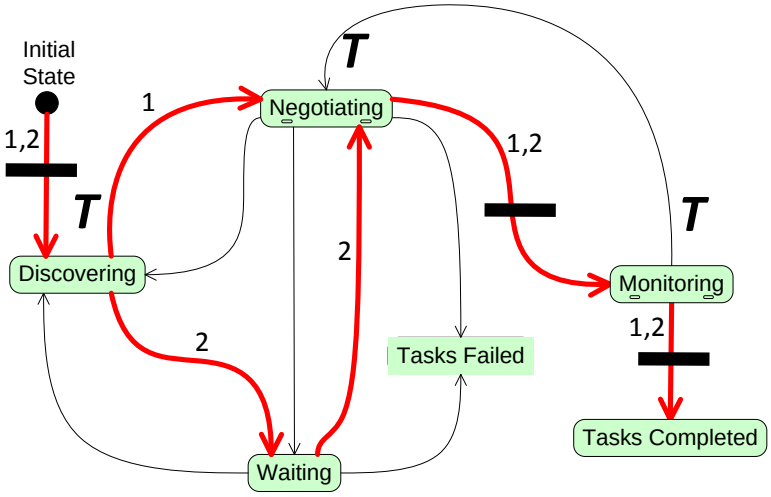

Figure 4.1. Two unique non-cyclic paths (numbered and denoted by thick arrows) from the *Initial State* to *Tasks Completed* state for grid computing system. Three single-transition s-t cuts appear as heavy bars over transitions. Trap states are denoted by ***T***.

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, Fig. 4.1 shows two paths through grid system Markov chain from the *Initial State* to the *Tasks Completed* state. For a small Markov chain, a well known algorithm for finding the shortest non-cyclic paths between two vertices in a graph, such as given in [38], 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 below). 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.

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 [32]. 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 Fig. 4.1, the state transition *Initial* → *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 report, 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 Fig. 4.1, there are three single-transition s-t cuts: *Initial* → *Discovering*, *Negotiating* → *Monitoring*, and *Monitoring* → *Tasks Completed*. Figure 3.7 (a, b) shows graphically the result of reducing the probability of transition for *Negotiating* → *Monitoring* to 0 in both the 8 h and 640 h 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* → *Discovering* and *Monitoring* → *Tasks Completed*, are similarly perturbed, as discussed further in [7]. As will be described in Sec. 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* → *Connecting* and *Connecting* → *Initial Slow Start*, shown in Fig. 4.2. Figure 3.8 shows the result of reducing the probability of transition for *Connecting* → *Initial Slow Start* to 0, which results in the proportion of flows reaching the *Flows Completed* state to fall to 0. Again, Sec. 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.

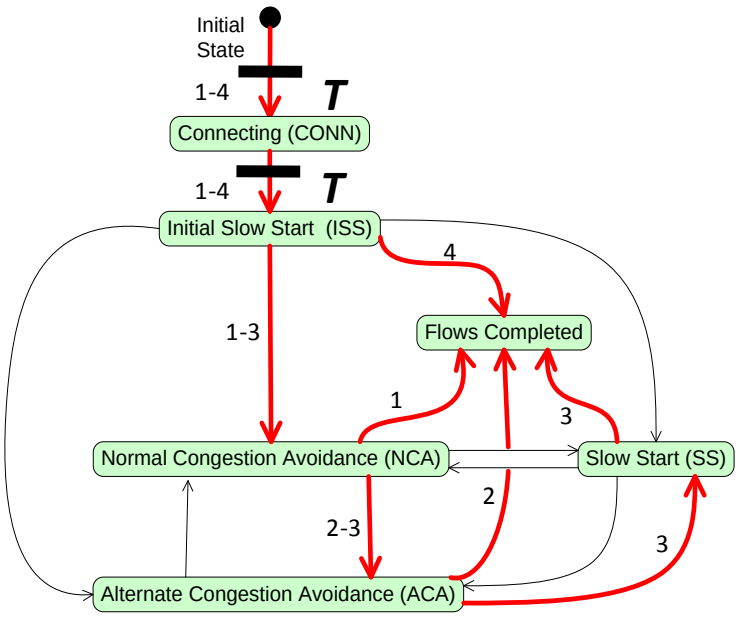

Figure 4.2. An example of 4 unique non-cyclic paths (denoted by thick arrows) from the *Initial State* to the *Flows Completed* state for the Abilene network system. There are 16 possible paths, of which 4 are shown. Two single-transition s-t cuts appear as heavy bars over transitions. Trap states are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

Thus for both the grid system and the Abilene System, it was possible to use a simple algorithm for enumerating all paths derived from [38] 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* → *Monitoring* and *Monitoring* → *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* → *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 a desired 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 "*to state*" state(s) which will receive increased probabilities of transition may be different from the "*from state*" or may be the same. In the latter case, when the "*to*" and "*from*" states are the same, the process remains in the same state, or merely transitions back to itself–which we refer to as a *self-transition*[2]. If a self-transition probability is raised so that it approaches 1, or even equals 1, the process remains in the "*from state*" for a prolonged time. In this case, the "*from state*" effectively becomes a *trap state* for processes. A *trap state* is distinguished from a permanent absorbing state, such as *Tasks Completed*, because the self-transition probability of a trap state may vary, while the self-transition probability of an absorbing state is always 1.0.

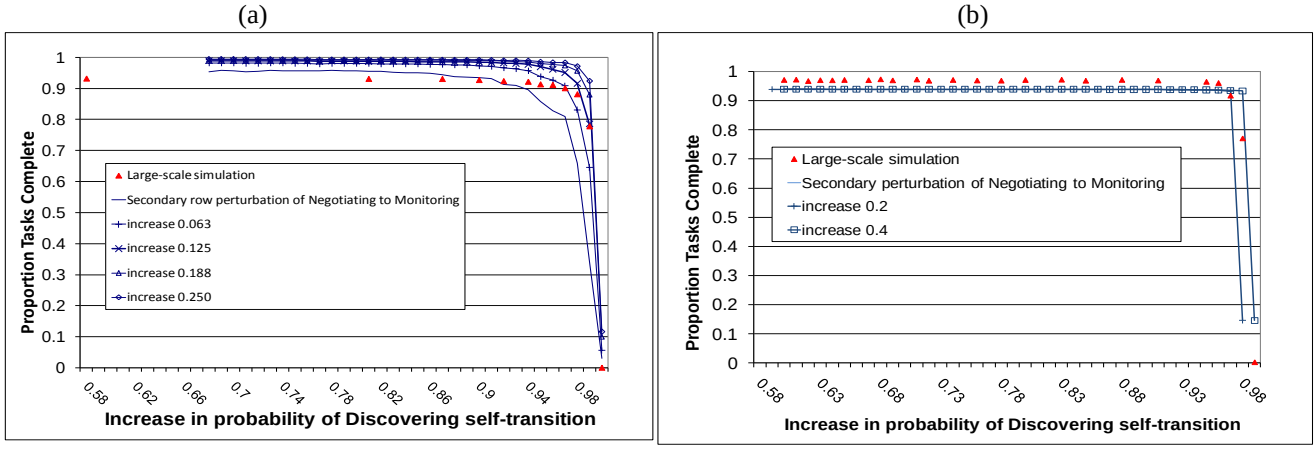

Figure 4.3 (a, b). Perturbation of *Discovering* State (*r= Discovering*) to predict effect of increasing probability of self-transition *Discovering* → *Discovering* ($c^{\uparrow}$ = *Discovering*) in (a) the 8 h case and (b) the 640 h case for the grid system model. Proportion of tasks completed in the large-scale and Markov 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_r = 0.01, L_s=0.25 , and v_s = 0.0625; for (b) L_r=0.75, v_r = 0.01, L_s=0.4 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 Fig. 4.3 when the *Discovering* state is made a trap state in the 8 h and 640 h cases. Tasks never leave the *Discovering* state, so that they cannot proceed to other states and finish. The perturbation described for the

[2] 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 → s_i, or f_{ii} is tabulated for processes whose duration in a state exceeds a time step.

Abilene system in Fig. 3.8 can also be accomplished by raising the *Connecting* self-transition to 1. 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 Completed* in the grid model or *Flows Completed* in the Abilene model). In the grid system model, the trap states are *Initial*, *Discovering*, *Negotiating*, and *Monitoring*, shown in Fig. 4.1; for the Abilene model, these are *Initial*, *Connecting*, and *Initial Slow Start*, shown in Fig. 4.2.

As Sec. 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 [7] but at less cost. However, the approach is insufficient for analysis of substantially larger Markov chains, as we describe in Sec. 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 report as a *multiple-transition s-t cut*. In such a minimal s-t 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.

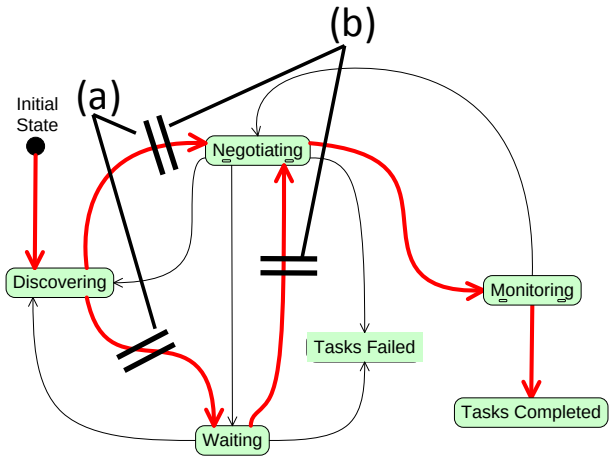

Figure 4.4. Two multiple-transition s-t cuts for the grid computing system: (a) *Discovering* → *Negotiating* and *Discovering* → *Waiting*; and (b) *Discovering* → *Negotiating* and *Waiting* → *Negotiating*.

In the Abilene system Markov chain, there are 8 possible multiple-transition s-t cuts listed in Table 4.1, one of which is shown as an example in Fig. 4.5. As in Fig. 4.1 and Fig. 4.2 for single-transition s-t cuts, it is easy to see that the two multiple-transition, minimal s-t cut sets in Fig. 4.4 disconnect the *Initial State* from the *Tasks Completed* state, while in Fig. 4.5, the sample multiple-transition minimal s-t cut set also disconnects the *Initial State* from the *Flows Completed* state.

Table 4.1. Complete list of 10 minimal s-t cut sets for Abilene system Markov chain. Two of these minimal s-t transition cut sets are single s-t transition cuts (1 and 2), while 8 are multiple-transition s-t cuts (3-10).

	Minimal s-t cut sets for Abilene Network System
1	Init→ Connected
2	Connected→ISS
3	ISS→NCA, ISS→ACA, ISS→SS, ISS→Flows Completed
4	ISS→Flows Completed, NCA→Flows Completed, ACA→Flows Completed, SS→Flows Completed
5	ISS→ACA, ISS→SS, ISS→Flows Completed, NCA→ACA, NCA→SS, NCA→Flows Completed
6	ISS→NCA, ISS→SS, ISS→Flows Completed, ACA→NCA, ACA→SS, ACA→Flows Completed
7	ISS→NCA, ISS→ACA, ISS→Flows Completed, SS→NCA, SS→ACA, SS→Flows Completed
8	ISS→ACA, ISS→Flows Completed, NCA→ACA, NCA→Flows Completed, SS→ACA, SS→Flows Completed
9	ISS→SS, ISS→Flows Completed, NCA→SS, NCA→Flows Completed, ACA→SS, ACA→Flows Completed
10	ISS→NCA, ISS→Flows Completed, ACA→NCA, ACA→Flows Completed, SS→NCA, SS→Flows Completed

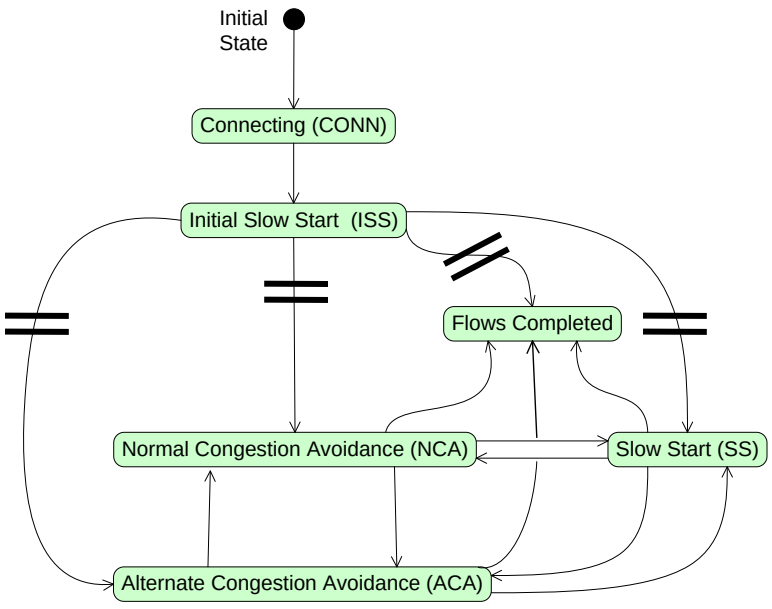

Figure 4.5. Example of multiple-transition, minimal s-t cut set for the Abilene network system

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* [44]. 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* → *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 Sec. 6, we provide an algorithm for finding multiple-transition s-t cuts that is appropriate for large Markov chains (which was actually used to compute the multiple-transition s-t cuts in Fig. 4.4 and Fig. 4.5). But first, in Sec. 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 State* and *Tasks Completed* states 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 in the Abilene system Markov chain model against the results produced by the perturbation algorithm. The conclusions are verified by showing the correspondence between specific minimal s-t cut sets and state transitions which, 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 problems, minimal s-t cut sets could be found either through the path enumeration algorithm adapted from [38] discussed earlier or through the node contraction algorithm, which will be described in Sec. 6.

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 state transitions of interest. Where appropriate, accentuating or mitigating effects of secondary row perturbation are discussed (see [7] 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 State* and *Tasks Completed* states in the grid system Markov chain correspond to state transitions, which if suitably perturbed using the perturbation algorithm described in Sec. 3, can adversely impact system performance. Table 5.1 shows the results of the application of the perturbation algorithm overviewed in Sec. 3.1 to the grid system Markov model for the 8 h grid system case, while Table 5.2 shows the same for the 640 h case. Specifically, the tables show the results of perturbing rows, $\boldsymbol{r}$, corresponding to the states *Waiting*, *Discovering*, *Negotiating*, and *Monitoring*, by raising the probability of transition from the states designated as $\boldsymbol{r}$ to states that correspond to primary increase columns $c^{\uparrow}$, while lowering the probability of transition from, $\boldsymbol{r}$, to other states that are designated as sink columns, $c^{\downarrow}$. Both tables show that in all cases where application of the perturbation algorithm causes declines in the proportion of tasks completed that approach 100% (i.e., the proportion of tasks completed approaches 0[3]), 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 State* and *Tasks Completed* states in Fig. 4.1 correspond to results produced by the perturbation algorithm in which the proportion of tasks completed is reduced to 0. The first case occurs when r = *Negotiating*. Here, Tables 5.1 (c) and 5.2 (c) show that designating *Monitoring* as the sink column $c^{\downarrow}$, i.e., lowering the probability of *Negotiating* → *Monitoring* to 0, resulted in the proportion of tasks completed approaching 0 (a percentage decline that approaches 100%), 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.,

[3] 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 completed also approaches, but does not equal, 0

made the primary increase column, $c^{\uparrow}$. In this case, Fig. 4.1 shows that the state transition *Negotiating* → *Monitoring* is a single-transition s-t cut.

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

(a) ***r*** = *Discovering*

Sink column ($\boldsymbol{c}^{\downarrow}$)	Primary increase column ($\boldsymbol{c}^{\uparrow}$)	Prop. *Tasks Completed* and % change when prob. ($r \rightarrow \boldsymbol{c}^{\downarrow}$) → *0*		Single-transition s-t cut exists
Waiting	*Discovering*	0.957	+0.07	No
Waiting	*Negotiating*	0.959	+0.42	No
Discovering	*Waiting*	0.939	-1.50	No
Discovering	*Negotiating*	0.963	+0.88	No
Negotiating	*Waiting*	0.894	-6.39	No
Negotiating	*Discovering*	0.651	-32.05	No

(b) ***r*** = *Waiting*

Sink column ($\boldsymbol{c}^{\downarrow}$)	Primary increase column ($\boldsymbol{c}^{\uparrow}$)	Prop. *Tasks Completed* and % change when prob. ($r \rightarrow \boldsymbol{c}^{\downarrow}$) → *0*		Single-transition s-t cut exists
Waiting	*Discovering*	0.974	+2.05	No
Waiting	*Negotiating*	0.981	+2.59	No
Discovering	*Waiting*	0.937	-1.69	No
Discovering	*Negotiating*	0.963	+0.59	No
Negotiating	*Waiting*	0.818	-14.55	No
Negotiating	*Discovering*	0.939	+1.70	No

(c) ***r*** = *Negotiating*

Waiting	*Discovering*	0.974	+1.83%	No
Waiting	*Negotiating*	0.985	+3.03%	No
Waiting	*Monitoring*	1.000	+4.53%	No
Discovering	*Waiting*	0.954	+0.09%	No
Discovering	*Negotiating*	0.957	+0.11%	No
Discovering	*Monitoring*	0.967	+1.22%	No
Negotiating	*Waiting*	0.923	-3.63%	No
Negotiating	*Discovering*	0.941	-1.48%	No
Negotiating	*Monitoring*	0.988	+3.23%	No
Monitoring	*Waiting*	***0.000***	-99.98%	**Yes**[a]
Monitoring	*Discovering*	***0.000***	-99.98%	**Yes**[a]
Monitoring	*Negotiating*	***0.000***	-99.98%	**Yes**[a]

(d) ***r*** = *Monitoring*

Negotiating	*Monitoring*	0.982	+2.94%	No
Negotiating	*Tasks Comp*	0.982	+3.04	No
Monitoring	*Negotiating*	***0.028***	-97.04%	**Yes**[b, *]
Monitoring	*Tasks Comp*	0.980	+2.84	No
Tasks Comp	*Negotiating*	***0.001***	-99.93%	**Yes**[b]
Tasks Comp	*Monitoring*	***0.002***	-99.83%	**Yes**[b]

(e) ***r*** = *Initial*

Discovering	*Initial*	***0.0***	***-100.00***	Yes[c]
Initial	*Discovering*	0.970	+1.57	No

In Fig. 4.1, transition corresponds to
[a]*Negotiating* to *Monitoring* single-transition s-t cut
[b]*Monitoring* to *Completed* single-transition s-t cut
[c]*Initial* to *Discovering* single-transition s-t cut

* Note that that the perturbation described in this row occurs even though the transition from *Monitoring* → *Negotiating* is not directly perturbed. See text above for the explanation.

Similarly, when r = *Monitoring*, Tables 5.1 (d) and 5.2 (d) show that designating *Tasks Completed* as the sink column $c^{\downarrow}$, i.e., lowering the probability of *Monitoring* → *Tasks Completed* to 0, creates the obvious result where the proportion of tasks completed 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, $c^{\uparrow}$. Again, Fig. 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 $c^{\downarrow}$), while raising the transition *Monitoring* → *Negotiating*, also resulted in the proportion tasks completed approaching 0. This happens because the probability of transition for *Monitoring* → *Tasks Completed* is initially very low (0.008 for the 8 h case and 0.009 for the 640 h case. See Fig. 3.2. Thus, the probability of *Monitoring* self-transition must be very high (over 0.99 in both parts of Fig. 3.2) to ensure that 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 application of the perturbation algorithm results in the proportion of tasks completed falling to 0 and existence of single-transition s-t cuts in 640 h grid system simulation. The table shows the proportion of tasks completed and percent change when the perturbation algorithm is applied to rows, ***r***, of the TPM in Fig. 3.2 (b) to decrease the probability of transition to 0 for the sink column, $\boldsymbol{c}^{\downarrow}$, or the state transition $\boldsymbol{r} \rightarrow \boldsymbol{c}^{\downarrow}$, while increasing the probability of transition in the primary increase column, $\boldsymbol{c}^{\uparrow}$, or $\boldsymbol{r} \rightarrow \boldsymbol{c}^{\uparrow}$ with a sink weight = 1. Secondary perturbation is excluded. The right-most column indicates if single-transition s-t cut exists for $\boldsymbol{r} \rightarrow \boldsymbol{c}^{\downarrow}$ in Fig. 4.1. In all cases where perturbation causes the proportion of tasks completed to approach 0, a positive correspondence exists with a single-transition s-t cut. Shaded cells represent perturbations where changes to the proportion of tasks completed were verified by large-scale simulation. Note: trap states are excluded.

(a) ***r*** = *Discovering*

Sink column ($\boldsymbol{c}^{\downarrow}$)	Primary increase column ($\boldsymbol{c}^{\uparrow}$)	Prop. *Tasks Completed* and % change when prob. ($r \rightarrow \boldsymbol{c}^{\downarrow}$) $\rightarrow 0$		Single-transition s-t cut exists
Waiting	*Discovering*	0.935	-0.04%	No
Waiting	*Negotiating*	0.935	+0.01%	No
Discovering	*Waiting*	0.935	-0.07%	No
Discovering	*Negotiating*	0.935	+0.03%	No
Negotiating	*Waiting*	0.933	-0.23%	No
Negotiating	*Discovering*	0.932	-0.31%	No

(b) ***r*** = *Waiting*

Sink column ($\boldsymbol{c}^{\downarrow}$)	Primary increase column ($\boldsymbol{c}^{\uparrow}$)	Prop. *Tasks Completed* and % change when prob. ($r \rightarrow \boldsymbol{c}^{\downarrow}$) $\rightarrow 0$		Single-transition s-t cut exists
Waiting	*Discovering*	0.937	+0.22%	No
Waiting	*Negotiating*	0.939	+0.38%	No
Discovering	*Waiting*	***0.934***	***-0.13%***	No
Discovering	*Negotiating*	0.936	+0.03%	No
Negotiating	*Waiting*	0.843	-9.82%	No
Negotiating	*Discovering*	0.932	-0.30%	No

(c) ***r*** = *Negotiating*

Waiting	*Discovering*	0.937	+0.19%	No
Waiting	*Negotiating*	0.938	+0.28%	No
Waiting	*Monitoring*	0.939	+0.37%	No
Discovering	*Waiting*	0.935	+0.00%	No
Discovering	*Negotiating*	0.935	+0.00%	No
Discovering	*Monitoring*	0.936	+0.07%	No
Negotiating	*Waiting*	0.931	-0.44%	No
Negotiating	*Discovering*	0.933	-0.27%	No
Negotiating	*Monitoring*	0.938	+0.35%	No
Monitoring	*Waiting*	***0.000***	-99.99%	**Yes**[a]
Monitoring	*Discovering*	***0.000***	-99.99%	**Yes**[a]
Monitoring	*Negotiating*	***0.000***	-99.98%	**Yes**[a]

(d) ***r*** = *Monitoring*

Negotiating	*Monitoring*	0.937	+0.24%	No
Negotiating	*Tasks Compl'd*	0.938	+0.27%	No
Monitoring	*Negotiating*	***0.186***	***-80.13%***	**Yes**[b, *]
Monitoring	*Tasks Compl'd*	0.949	+1.52%	No
Tasks Compl'd	*Negotiating*	***0.006***	***-99.40%***	**Yes**[b,]
Tasks Compl'd	*Monitoring*	***0.016***	***-98.32%***	**Yes**[b]

(e) ***r*** = *Initial*

Discovering	*Initial*	***0.0***	***-100.00***	Yes[c]
Initial	*Discovering*	0.970	+1.96	No

In Fig. 4.1, corresponds to
[a] *Negotiating* to *Monitoring* single-transition s-t cut
[b] *Monitoring* to *Completed* single-transition s-t cut
[c] *Initial* to *Discovering* single-transition s-t cut

* Note that that the perturbation described in this row occurs even though the transition from *Monitoring* → *Negotiating* is not directly perturbed. See text above for the explanation.

Finally, there were situations where the proportion of tasks completed 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, $c^{\uparrow}$, while *Negotiating* was made the sink column, $c^{\downarrow}$. Here the combined probabilities of transition of matrix elements $c^{\uparrow}$ and $c^{\downarrow}$ approached, but did, not equal 1. Hence raising $c^{\uparrow}$ to the limit while lowering $c^{\downarrow}$ 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. Note that these multiple-transition s-t cuts were identified by the node contraction algorithm to be described below, but not by the path enumeration algorithm or by the perturbation algorithm described in Sec. 3.1.

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 h and 640 h 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, the proportion of tasks completed approaches 0. The fourth state,

Waiting, is not a trap state; but it lies on 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.

Table 5.3. Identification of trap states in 8 h and 640 h grid system. The table shows the proportion of tasks completed by the Markov simulation for states in Fig. 3.1 when their probability of self-transition is raised to 1. The states where the proportion of tasks completed falls to 0 correspond to trap states. Shaded cells indicate verification by the large-scale simulation.

State for which probability of self-transition is raised to 1	***Proportion Flows Complete***		***Corresponds to Trap State***
	8-hour	*640-hour*	
Waiting	***0.400***	***0.756***	No
Discovering	***0.030***	***0.019***	Yes
Negotiating	0.018	0.045	Yes
Monitoring	0.000	0.000	Yes

5.1.3 Efficiency of Minimal s-t Cut Set Analysis in the Grid System Case

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 Fig. 4.1 correspond to cases where the perturbation algorithm found perturbation combinations that caused the proportion of tasks completed to approach 0. For all other perturbation combinations shown in Tables 5.1 and 5.2, the proportion of tasks completed remained relatively stable (i.e., did not decline significantly).

Using 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 identified single-transition s-t cuts, trap states and minimal multiple-transition s-t cut sets, it was then desirable to apply the perturbation algorithm to the corresponding TPM rows to verify these conditions and document the drop in the proportion of tasks completed. To do this for the three single-transition s-t cuts shown in Fig. 4.1 requires systematically lowering one transition probability in each row with a weight of 1.0—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 h case, this entailed about 71 s of execution time. Reducing the two two-transition cut sets in Fig. 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 min required for the original complete perturbation. For the 640 h case, the time to perform the same perturbations was about 230 s, or about 1.4% of the total 4.5 h. All experiments were executed on a Dell PowerEdge 6950 with quad, dual-core 3.0GHz processors and 32GB memory, running under Windows 2003. The results described above for the grid system model (as well as the Abilene system described below) 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 described in [7], which itself was found to constitute a two-order magnitude improvement over large-scale simulation [7].

Table 5.4. 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 [7] and (c) large-scale simulation [1].

		Minimal s-t cut set Identification		Exhaustive search of TPM rows with perturbation algorithm	Large-scale simulation
		Path Enumeration Algorithm[a]	Perturbation of individual state transitions only		
Grid Computing System	8-hour	<0.01 s	244 s	56 minutes	205 hours
	640-hour	<0.01 s	230 s	4.5 hours	870 hours
Abilene Network		<0.01 s	450 s	27.3 hours	Not available

[a]Note: results achieved by path enumeration were also achieved using the node contraction algorithm to be described in the Sec. 6.

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 Fig. 3.7 or Fig. 3.8.

5.2 Abilene System

In this section, we verify that all minimal s-t cut sets for paths between the *Initial State* and *Flows Completed* states in the Abilene system Markov chain model correspond to state transitions, which if perturbed using the perturbation algorithm as described in Sec. 3, can adversely impact system performance. 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 Avoidance* (*NCA*), *Alternate Congestion Avoidance* (*ACA*), and *Slow Start* (*SS*) to other states is lowered to 0 (i.e., designated as sink columns $c^{\downarrow}$). At the same time probabilities of transitions to other states are raised (i.e., designated as primary increase columns $c^{\uparrow}$). 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 h 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* (i.e., $\boldsymbol{r}$ = *Connecting* and *Initial Slow Start* is made $c^{\downarrow}$), when lowered to 0, causes the proportion of flows completed to approach 0. Fig. 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 one 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 that adversely impact system performance, when probability of transition is perturbed to fall to 0.

Table 5.5. Correspondence between cases where perturbation algorithm results in proportion of tasks completed falling to 0 and existence of single-transition s-t cuts in the Abilene system Markov chain model. The table shows the proportion of tasks completed and percent change when perturbation algorithm is applied to rows, ***r***, of the TPM in Fig. 3.5 to decrease the probability of transition to 0 for the sink column, $\boldsymbol{c}^{\downarrow}$, or the state transition $\boldsymbol{r} \rightarrow \boldsymbol{c}^{\downarrow}$, while increasing the probability of transition in the primary increase column, $\boldsymbol{c}^{\uparrow}$, or $\boldsymbol{r} \rightarrow \boldsymbol{c}^{\uparrow}$ with a sink weight = 1. Secondary perturbation is excluded. The right-most column indicates if a single-transition *s-t* cut exists for $\boldsymbol{r} \rightarrow \boldsymbol{c}^{\downarrow}$ in Fig. 4.2. In all cases where perturbation causes proportion of tasks completed to approach 0, a positive correspondence exists with a single-transition s-t cut. Note: trap states are discussed separately below.

(a) ***r*** *= Initial Slow Start (ISS)*

Sink column ($\boldsymbol{c}^{\downarrow}$)	Primary increase column ($\boldsymbol{c}^{\uparrow}$)	Prop. *Tasks Complete*	% change when prob. ($r \rightarrow \boldsymbol{c}^{\downarrow}$) $\rightarrow 0$	single-transition s-t cut exists
ISS	*NCA*	0.997	-0.20	No
ISS	*ACA*	0.998	-0.10	No
ISS	*SS*	0.997	-0.20	No
NCA	*ISS*	0.999	0.00	No
NCA	*ACA*	0.998	-0.10	No
NCA	*SS*	0.998	-0.10	No
ACA	*ISS*	0.999	0.00	No
ACA	*NCA*	0.998	-0.10	No
ACA	*SS*	0.998	-0.10	No
SS	*ISS*	0.999	0.00	No
SS	*NCA*	0.998	-0.10	No
SS	*ACA*	0.998	-0.10	No

(b) ***r*** *= Normal Congestion Avoidance (NCA)*

Sink column ($\boldsymbol{c}^{\downarrow}$)	Primary increase column ($\boldsymbol{c}^{\uparrow}$)	Prop. *Tasks Complete*	% change when prob. ($r \rightarrow \boldsymbol{c}^{\downarrow}$) $\rightarrow 0$	single-transition s-t cut exists
NCA	*ACA*	0.999	0.00	No
NCA	*SS*	0.999	0.00	No
ACA	*NCA*	0.999	0.00	No
ACA	*SS*	0.999	0.00	No
SS	*NCA*	0.999	0.00	No
SS	*ACA*	0.999	0.00	No

(c) ***r*** *= Alternate Congestion Avoidance (ACA)*

Sink column ($\boldsymbol{c}^{\downarrow}$)	Primary increase column ($\boldsymbol{c}^{\uparrow}$)	Prop. *Tasks Complete*	% change when prob. ($r \rightarrow \boldsymbol{c}^{\downarrow}$) $\rightarrow 0$	single-transition s-t cut exists
NCA	*ACA*	0.999	0.00	No
NCA	*SS*	0.999	0.00	No
ACA	*NCA*	0.998	-0.10	No
ACA	*SS*	0.998	-0.10	No
SS	*NCA*	0.998	-0.10	No
SS	*ACA*	0.999	0.00	No

(d) ***d*** *= Slow Start (SS)*

Sink column ($\boldsymbol{c}^{\downarrow}$)	Primary increase column ($\boldsymbol{c}^{\uparrow}$)	Prop. *Tasks Complete*	% change when prob. ($r \rightarrow \boldsymbol{c}^{\downarrow}$) $\rightarrow 0$	single-transition s-t cut exists
NCA	*ACA*	0.999	0.00	No
NCA	*SS*	0.999	0.00	No
ACA	*NCA*	0.998	-0.10	No
ACA	*SS*	0.998	-0.10	No
SS	*NCA*	0.998	-0.10	No
SS	*ACA*	0.999	0.00	No

(e) ***d*** *= Connected (Conn)*

Sink column ($\boldsymbol{c}^{\downarrow}$)	Primary increase column ($\boldsymbol{c}^{\uparrow}$)	Prop. *Tasks Complete*	% change when prob. ($r \rightarrow \boldsymbol{c}^{\downarrow}$) $\rightarrow 0$	single-transition s-t cut exists
ISS	*Connected*	0.071	-92.89	Yes[a]
Connected	*ISS*	0.999	-0.10	No

[a]In Figure 4.2, corresponds to *Connected* $\rightarrow$ *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*. These are the only states that result in the proportion of flows completed approaching 0 when their self-transition probabilities are raised to 1.0. 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 also predicted through path enumeration. 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 the proportion of 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 Avoidance*, *Alternate Congestion Avoidance*, and *Slow Start*—a smaller reduction in flows completed occurs that is relatively proportional to the chances that a flow enters the 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. The table shows proportion of flows completed by the simulation for states in the Abilene model when their probability of self-transition is raised to 1. The states where the proportion of flows completed approaches 0 correspond to trap states.

State for which probability of self-transition is raised to 1	***Proportion Flows Complete***	***Corresponds to Trap State***
Connected	0.008	Yes
Initial Slow Start	0.007	Yes
Normal Congestion Avoidance	0.824	No
Alternate Congestion Avoidance	0.967	No
Slow Start	0.918	No

The path enumeration algorithm identified 8 minimal s-t cut sets that were multiple-transition s-t cuts in the Abilene model. These are listed in Table 4.1, one of which is shown in Fig. 4.5. Simultaneously lowering the transition probabilities to 0 of all state transitions identified by these cut sets does indeed cause the proportion of 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 4.1, 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 of the minimal s-t cut set generation approach when compared to the perturbation algorithm overviewed in Sec. 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 *Connecting*→*Initial Slow Start*, this involves just one perturbation combination, which ran about 13 s. With regard to 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 4.1 involved roughly 10-20 s each. These results are summarized in Table 5.4. As in the case of the grid system, these computations required a very small fraction of the 27.3 h that were needed to apply the perturbation algorithm to all rows of the Abilene Markov chain TPMs.

6. Application of Minimal s-t Cut Set Identification to 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 earlier sections 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 [45, 46]. 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 more work remains to be done, they do provide evidence 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 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 identifies single-transition s-t cuts and trap states. However, the algorithm also identifies 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 *Flows Completed*). The set of state transitions in such a minimal s-t cut set will represent related circumstances in the domain being modeled, or may represent unrelated events which could randomly occur together. We assume that in most large domains, the most critical cut sets will have a small number of transitions, since small combinations are more likely, as discussed in [47]. Therefore, a reasonable goal would be to generate minimal s-t cut sets consisting of a limited number of state transitions. It is also 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 directed graphs [33, 35, 48]. All require considerable computational effort for large graphs. For instance, in [33], enumeration of all s-t cut sets was found to have a complexity of $O\,|E|$ per cut set listed. However, this algorithm can be computationally expensive as well, since Markov chains with as few as fifty states can contain over 10^8 minimal s-t cut sets on paths between the initial and absorbing states, as we will show 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 [34]. 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 minimal and near minimal s-t cut sets in a sample of larger Markov chain problems. Section 6.4 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 State* 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 for the proportion of tasks completed, such as are shown in Fig. 3.7 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 examples 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 Secs. 4 and 5. Then we apply the node contraction algorithm to a much larger example problem to see what these results look like. In Sec. 6.3, we provide a more extended analysis of the application of the algorithm to a set of larger problems 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 Fig. 4.1 and two multiple-transition s-t cuts that appear in Fig. 4.4. This algorithm also found 10 minimal s-t 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. In Sec. 5, the single- and multiple-transition cuts were shown to correspond exactly to state 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 [49]. 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 the TPM appears in Fig. 6.1 (with a more detailed view of the first 50 states of the TPM in Appendix B.3). This TPM is a sparse matrix in which processes proceed by following state transitions roughly along the matrix diagonal (dark gray cells) 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.

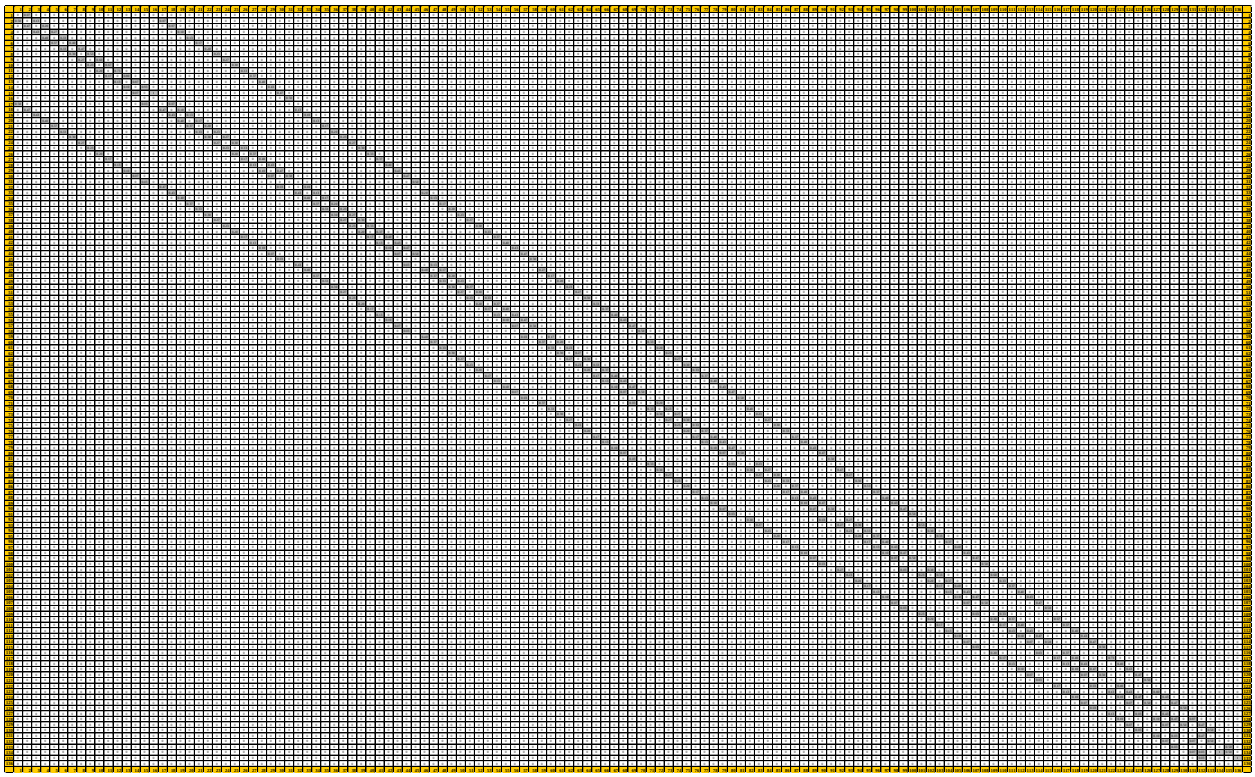

Figure 6.1. Compressed view of TPM rw136 for a Markov chain of 136 states from [49]. The horizontal axis represents the numbered states in ascending order from left to right; the vertical axis represents the states in descending order from bottom to top. Dark gray cells represent non-zero values. The pattern of shaded cells clearly shows a sparse diagonal matrix. For a more detailed view of the first 50 states of this TPM, see Appendix B.3.

This problem is of sufficient size and complexity that it effectively prohibits 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 path enumeration algorithm quickly confirms that there are at least two paths having no common states (other than the initial state or the single absorbing state); hence, no 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 were 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 [7] and overviewed in Sec. 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 executing the node contraction algorithm for 200 repetitions. These cut sets represent low combinations of number of *from* states and state transitions, and low sums of transition probabilities for cut set members. Each was verified as a minimal s-t cut set using the Markov simulation program.

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

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 disconnect the graph and cause the proportion of processes that reach the absorbing state to fall to 0, we first executed the Markov simulation program on the 136 × 136 TPM and modeled the evolution of the 136-element state vector, using the procedure overviewed in Sec. 3 and described in detail in [7]. By modifying the Markov perturbation algorithm to simultaneously lower a set of transition probabilities for state transitions to 0 in any of the cut sets in Table 6.1, it was possible to model the effect of these reductions on the flow of processes into the designated absorbing state, state 134.

6.3 Performance of the Node Contraction Algorithm on Four Larger Markov Chain Problems

In this section we apply minimal s-t cut set analysis using node contraction to a set of large Markov chain problems. 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.

Investigation of 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 implemented the minimal s-t cut set enumeration algorithm of [33], which enumerates all s-t cut sets in a directed graph. 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 [33], 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 [50] and are based on [51] and [52] respectively. Matrix 3 was a 50 × 50 subset of a very large 136 ×136 Markov chain matrix described in [49]. Matrix 4 was generated using [50] and is based on [53]. Each of these matrices was originally an ergodic Markov chain that was modified to become an absorbing chain 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 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 criterion 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 [47], 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 edges in the cut set 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.

We 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 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 have greater impact on system behavior. No doubt further research is necessary to investigate criteria for choosing 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 s-t 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 enumeration algorithm of [33] 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 enumerated minimal s-t cut sets and the time required to compute the enumeration. 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 enumerated by the algorithm of [33] 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. At 10 000 repetitions, the node contraction generated 77.2% (variance 555.2) of the top 100 ranked cut sets in 0.14% of the time for Sorts A-C. At 100 000 repetitions, node contraction generated 91.4% (variance 432.0) in 1.3% of the time

Matrix		**Minimal s-t cut sets enumerated using algorithm of [33]**		**Proportion (in %) of 100 top-ranked minimal s-t cut sets ranked by criteria A, B and C, which were found by the node contraction algorithm**											
		Number of minimal s-t cut sets	Time	After 1000 repetitions				After 10,000 repetitions				After 100,000 repetitions			
				Time	Sort A	Sort B	Sort C	Time	Sort A	Sort B	Sort C	Time	Sort A	Sort B	Sort C
1	50	530,432	332.1 s	63 s	56	67	22	640 s	80	100	96	---	---	---	---
2	50	28,230,288	21.6 hours	17 s	49	58	36	171 s	93	98	65	1710 s	99	100	99
3	50	27,242,634	36 hours	22 s	48	86	87	218 s	67	100	100	2288 s	88	100	100
4	40	422,060,801	156.1 hours	11s	15	30	22	106 s	30	80	62	1051 s	37	100	100

With the exception of Matrix 1, Table 6.2 shows that the node contraction algorithm generated as much as 91.4% of the top 100 ranked cut sets that were generated by the enumeration algorithm of [33] in 1.3% of the time needed (for 100 000 repetitions under all three sorts).For instance, for Matrices 2 and 3, the 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 enumeration algorithm of [33]. 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 min (as opposed to 156.1 h by the algorithm of [33]). 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 algorithm of [33], before it began to produce a large percentage of highly-ranked cut sets (hence, we did not attempt to use the node contraction algorithm at the highest level of effort). 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 to 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

In Secs. 3–6, we have discussed four methods of 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 [7, 42] and overviewed Sec. 3.1 (note: a second perturbation algorithm is discussed in the next section and presented in Appendix C),
2. A algorithm, such as discussed in Sec. 4 to enumerate all paths between an initial state and an absorbing state of interest and select state transitions common to all paths,
3. An algorithm that enumerates all minimal s-t cut sets between an initial state and an absorbing state of interest [33, 35, 48],
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 cut set 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 that described in Sec. 3 can be used directly as in [7]. 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. In the next section, we will present an additional approach in which eigendecomposition is used to identify areas on a Markov chain TPM that are sensitive to perturbation. This fifth approach is intended to be used in a complementary way with the four methods discussed above.

With regards to future work on graph-theoretic approaches, 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 Sec. 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 [34]. Along these lines, it may be possible to combine node contraction with lumping techniques, mentioned

earlier in the section on previous work [10–12], to reduce problem size. By selectively lumping vertices (states) into clusters, in such a 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 node contraction algorithm or such as that of [54] and others on graph division. As mentioned earlier, it is also 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.

Finally, one may also explore other approaches that can be used to generate minimum weight cut sets between two vertices in a directed graph which do not involve Markov chain analysis. These include, for instance, adaptation of methods for analysis of network flows [37–39], as mentioned earlier. Here, network flow theory is used to rank cut sets by their nearness to maximum flow and minimum capacity, rather than the criteria discussed earlier. To enable such rankings, the work of [55, 56] could be adapted.

7. Theoretical Model of a Markov Chain

The compressed representation of the large-scale grid system as a 7-state Markov chain makes it possible to search for critical pathways to system failure by examining the paths in the graph induced by the state transitions. In this section we will discuss an approach based on direct examination of the elements of the transition probability matrices. Our discussion will yield methods that produce results that are consistent with results of the methods in the previous sections and thus are, in some sense, complimentary to them. In particular, we will derive a very good approximation of the proportion of tasks completed based on the eigendecomposition of the TPMs. We will call this approximation the theoretical model.

7.1 Eigendecomposition of an Absorbing Markov Chain

As mentioned in Sec. 3.1, the state of the system at time step m can be described in terms of a row vector v_m of length seven, where the entry v_i gives the proportion of tasks in state s_i at that time step. Now since there are 85 time steps per time period, at time step m, the system is in time period $tp(m) := [(m-1)/85 + 1]$. Then the following equation determines the evolution of the Markov system at time period m,

$$v_m = v_{m-1} \cdot \mathcal{P}_{tp(m)} \tag{7.1}$$

where $\mathcal{P}_{tp(m)}$ is the TPM corresponding to time period $tp(m)$. Recall that row i in this matrix corresponds to the probabilities of transition of tasks from state s_i to some state s_j. Note that at every time step, the tasks in state s_i must transition to some state and thus, the transition probabilities given in row i must sum to 1 for all i. In other words, the TPMs are stochastic.

In an absorbing Markov chain, one can classify each of the states of the chain as either transient or absorbing. A state is termed *transient* if it is possible to leave that state and is called absorbing otherwise. In the model of the large-scale grid system, the states *Initial*, *Waiting*, *Discovering*, *Negotiating*, and *Monitoring* are all transient, while the states *Tasks Completed* and *Tasks Failed* are absorbing, so called because once the system arrives in that state, there is zero probability of leaving it. Notice that the transition probability matrix associated with an absorbing Markov chain can always be taken to have the following form

$$\mathcal{P} = \begin{bmatrix} Q & R \\ 0 & I \end{bmatrix}$$

where Q gives the probabilities of transitioning between transient states, R gives the probabilities of transitioning from a transient state to an absorbing state, and the zero submatrix represents the fact that it is impossible to transition from an absorbing state to a transient state [57]. The submatrix I is the $l \times l$ identity matrix where l is the number of absorbing states, which represents the fact that the only transitions from the l absorbing states are self-transitions. Observe that the transition probability matrices for the grid system Markov model are already in this form; as we can see from Fig. 3.2(b), the first 5×5 submatrix corresponds to transitions among the transient states, the next 5×2 submatrix corresponds to transitions from transient to absorbing states, the lower left 2×5 submatrix is zeros, and the lower right 2×2 submatrix is the identity matrix. This is shown in Fig. 7.1.

	Initial	Wait	Disc	Ngt	Mon	Compl	Fail
Initial	0.9997	0	0.0003	0	0	0	0
Wait	0	0.6292	0.0252	0.3441	0	0	0.0015
Disc	0	0.0766	0.6133	0.3101	0	0	0
Ngt	0	0.0378	0.0015	0.0637	0.8710	0	0.0259
Mon	0	0	0	0.0004	0.9883	0.0113	0
Compl	0	0	0	0	0	1.0	0
Fail	0	0	0	0	0	0	1.0

Figure 7.1. Canonical form of summary TPM in Fig. 3.2 (b). In this form, Q gives the probabilities of transitioning between transient states, R gives the probabilities of transitioning from a transient state to an absorbing state, the zero submatrix indicates that it is impossible to transition from an absorbing state to a transient state, and I is the $l \times l$ identity matrix where l is the number of absorbing states, where the only transitions are self-transitions.

Now, using this special form of the TPM, it is possible to express powers of the matrix $\mathcal{P}$ in terms of the submatrices. It follows easily by induction that $\mathcal{P}^S$ is given by

$$\mathcal{P}^S = \begin{bmatrix} Q^S & (I + \sum_{i=1}^{S-1} Q^i)R \\ \mathbf{0} & I \end{bmatrix}.$$

Furthermore, it is possible to express a product of powers of different TPMs again in terms of these submatrices. Let $\mathcal{P}_i$ represent the TPM for the ith time period with submatrices Q_i and R_i. Then to calculate the product of the matrices corresponding to the first $m=kS+t$ time steps, where S is the number of time steps per time period and t is the number of time steps elapsed in the current $(k+1)$th period, one may use the following equation

$$\mathcal{P}_1^S \cdots \mathcal{P}_k^S \cdot \mathcal{P}_{k+1}^t = \begin{bmatrix} Q_1^S \cdots Q_k^S \cdot Q_{k+1}^t & A_m \\ \mathbf{0} & I \end{bmatrix}.$$

The submatrix A_m is given by

$$A_m = \left(I + \sum_{j=1}^{S-1} Q_1^j\right) R_1 + \sum_{l=2}^{k} Q_1^S \cdots Q_{l-1}^S \left(I + \sum_{j=1}^{S-1} Q_l^j\right) R_l + Q_1^S \cdots Q_k^S \left(I + \sum_{j=1}^{t-1} Q_{k+1}^j\right) R_{k+1}. \quad (7.2)$$

Here l is the index for the *lth* time period. Plugging this product of matrices into Eq. (7.2) gives a closed form equation for the value of the state vector $\mathbf{v}_m$ with $m = kS + t$:

$$\mathbf{v}_m = \mathbf{v}_0 \mathcal{P}_1^S \cdots \mathcal{P}_k^S \cdot \mathcal{P}_{k+1}^t.$$

Finally, since $\mathbf{v}_0$ consists of simply a 1 in the first entry and 0s elsewhere, it follows that the proportion of tasks completed at time step m is given by the (1, 6) entry of $\mathcal{P}_1^S \cdots \mathcal{P}_k^S \cdot \mathcal{P}_{k+1}^t$, or the (1, 1) entry of the submatrix A_m. Thus, it is vital to examine alternative methods of expressing the submatrix A_m.

First, we examine alternative formulations for the matrices Q_i. For the Markov models considered here, we verified that the eigenvalues of the Q_i matrices are distinct. Thus, by the Spectral Theorem, it is possible to express each Q_i in terms of its projections onto each of its eigenspaces [58]. To be more precise, consider the equation

$$Q_i \mathbf{y} = \lambda \mathbf{y}. \tag{7.3}$$

Suppose the Q_i are 5 × 5 matrices with distinct eigenvalues. Thus, there are five distinct values of λ; for the purposes of this report, the eigenvalues will always be ordered according to their absolute value (or modulus) of λ that satisfy Eq. (7.3), i.e., $|\lambda^{(1)}| \geq |\lambda^{(2)}| \geq \ldots . |\lambda^{(5)}|$. Then, associated with each eigenvalue, there is a distinct one-dimensional family of vectors **y** that satisfy the equation, i.e. for a given **y** that satisfies the equation for a given λ, every scalar multiple of **y** will also satisfy the equation for that λ. A column vector **y** that satisfies Eq. (7.3) for a given value of λ is known as a *right eigenvector* corresponding to the eigenvalue λ. For each of these eigenvalues, there also exists a one-dimensional family of row vectors **x** that satisfy the following equation

$$\mathbf{x} Q_i = \lambda \mathbf{x}. \tag{7.4}$$

The vector **x** is a *left eigenvector* corresponding to the eigenvalue λ. Then by the Spectral Theorem [57], since the eigenvalues of Q_i are distinct for all *i*, it is possible to find a set of left eigenvectors, $\{\mathbf{x}^{(j)}\}$, and a set of right eigenvectors, $\{\mathbf{y}^{(k)}\}$, such that

$$\mathbf{x}^{(j)} \cdot \mathbf{y}^{(k)} = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{if } j \neq k \end{cases} .$$

Then for each *j*, the matrix $P^{(j)} = \mathbf{y}^{(j)}\mathbf{x}^{(j)}$ is known as the *j*th *eigenprojection* of Q. Notice that due to the biorthogonality of the eigenvectors, the product of $P^{(j)}$ and $P^{(k)}$ is the zero matrix for $j \neq k$. Another consequence of the Spectral Theorem is that it is then possible to write each matrix Q_i in terms of these eigenprojections:

$$Q_i = \sum_{j=1}^{5} \lambda_i^{(j)} P_i^{(j)}.$$

Furthermore, as a result of the biorthogonality of the eigenvectors, there is a simple expression for any power of Q_i, in particular,

$$Q_i^S = \sum_{j=1}^{5} (\lambda_i^{(j)})^S P_i^{(j)}. \tag{7.5}$$

Since there are 85 time steps per time period in the large-scale grid system, each TPM will be raised to the 85th power by the end of its time period. Observe also that for small values of $\lambda_i^{(j)}$, this value raised to the 85th power will become negligible; specifically any $|\lambda^{(j)}| < 0.88$ will contribute a negligible amount to the calculation of Q_i^S for S = 85. Thus it is sufficient to express Q_i^S using only the leading terms of the sum. The Q_i in the Markov model for the 8 h simulation have three eigenvalues with an absolute value greater than 0.88, so in this case, Q_i^S is approximated as the first three terms of the sum in Eq. (7.5). Meanwhile, the Q_i in the Markov model for the 640 h simulation with load level 75% has only two eigenvalues greater than 0.88 (after the first 6 time periods), and thus the Q_i^S may be approximated with only the first two terms of the sum.

Next, we turn to alternative means of representing the submatrix R_i for each of the TPMs $\mathcal{P}_i$. In general, let $\mathcal{P}$ be any transition probability matrix of an absorbing Markov chain. Consider the long-term proportion of tasks in a given absorbing state s_j assuming that all the tasks began in a transient state s_i. The matrix $B = \{b_{ij}\}$ with b_{ij} equal to the probability of being absorbed into state s_j given a beginning in transient state s_i precisely

describes these proportions. The formula for this matrix B is then given in terms of the N fundamental matrix of $\mathcal{P}$ and the submatrix R:

$$B = NR,$$

where $N = (I - Q)^{-1}$ [10]. Additionally, it is well-known that the long-term proportion of tasks in a particular absorbing state, given that all the tasks began in state i, is equal to the ith entry of one of the leading two eigenvectors of the matrix $\mathcal{P}$ that correspond to the eigenvalue 1, where two eigenvectors are being considered because there are two absorbing states [10]. In particular, let V be a matrix consisting of the first two eigenvectors, but with the rows corresponding to the absorbing states removed. It is clear that the entries of V give exactly the long-term proportion of tasks in a particular absorbing state for each of the different possible transient starting states, and thus, $V = B$. Finally, one may solve for R, to find that

$$R = (I - Q)V = \left(I - \sum_{\mu=1}^{5} \lambda^{(\mu)} P^{(\mu)} \right) V.$$

Now it is possible to plug the approximation for Q_i^S in terms of its largest eigenvalues and the formula for R_i into Eq. (7.2). For the 640 h simulation these are the two largest eigenvalues, and for the 8 h simulation they are the three largest eigenvalues. We will restrict our discussion to the 640 h case because the 8 h case is very similar. Further simplification is obtained when we make use of the fact that within a given time step, the product of distinct eigenprojections is zero and each eigenprojection is idempotent, i.e., each eigenprojection squared is equal to the original eigenprojection. We can then rewrite $\sum_{j=1}^{S-1} Q_i^j R_i$ as:

$$\begin{aligned}
\sum_{j=1}^{S-1} Q_i^j R_i &= \left(\sum_{j=1}^{S-1} (\lambda_i^{(1)})^j P_i^{(1)} + (\lambda_i^{(2)})^j P_i^{(2)} \right) \left(I - \sum_{k=1}^{5} \lambda_i^{(k)} P_i^{(k)} \right) V_i \\
&= \sum_{j=1}^{S-1} (\lambda_i^{(1)})^j P_i^{(1)} (1 - \lambda_i^{(1)}) V_i + \sum_{j=1}^{S-1} (\lambda_i^{(2)})^j P_i^{(2)} (1 - \lambda_i^{(2)}) V_i \\
&= \left(\left(\lambda_i^{(1)} - (\lambda_i^{(1)})^S \right) P_i^{(1)} + \left(\lambda_i^{(2)} - (\lambda_i^{(2)})^S \right) P_i^{(2)} \right) V_i
\end{aligned}$$

where the final equation comes from the fact that the sums in the second line are telescoping. Now employing this reduction, the following formula for A_m is obtained:

$$\begin{aligned}
A_m \approx\ & \left(I - (\lambda_1^{(1)})^S P_1^{(1)} - (\lambda_1^{(2)})^S P_1^{(2)} \right) V_1 \\
& + \sum_{l=2}^{k} \prod_{i=1}^{l-1} \left((\lambda_i^{(1)})^S P_i^{(1)} + (\lambda_i^{(2)})^S P_i^{(2)} \right) \left(I - (\lambda_l^{(1)})^S P_l^{(1)} - (\lambda_l^{(2)})^S P_l^{(2)} \right) V_l \\
& + \prod_{i=1}^{k} \left((\lambda_i^{(1)})^S P_i^{(1)} + (\lambda_i^{(2)})^S P_i^{(2)} \right) \left(I - (\lambda_{k+1}^{(1)})^S P_1^{(1)} - (\lambda_{k+1}^{(2)})^S P_1^{(2)} \right) V_{k+1}
\end{aligned} \tag{7.6}$$

Although this formula appears particularly cumbersome, there is much insight to be gained upon closer examination. This formulation for A_m illustrates that the primary quantities that affect A_m, and thus affect the proportion of tasks completed, are $\{\lambda_i^{(1)}\}, \{\lambda_i^{(2)}\}, \{P_i^{(1)}\}, \{P_i^{(2)}\}$ and $\{V_i\}$, where i ranges from 1 to the number of time periods; in other words, the leading eigenvalues and associated eigenvectors and projections of the

transient part of the TPM for each period. These quantities largely determine the job completion rate as a function of time. We can demonstrate this by observing that an approximation of the proportion of completed tasks using Eq. (7.6) compares very favorably with the results of a large-scale grid simulation or Markov simulation. Indeed Figs. 7.2–7.5 illustrate the results of approximating the probability of completion using the (1, 1) entry of A_m from Eq. (7.6) plotted against the results of the large-scale simulation and the Markov model for different lengths of observation time and different system load levels. The effect of perturbations of the performance of the large-scale grid system can be modeled with good qualitative agreement by appropriate perturbations of the Markov chain as demonstrated in [7] and in Figs. 7.6–7.9. The discussion in this section shows that we should be able to use the analytical formula given in Eq. (7.6) instead of the Markov simulation, as long as the eigenvalues of the perturbed system are distinct and well separated from the boundary of the unit circle, which is the case in these models. Evaluating the effect of the degree of perturbation on the proportion of tasks completed is thus substantially faster and easier because this involves the calculation of eigenvalues and eigenvectors. Moreover, as the discussion of the next subsections will show, there is a correlation between changes in the eigenvalues and decrease in performance. Thus there is evidence that they can play a useful role in the development of a methodology for predicting deleterious perturbations.

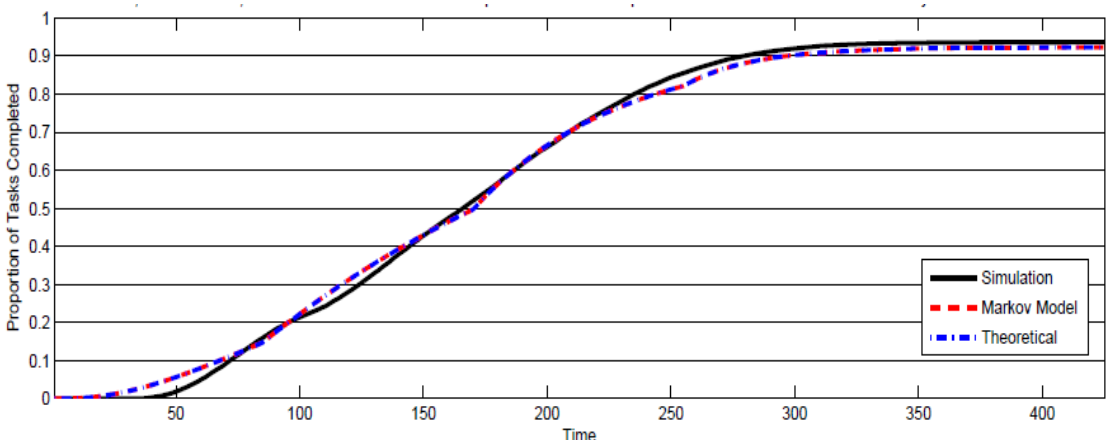

Figure 7.2: A comparison of the evolution of the proportion of tasks completed over time for the large-scale simulation, the Markov model, and the theoretical approximation for a simulated 8 h day with two hours of overtime.

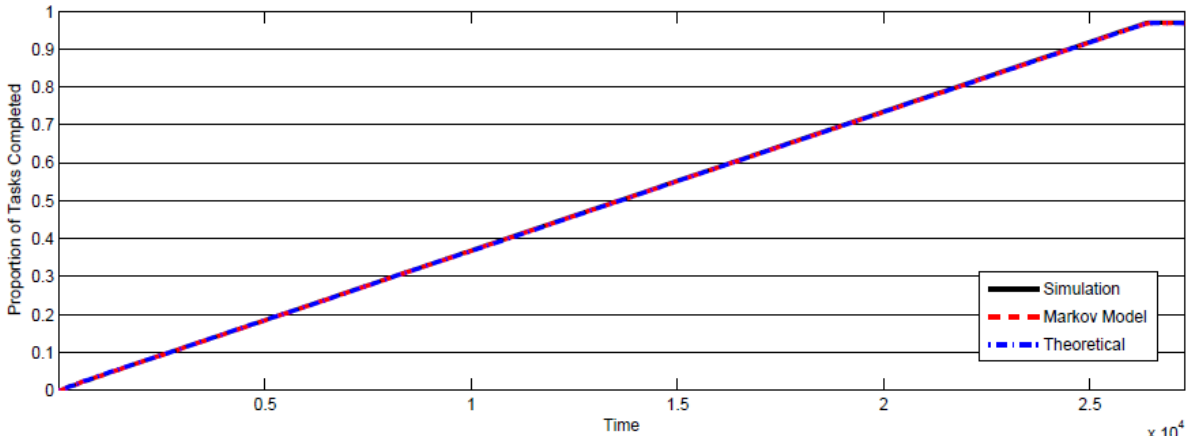

Figure 7.3: A comparison of the evolution of the proportion of tasks completed over time for the large-scale simulation, the Markov model, and the theoretical approximation for a simulated 640 h duration at a load level of 75%.

7.2 Quantifying Perturbation Effects

The perturbation method employed in the development of the Theoretical Method is similar to that described in [7]. A number of important differences exist, though, so the precise perturbation procedure for this report is described in Appendix C. Like the method in [7], this perturbation method is designed to determine the long-term effects of decreasing (or increasing) certain transition probabilities from a particular state while proportionally increasing (decreasing) other transition probabilities from that state. The development of the method described in Appendix C serves to demonstrate that more than one approach to systematic perturbation of a set to TPMs is possible.

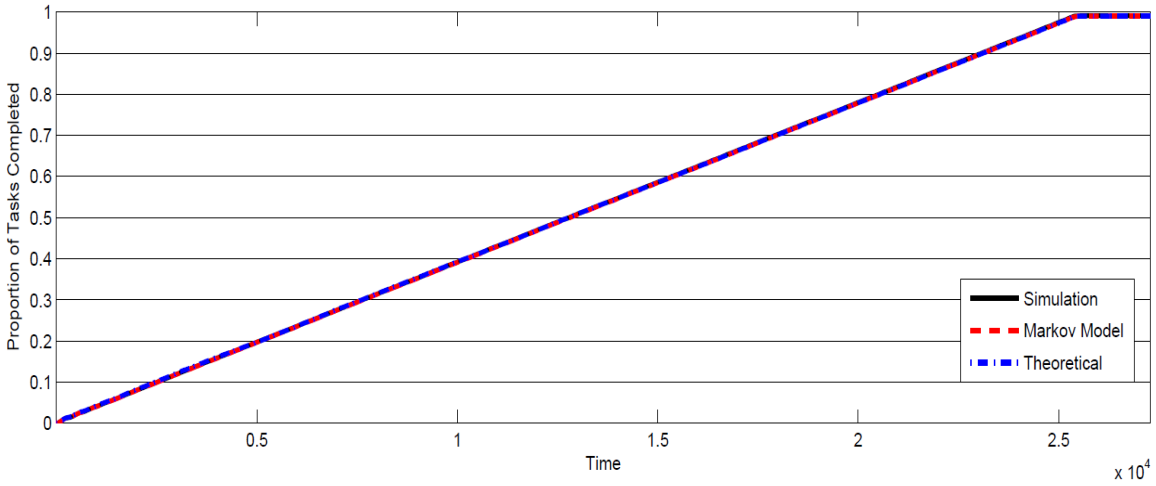

Figure 7.4: A comparison of the evolution of the proportion of tasks completed over time for the large-scale simulation, the Markov model, and the theoretical approximation for a simulated 640 h duration at a load level of 50%.

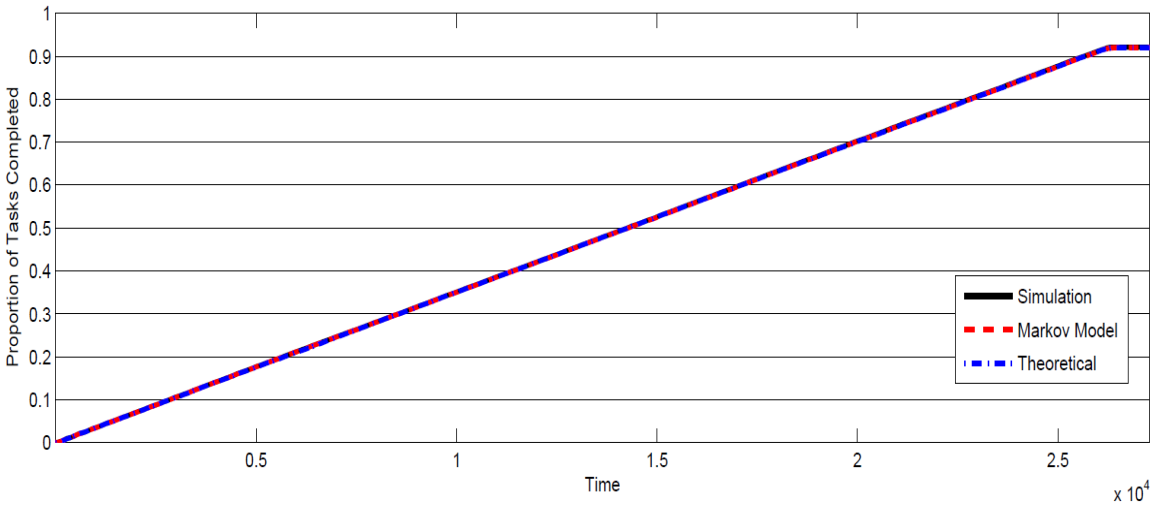

Figure 7.5: A comparison of the evolution of the proportion of tasks completed over time for the large-scale simulation, the Markov model, and the theoretical approximation for a simulated 640 h duration at a load level of 100%.

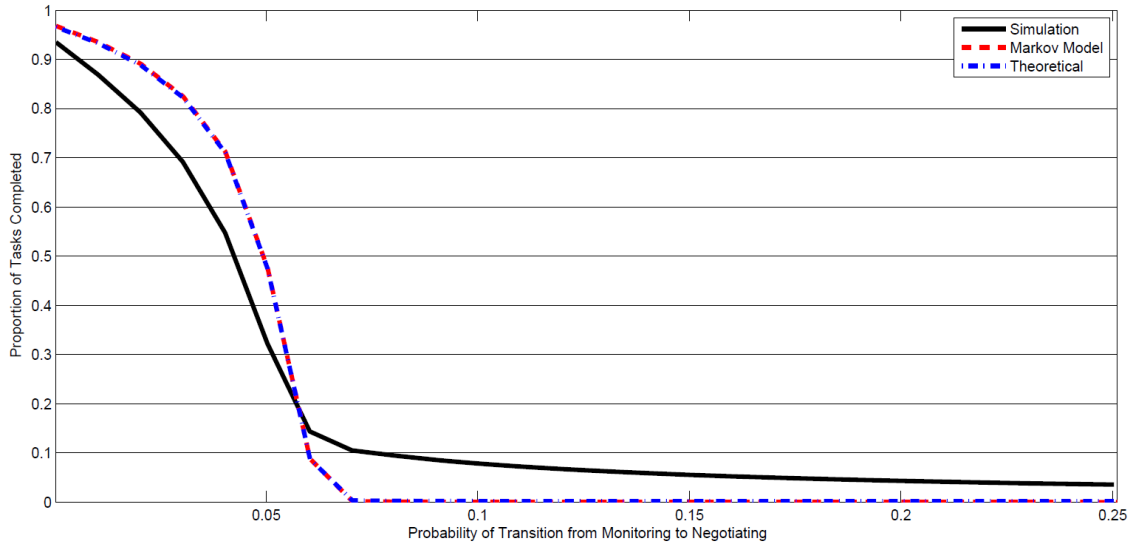

Figure 7.6: A comparison of the large-scale simulation, the Markov model, and the theoretical approximation for the proportion of tasks completed given an increase in the transition probability from *Monitoring* to *Negotiating* and a decrease of weight 0.2 in the transition from *Monitoring* to *Completion*. These results are for a simulated 640 h duration at a load level of 75%.

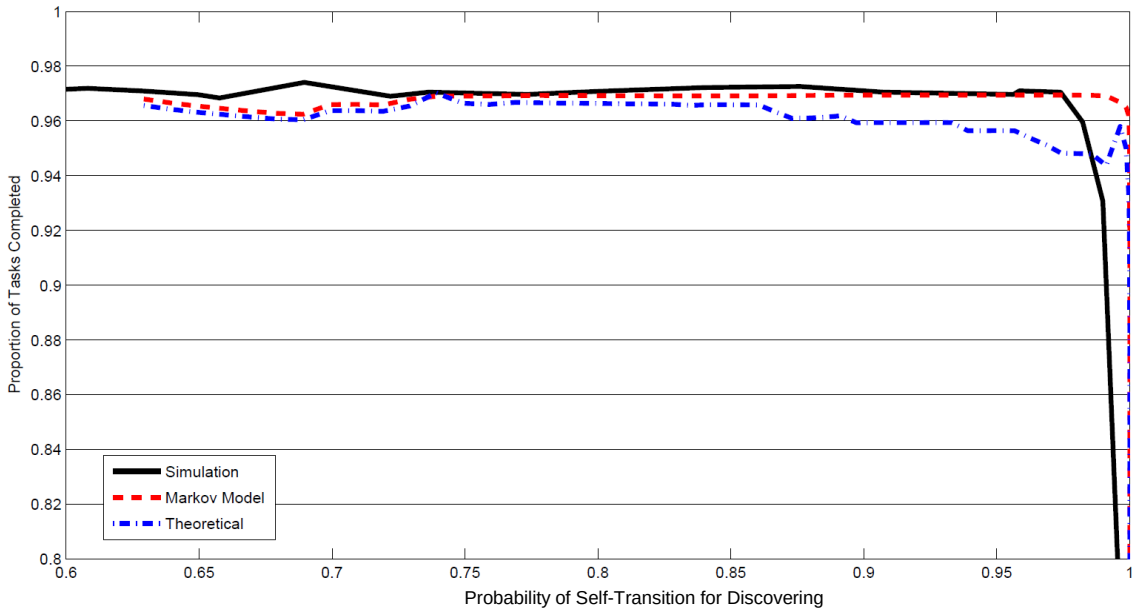

Figure 7.7. A comparison of the large-scale simulation, the Markov model, and the theoretical approximation for the proportion of tasks completed given an increase in the self-transition probability for *Discovering* and a decrease of weight 0.2 in the transition from *Discovering* to *Negotiating*. These results are for a simulated 640 h duration with a load level of 75%.

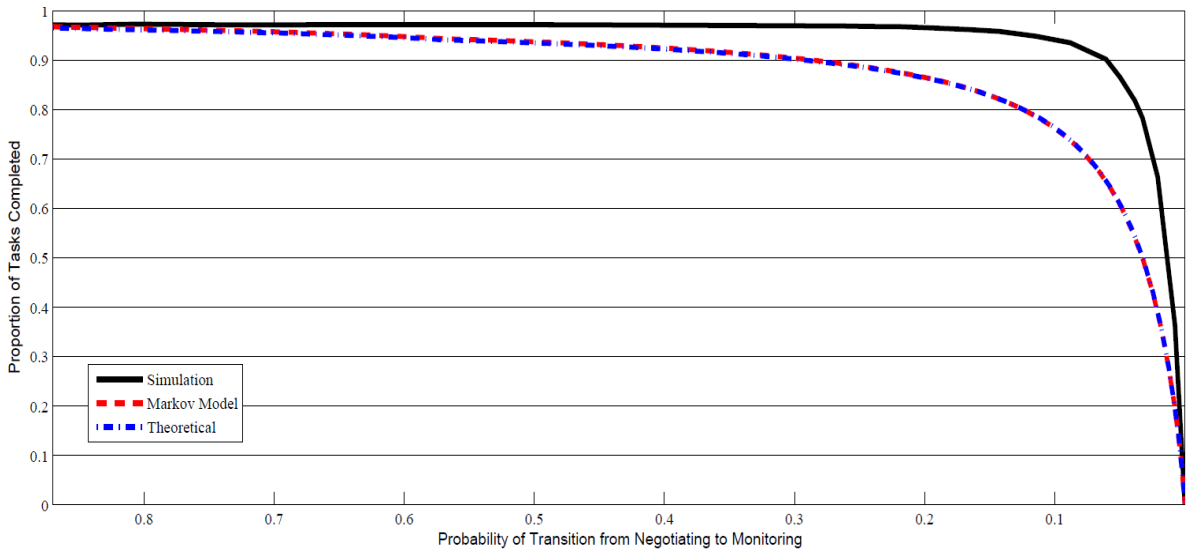

Figure 7.8: A comparison of the large-scale simulation, the Markov model, and the theoretical approximation for the proportion of tasks completed given a decrease in the transition probability from *Negotiating* to *Monitoring* and an increase of weight 1.0 in the self-transition for *Negotiating*. These results are for a simulated 640 h duration at a load level of 75%.

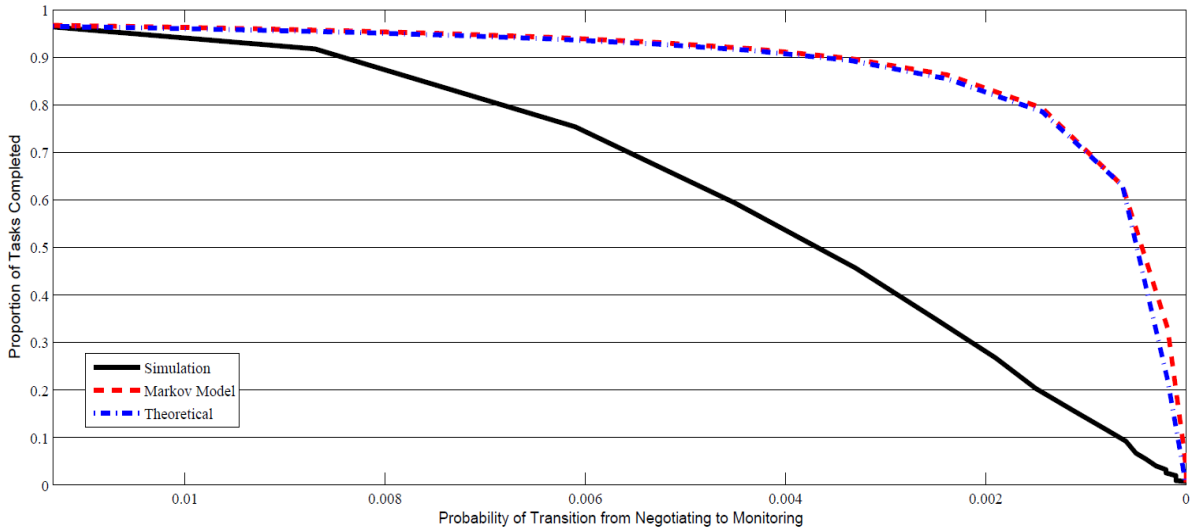

Figure 7.9: A comparison of the large-scale simulation, the Markov model, and the theoretical approximation for the proportion of tasks completed given a decrease in the transition probability from *Monitoring* to *Completion* and an increase of weight 1.0 in the transition from *Monitoring* to *Negotiating*. These results are for a simulated 640 h duration at a load level of 75%.

We will now discuss the use of measuring and predicting performance deterioration in the 640 h simulation. As shown in Sec. 7.1, the primary quantities that impact the proportion of tasks completed are $\{\lambda_i^{(1)}\}$ and $\{\lambda_i^{(2)}\}$ first and second leading eigenvalues of the submatrices $\{Q_i\}$, $\{P_i^{(1)}\}$ and $\{P_i^{(2)}\}$, the first and second leading eigenprojections of the $\{Q_i\}$, and finally $\{V_i\}$, the leading right eigenvector of $\{Q_i\}$. Here i is the index of the *ith* time period. By appropriately quantifying the changes in these quantities caused by a perturbation, we are able to demonstrate their usefulness in identifying those perturbations that lead to significant decrease in performance. This section examines some means for quantifying the changes $\{\lambda_i^{(1)}\}, \{\lambda_i^{(2)}\}, \{P_i^{(1)}\}, \{P_i^{(2)}\},$ and $\{V_i\}$, and examines the predictive power of these quantities.

To quantify the change in the leading two eigenvalues of the Q_i submatrices, the mean value of the percent change in the sum of the first two eigenvalues was calculated. To be more precise, the quantity examined was given by

$$\text{Diff}\,\lambda_i \doteq 100 * \frac{1}{N}\sum_{i=1}^{N}\frac{\left|(\lambda_i^{(1)'} + \lambda_i^{(2)'}) - (\lambda_i^{(1)} + \lambda_i^{(2)})\right|}{\left|\lambda_i^{(1)} + \lambda_i^{(2)}\right|} \tag{7.7}$$

where $\lambda_i^{(1)'}$ and $\lambda_i^{(2)'}$ are the leading eigenvalues of the perturbed submatrices Q_i' and N is the number of time periods. This method for quantifying the change in the leading eigenvalues was chosen because it captured the magnitude of the change over all the time periods. A similar formula was used to quantify the change in the leading two eigenprojections of the Q_i; for the eigenprojections, the mean value of the percent change in the norm of the sum of the first two eigenprojections was calculated. In other words, the quantity of interest was

$$\text{Diff}\,P_i \doteq 100 * \frac{1}{N}\sum_{i=1}^{N}\frac{\text{norm}\left((P_i^{(1)'} + P_i^{(2)'}) - (P_i^{(1)} + P_i^{(2)})\right)}{\text{norm}\left(P_i^{(1)} + P_i^{(2)}\right)} \tag{7.8}$$

where the 2-norm was used, i.e., the norm function calculated the square root of the sum of the squares of the entries of the matrix. Next, to measure the change in the leading right eigenvector of the $\mathcal{P}_i$, the mean value of the percent change in the norm of the eigenvector was used, i.e.,

$$\text{Diff}\,V_i \doteq 100 * \frac{1}{N}\sum_{i=1}^{N}\frac{\text{norm}\,(V_i' - V_i)}{\text{norm}\,(V_i)}. \tag{7.9}$$

Additional attempts were made to quantify the total change across the quantities $\{\lambda_i^{(1)}\}, \{\lambda_i^{(2)}\}, \{P_i^{(1)}\}, \{P_i^{(2)}\}$, and $\{V_i\}$. One formula considered was the product of the mean percent change in the norm of the eigenprojections and the mean percent change in the norm of the leading right eigenvector, in other words,

$$\text{Diff}\,P_i * \text{Diff}\,V_i \doteq 100 * \frac{1}{N}\sum_{i=1}^{N}\frac{\text{norm}\left((P_i^{(1)'} + P_i^{(2)'}) - (P_i^{(1)} + P_i^{(2)})\right)}{\text{norm}\left(P_i^{(1)} + P_i^{(2)}\right)} * 100 * \frac{1}{N}\sum_{i=1}^{N}\frac{\text{norm}\,(V_i' - V_i)}{\text{norm}\,(V_i)}. \tag{7.10}$$

Another quantity investigated is given by the formula below

$$\text{Prod Diff}\,\lambda_i * P_i * V_i \doteq 100 * \frac{1}{N}\sum_{i=1}^{N}\frac{\text{norm}\left((\lambda_i^{(1)'}P_i^{(1)'} + \lambda_i^{(2)'}P_i^{(2)'})V_i' - (\lambda_i^{(1)}P_i^{(1)} + \lambda_i^{(2)}P_i^{(2)})V_i\right)}{\text{norm}\left((\lambda_i^{(1)}P_i^{(1)} + \lambda_i^{(2)}P_i^{(2)})V_i\right)}. \tag{7.11}$$

Recall that the quantities $\lambda_i^{(1)}P_i^{(1)}V_i$ and $\lambda_i^{(2)}P_i^{(2)}V_i$ appear throughout Eq. (7.6), which approximates the proportion of tasks completed. Thus, the value from Eq. (7.11) should be a good measurement of changes in the proportion of tasks completed.

Table 7.1. Changes in the leading eigenvalues, eigenprojections, eigenvectors, and proportion of tasks completed caused by various perturbation combinations using the primary decrease perturbation method described in Appendix C. The perturbation combinations are defined by the choice of sink and primary increase columns with $v = 1$ and $w = 0.5$. The table shows the proportion of tasks completed by the Markov simulation program and indicates whether the perturbation combination corresponded to a single-transition s-t cut (see sec. 5). The system was simulated over 640 h at a load level of 75%.

Sink Column ($c^{\downarrow}$)	Primary increase column ($c^{\uparrow}$)	Diff λ_i Eq. (7)	Diff P_i Eq. (8)	Diff V_i Eq. (9)	Prod Diff P_i, V_i Eq. (10)	Prod Diff $\lambda_i * P_i * V_i$ Eq (11)	Percent change in prop. *Tasks Completed*	Single-transition s-t cut exists
(a) ***r*** = *Waiting*								
Waiting	*Discovering*	0.00	0.59	0.03	0.02	0.50	+0.06	No
Waiting	*Negotiating*	0.00	1.14	0.23	0.26	1.10	-0.06	No
Discovering	*Waiting*	0.00	0.13	0.07	0.01	0.14	-0.03	No
Discovering	*Negotiating*	0.00	0.07	0.03	0.00	0.08	-0.01	No
Negotiating	*Waiting*	0.59	97.68	40.50	3995.66	87.24	-7.94	No
Negotiating	*Discovering*	0.04	3.84	0.15	0.57	3.48	-0.13	No
(b) ***r*** = *Discovering*								
Waiting	*Discovering*	0.01	0.16	0.02	0.00	0.08	+0.04	No
Waiting	*Negotiating*	0.01	0.16	0.02	0.00	0.08	+0.04	No
Discovering	*Waiting*	0.02	0.92	0.04	0.04	0.68	-0.04	No
Discovering	*Negotiating*	0.02	0.93	0.04	0.03	0.70	-0.03	No
Negotiating	*Waiting*	0.03	2.18	0.18	0.38	2.06	-0.27	No
Negotiating	*Discovering*	0.03	2.26	0.40	0.90	2.20	-0.31	No
(c) ***r*** = *Negotiating*								
Waiting	*Discovering*	0.00	0.28	0.16	0.05	0.31	+0.09	No
Waiting	*Negotiating*	0.00	0.33	0.17	0.06	0.36	+0.09	No
Waiting	*Monitoring*	0.00	0.37	0.41	0.15	0.45	-0.01	No
Discovering	*Waiting*	0.00	0.02	0.00	0.00	0.02	0.00	No
Discovering	*Negotiating*	0.00	0.01	0.00	0.00	0.02	0.00	No
Discovering	*Monitoring*	0.00	0.13	0.05	0.01	0.15	+0.02	No
Negotiating	*Waiting*	0.00	0.31	0.07	0.02	0.29	+0.04	No
Negotiating	*Discovering*	0.00	0.10	0.00	0.00	0.05	-0.01	No
Negotiating	*Monitoring*	0.00	1.54	1.71	2.64	1.82	-1.66	No
Monitoring	*Waiting*	0.08	76.12	86.00	6546.52	87.04	-100.00	Yes
Monitoring	*Discovering*	0.09	76.21	86.00	6554.38	87.13	-100.00	Yes
Monitoring	*Negotiating*	0.04	76.43	86.00	6572.95	87.04	-100.00	Yes
(d) ***r*** = *Monitoring*								
Negotiating	*Monitoring*	0.01	0.48	0.13	0.06	0.52	+0.13	No
Negotiating	*Tasks Comp*	0.01	0.48	0.13	0.06	0.52	-0.13	No
Monitoring	*Negotiating*	14.19	127.40	3.01	383.42	159.05	-2.67*	Yes*
Monitoring	*Tasks Comp*	14.37	126.61	2.86	362.30	157.99	-2.58	No
Tasks Comp	*Negotiating*	0.56	3.94	97.51	383.81	97.51	-100.00	Yes
Tasks Comp	*Monitoring*	0.57	3.92	97.51	382.51	97.51	-100.00	Yes
(e) ***r*** = *Initial*								
Initial	*Discovering*	18.28	150.91	0.15	22.49	3.39	+2.14	No
Discovering	*Initial*	0.01	0.05	42.75	2.11	42.28	-100.00	Yes

*In this perturbation, assigning a weight of 1.0, rather than 0.5, to the primary increase column would ensure no tasks transition to *Monitoring*, which corresponds to a single-transition s-t cut along *Negotiating* → *Monitoring*. In Sec. 5.1 results are reported for this perturbation with a weight of 1.0.

The values from each of these equations were calculated for each of the possible perturbation combinations r, $c^{\downarrow}$, and $c^{\uparrow}$. The primary decrease perturbation method was used for these combinations with a value of v = 1, in other words, the (r, $c^{\downarrow}$) entry was perturbed to equal zero in each of the TPMs and the appropriate increase was then proportionally distributed among the other entries in row r. The values for Eqs. (7.7–7.11) were only calculated for the TPMs derived from this extreme of the perturbations. The rationale for using the extreme value of the perturbation was that if decreasing the value of a particular transition probability would affect the proportion of tasks completed, then cutting off that transition entirely would produce the largest effect on the proportion of tasks completed. Thus, to predict if decreasing a particular transition probability will have any effect on the long-term behavior of the grid system, it should suffice to predict if eliminating that transition will affect long-term behavior. Table 7.1 below summarizes the values obtained from Eqs. (7.7–7.11) for each of the perturbations along with the absolute value of the percent change in the proportion of tasks completed. For comparative purposes, the table also shows the results of the minimal s-t cut set analysis from Sec. 5.

Performing a multivariable linear regression on the decrease in proportion of tasks completed (shown in column 8) with the quantities in Eqs. (7.7–7.11) as predictor variables, is a useful way to conceptualize the results of Table 7.1. The regression coefficients can be used to determine which variables are most consequential in accounting for these performance drops. We calculated the coefficients, their 95% confidence intervals, residuals and some goodness of fit statistics. All computations were carried out using MATLAB 7.4.0 (2007a). Using the data contained in the rows of Table 7.1 we obtained the regression coefficients shown in Table 7.2, together with the endpoints of the related confidence intervals.

Table 7.2. Coefficients of regression.

		Diff λ_i Eq. (7)	Diff P_i Eq. (8)	Diff V_i Eq. (9)	Prod Diff P_i, V_i Eq. (10)	Prod Diff $\lambda_i * P_i * V_i$ Eq. (11)
Coefficients of multivariable linear regression		-6.6057	0.8297	-1.0580	-0.0102	-0.0287
	Confidence interval left Endpoint	-11.0377	0.2799	-1.2970	-0.0181	-0.1871
	Confidence interval right Endpoint	-2.1738	1.3796	-0.8190	-0.0022	0.1296

Clearly the coefficient corresponding to Diff λ_i has the largest magnitude, and therefore we conclude that the change in eigenvalues has the largest influence in predicting the decrease in system performance. Although a high accuracy fit of the data is of secondary importance here, we used residuals and the coefficient of determination, r^2, as evidence to test the strength of our conclusion. The residuals (see Fig. 7.10) do not show any obvious systematic variation that would undermine the linear regression hypothesis. The residual values are circled and the vertical lines through them indicate the confidence intervals. Fig. 7.10 shows there are 2 outliers (indicated in red) and the estimated variance is 116.55. The coefficient of determination for this calculation is r^2=0.9373, thus the fit is very good but more importantly it suggests that our conclusion is a reasonable one. We then performed a second computation using just the first three quantities (Eq. (7.7), (7.8) and (7.9)) as predictor variables. The corresponding coefficients were –1.41383 (for Eq. (7.7)), 0.1863 (for Eq. (7.8)), and −1.2039 (for Eq. (7.9)). Based on the magnitude of the coefficients, we see that the change in both eigenvalues and eigenvectors are influential. Here the number of outliers increased to 4, as shown in Fig. 7.11. However, the value of r^2 remained stable at 0.9205, while the estimated variance rose to 137.19.

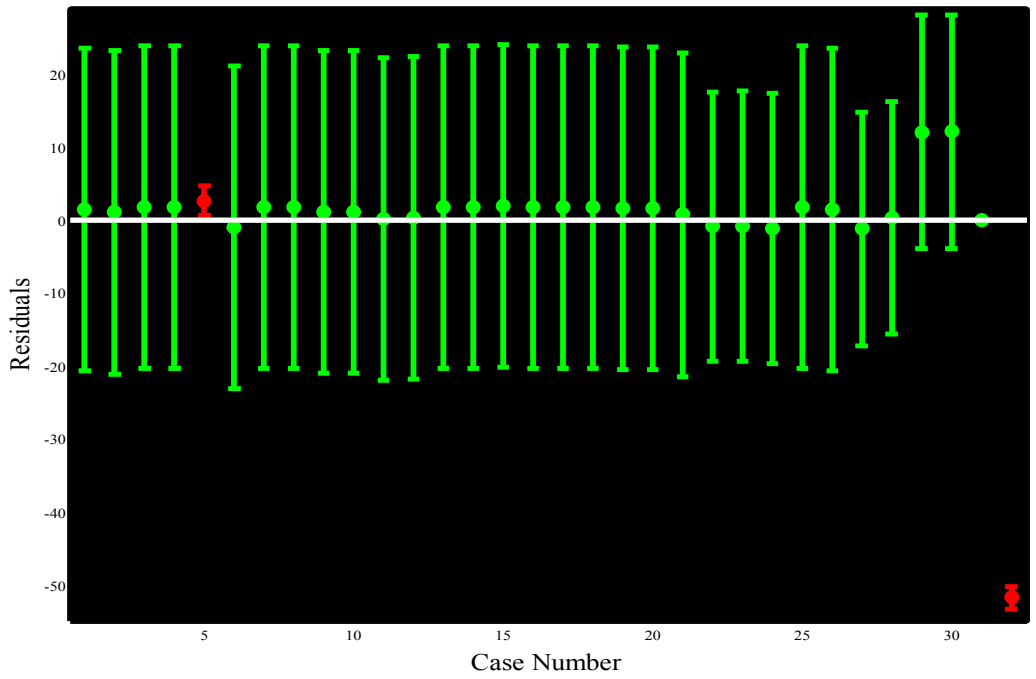

Figure 7.10. Plot of the residuals and confidence intervals for multilinear regression analysis on the change in proportion of tasks completed in scenarios shown in Table 7.1, which resulted from perturbation using the primary decrease method. This calculation was done for 5 predictor variables corresponding to Eqs. (7.7–7.11).

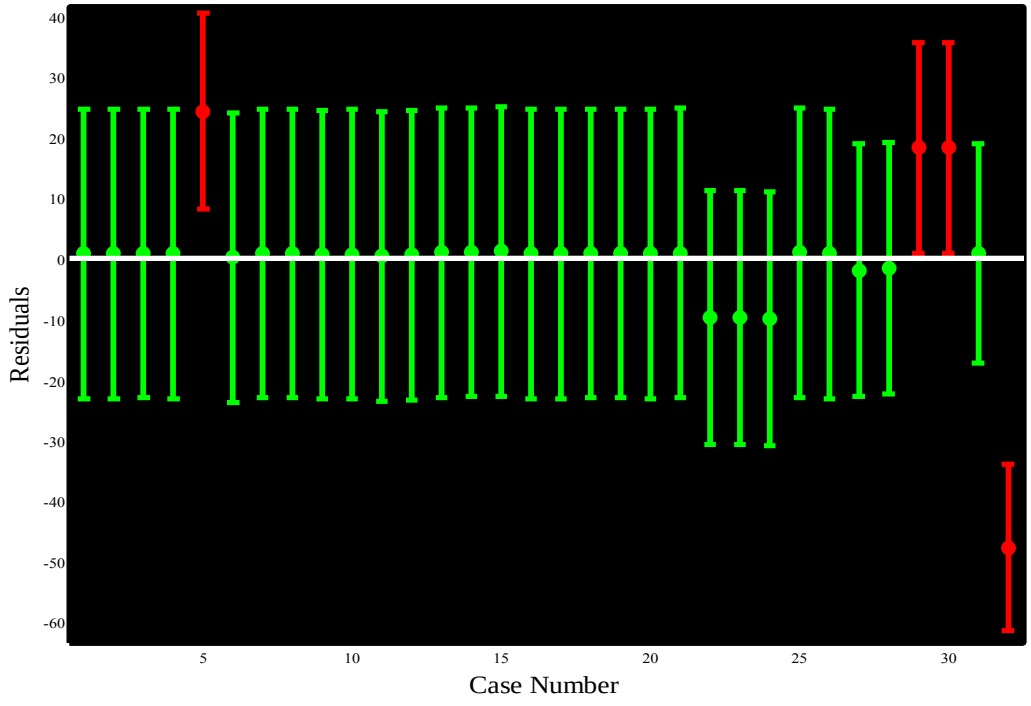

Figure 7.11. Plot of the residuals and confidence intervals for multilinear regression analysis on the change in proportion of tasks completed in scenarios shown in Table 7.1, which resulted from perturbation using the primary decrease method. This calculation was done for 3 predictor variables corresponding to Eqs. (7.7–7.9).

Table 7.3 Changes in the leading eigenvalues, eigenprojections, eigenvectors, and proportion of tasks completed caused by various perturbation combinations using the primary increase method described in Appendix C. The perturbation combinations are defined by the choice of sink and primary increase columns with $w = 0.5$. The table shows the proportion of tasks completed by the Markov simulation program and indicates whether the perturbation combination corresponded to a single-transition s-t cut (see Sec. 5). The system was simulated over 640 h at a load level of 75%.

Primary increase column ($c^{\uparrow}$)	Sink Column ($c^{\downarrow}$)	Diff λ_i Eq. (7)	Diff P_i Eq. (8)	Diff V_i Eq. (9)	Prod Diff P_i, V_i Eq. (10)	Prod Diff $\lambda_i * P_i * V_i$ Eq (11)	Percent change in prop. *Tasks Completed*	Corresponds to Single-transition s-t cut or trap state
(a) ***r*** = *Waiting*								
Waiting	*Discovering*	0.63	103.27	44.26	4571.00	92.57	-21.29	No
Waiting	*Negotiating*	0.62	103.27	44.37	4582.00	92.61	-21.29	No
Discovering	*Waiting*	0.02	9.06	8.43	76.00	9.10	-3.58	No
Discovering	*Negotiating*	0.01	13.87	13.59	189.00	13.97	-5.89	No
Negotiating	*Waiting*	0.00	0.87	0.25	0.00	0.82	+0.19	No
Negotiating	*Discovering*	0.00	0.87	0.25	0.00	0.82	+0.19	No
(b) ***r*** = *Discovering*								
Waiting	*Discovering*	0.02	0.79	0.18	0.00	0.30	-0.2369	No
Waiting	*Negotiating*	0.02	0.79	0.18	0.00	0.30	-0.2369	No
Discovering	*Waiting*	0.63	96.80	72.01	6970.00	99.17	-100.05	trap state
Discovering	*Negotiating*	0.63	96.80	71.76	6946.00	99.25	-100.05	trap state
Negotiating	*Waiting*	0.02	1.50	0.09	0.00	1.07	+0.05	No
Negotiating	*Discovering*	0.02	1.50	0.09	0.00	1.07	+0.05	No
(c) ***r*** = *Negotiating*								
Waiting	*Discovering*	0.61	103.11	111.189	11465.00	119.05	-100.00	s-t cut[1]
Waiting	*Negotiating*	0.61	103.11	111.03	11448.00	118.79	-100.00	s-t cut[1]
Waiting	*Monitoring*	0.61	103.11	111.24	11470.00	118.96	-100.00	s-t cut[1]
Discovering	*Waiting*	0.63	101.67	104.54	10628.00	113.31	-100.00	s-t cut[1]
Discovering	*Negotiating*	0.63	101.67	104.52	10626.00	113.21	-100.00	s-t cut[1]
Discovering	*Monitoring*	0.63	101.67	104.60	10635.00	113.27	-100.00	s-t cut[1]
Negotiating	*Waiting*	0.95	114.45	115.48	13218.00	123.12	-100.00	trap state
Negotiating	*Discovering*	0.95	114.45	115.00	13163.00	122.76	-100.00	trap state
Negotiating	*Monitoring*	0.95	114.45	114.50	13105.00	122.06	-100.00	trap state
Monitoring	*Waiting*	0.00	2.86	3.39	10.00	3.33	+3.13	No
Monitoring	*Discovering*	0.00	2.84	3.37	10.00	3.31	+3.16	No
Monitoring	*Negotiating*	0.009	2.71	3.20	9.00	3.15	+2.94	No
(d) ***r*** = *Monitoring*								
Negotiating	*Monitoring*	0.94	64.96	97.20	6314.00	97.20	-100.00	s-t cut[2]
Negotiating	*Tasks Comp*							s-t cut
Monitoring	*Negotiating*	0.58	2.98	96.35	287.00	96.4007	-100.09	trap state
Monitoring	*Tasks Comp*							trap state
Tasks Comp	*Negotiating*	18.46	113.44	0.13	15.00	110.65	+0.13	No
Tasks Comp	*Monitoring*	18.46	113.44	0.13	15.00	110.65	+0.13	No
(e) ***r*** = *Initial*								
Initial	*Discovering*	0.01	0.06	43.13	2.00	42.68	-100.00	trap state
Discovering	*Initial*	18.28	251.34	0.57	143.00	3.70	+2.14	No

[1]This perturbation is equivalent to a single-transition s-t cut. This is because as the probability of transition to the state corresponding to the primary increase column approaches 1, the probability of transition to all other columns goes to 0 – including the state transition from *Negotiating* to *Monitoring*, which is a single-transition s-t cut.

[2]As probability of transition from *Monitoring* to *Negotiating* approaches 1, all tasks are redirected out of the *Monitoring* state. This is the reverse of the *Negotiating* to *Monitoring* single-transition s-t cut and is effectively equivalent to it. See Table 7.1 and Sec. 5.1.

A similar analysis was carried out for perturbations generated by the primary increase method with w=0.5. With this method, the value of transition probability in the $(r, c^{\uparrow})$ entry was increased in increments of 0.01 and the summary value of $(r, c^{\uparrow})$ was recorded. These increases were performed until the last point at which the summing value of the $(r, c^{\uparrow})$ entry was strictly less than 1. For these perturbations the values of the quantities from Eqs. (7.7−7.11) are given in Table 7.3 along with the mean percent change in the proportion of tasks completed for each perturbation combination. As in Table 7.2, a multilinear regression calculation was carried out on change in proportion of tasks completed for each of the 30 perturbation scenarios indicated in the rows. The predictor variables are again the values from Eqs. (7.7−7.11). For the 5 variables, the corresponding regression coefficients were: –0.3038 (for Eq. (7.7)), 0.0284 (for Eq. (7.8)), –1.3038 (for Eq. (7.9)), 0.003 (for Eq. (7.10)), and 0.0189 (for Eq. (7.11)). The r^2 value or coefficient of determination was 0.9212 while the variance was 236.0. Thus, the linear fit was very good and roughly comparable to that obtained for the primary decrease method. Fig. 7.12 shows a plot of the residuals together with the confidence intervals. There are 4 outliers. Thus in the primary increase method, the change in the leading eigenvectors was most predictive of the drop in tasks completed.

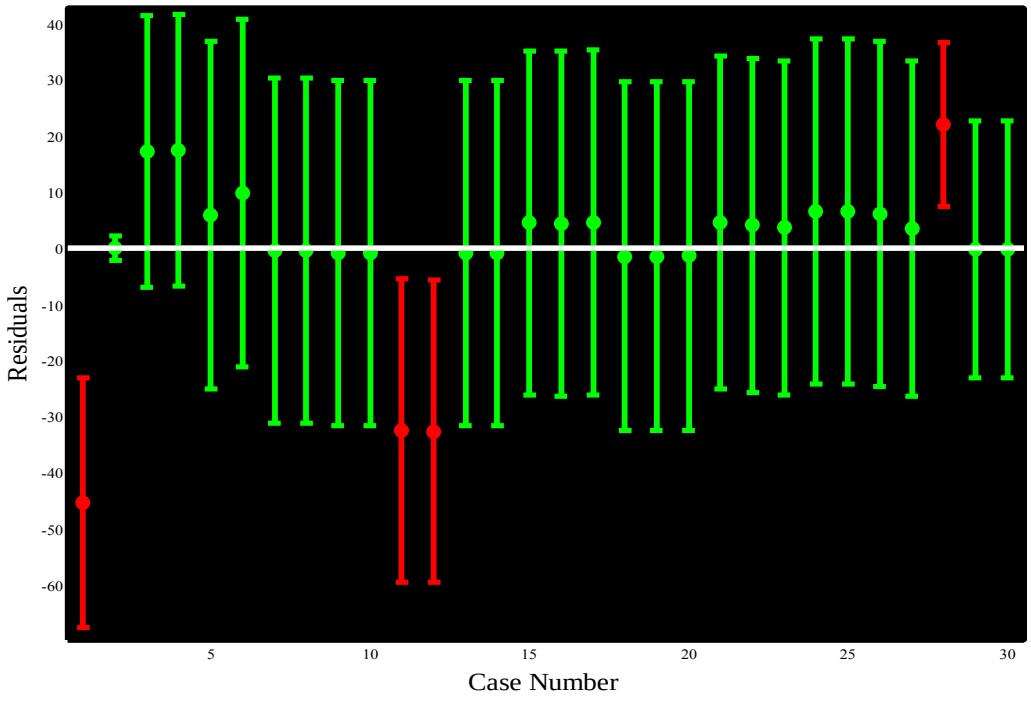

Figure 7.12. Plot of the residuals and confidence intervals for multilinear regression analysis on change in proportion of tasks completed in scenarios shown in Table 7.3, which resulted from perturbation using the primary increase method. This calculation was done for 5 predictor variables corresponding to Eqs. (7.7–7.11).

Using just the values from Eqs. (7.7–7.9) (i.e., eliminating the Eq. 7.10 and Eq. 7.11 that are derived from them), the regression coefficients are: –1.0482 (for Eq. (7.7)), 0.1374 (for Eq. (7.8)), –1.0712 (for Eq. (7.9)). In the residual plot, the number of outliers (shown in Fig. 7.13) was 3. Here the r^2 value was 0.9106 while the variance was 247.13.

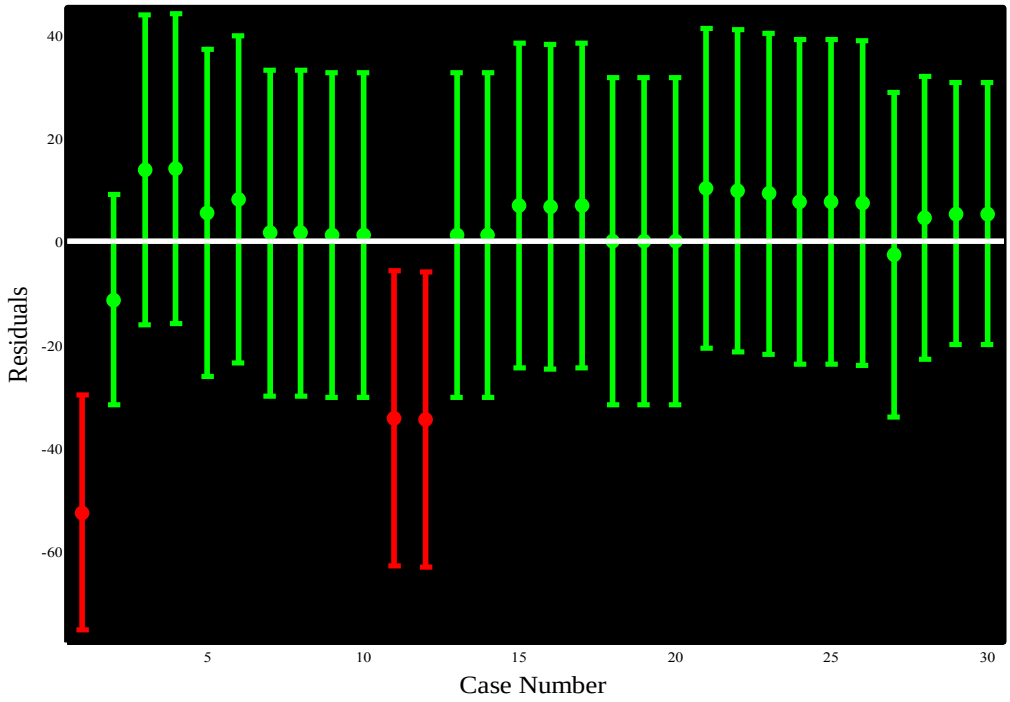

Figure 7.13. Plot of the residuals and confidence intervals for multilinear regression analysis on the change in proportion of tasks completed in scenarios shown in Table 7.3, which resulted from perturbation using the primary increase method. This calculation was done for 3 predictor variables corresponding to Eqs. (7.7–7.9).

In summary we evaluated how predictive the changes in the leading eigenvalues, leading eigenvectors and *eigenprojections of the* matrices Q_i (see Eq. (7.3) and Sec. 7.1) are of changes in the proportion of tasks completed by constructing a regression on changes in task completion proportion using the quantities in Eqs. (7.7–7.11) as independent variables. The regression fit was good as measured by the coefficient of determination, r^2, for both the perturbation decrease and the perturbation increase methods. It is reasonable therefore to consider the use of quantities defined by Eqs. (7.7–7.11), that is, changes in the leading eigenvalue and eigenvectors of Q_i as signals of large deterioration in system performance. In Sec. 5.1 we demonstrated the predictive power of minimal s-t cut sets in forecasting decreases in the number of task completions. Indeed, single s-t cut transitions, i.e., single transitions that cut off all paths joining the initial state to the completion state, are exactly associated with decreases of 100%. Inspection of Tables 7.1 and 7.3 shows that large values for the quantities in Eqs. (7.7–7.11) occur when a single-transition s-t cut or trap state is present in all cases except one. In the third row of Table 7.1(d), the particular perturbation chosen here prevents a drop in the proportion of tasks completed by distributing half of the increase weight to the transition from the state *Monitoring* to the state *Task Completed*. This has a compensating effect. Note that the predictive power is somewhat weaker than that provided by the minimal s-t cut sets in the sense that elevated values of Eqs. (7.7 –7.11) are necessary but not sufficient conditions for inferring performance loss. One can see examples in the tables of cases where values are elevated but there is no decrease in the proportion of tasks completed. Nevertheless, by comparing them with the results of cut set analysis we have provided additional evidence that the eigenvalue based quantities are effective signals. The existence of elevated values by themselves indicates that further analysis of the system is warranted. Alternatively, elevated values can be used as part a suite of measurements of system performance, which include cut set analysis and fast simulation of the Markov chain model based on the theoretical model. In this chapter we can see that the results of all three methodologies are largely consistent.

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 [7, 42] on the piecewise 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 report greatly reduce the time and effort required to identify failure scenarios discussed in these earlier papers.

Our first technique, minimal cut set analysis, is based on the graph topology induced by the Markov chains that constitute the pieces of the approximation. In our earlier work [7], 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 Sec. 3.1). The effect of any single perturbation of a critical transition could be determined by finding the proportion of tasks completed at the end of an observed time interval. In minimal cut set analysis, described in Secs. 4 and 5 of this report, key concepts from graph theory are used as a basis to define more efficient methods for finding critical transitions where perturbation can lead to large performance degradation. In this approach, minimal s-t cut sets are computed that disconnect all paths through the Markov chain graph from the initial state to a desired absorbing state. The edges in these cut sets reliably identify the critical state transitions which lead to drastic performance declines when perturbed. When applied to the Markov chain for the grid system and that of a second problem, the Abilene network, minimal s-t cut set analysis could be used to find the same critical transitions as those found by the perturbation algorithm. These critical transitions can then be perturbed directly to test for performance thresholds. Overall, the minimal cut set analysis approach resulted in a two-order of magnitude savings in computational cost over the perturbation algorithm.

To investigate the scalability of this new approach, one method that can be used to find minimal s-t cut sets—the node contraction algorithm—was applied to graph topologies induced by significantly larger stochastic matrices than the ones derived from the grid computing system or Abilene network. The results (Sec. 6) show that, with some exceptions, the node contraction algorithm was able to find a large proportion of minimal s-t cut sets that contained the most critical transitions in two orders of magnitude less time (20 min or less) than by using a well-known algorithm that was guaranteed to enumerate all minimal s-t cut sets [33]. Further, the node contraction algorithm was able to find minimal s-t cut sets that consisted of more than one critical state transition, which the earlier perturbation algorithm [7] was unable to find. The ability to efficiently analyze large, complex Markov chain problems in which combinations of critical state transitions can affect system performance is a significant improvement over the earlier perturbation algorithm. This ability is also likely to be an important factor in the potential use of this approach as a tool for predicting dynamic behavior in real-world distributed systems.

The second technique, the "theoretical method" is based on the fact that the total observation time is much larger than the duration of a single Markov chain step. We developed an accurate approximation of the probability of task completion (or more generally reaching a desired end state), in terms of the eigenvalues and eigenvectors associated with the transient part of the Markov chain. Like minimal cut set analysis, this analysis eliminates much of the computation required by earlier perturbation algorithm described in [7]. This technique is complementary to the minimal s-t cut set identification methods in the sense that the effect of any system perturbation (as modeled by a perturbation in the elements of the transition probability matrices) can be assessed by computing the approximation of the probability of task completion using the theoretical method. While the methods for identifying minimal s-t cut sets provide a qualitative approach to the identification of critical transitions in the system, the quantitative effect of perturbing these transitions at a specific level can be approximated by the theoretical method. Moreover the theoretical method can be used to test for threshold effects that would not be captured by examining the graph topology alone. In the latter case,

a minimum amount of Markov simulation is applied to critical transitions to obtain quantitative effects and test for thresholds. The analysis in Sec. 7 shows good comparisons with the results obtained by large-scale simulation or Markov chain iteration. The analysis also suggests that the theoretical method and minimal s-t cut set analysis methods are substantially in agreement in identifying single state transitions that may critically affect system performance.

Given the existence of rapid and efficient methods for computing the leading eigenvalues and corresponding eigenvectors of matrices, a natural question arises about how tight a connection there is between the degree of perturbation of the eigen-elements and degree of performance degradation. Indeed a very strong correlation would imply that eigen-elements are predictors of system failure and could be used to identify critical perturbations without using any other method. We found there is a positive but moderate correlation. However in some cases, large changes in the eigen-elements are not necessarily associated with a large change in the task completion rate. Thus each critical transition candidate identified by this method must be subjected to further investigation. Nevertheless, the theoretical method can be used to determine the effect of these perturbations, once they are identified without resorting to a calculation based on iteration of the transition probability matrices.

In summary, we have presented two methods—minimal cut set analysis and the theoretical method—that exhibit potential for efficient analysis of Markov chain representations of dynamic systems. We have shown that there is a large degree of consistency between the predictions these methods make, which may argue for their use as complementary tools in operational settings. While this work is in its beginning stages, the results to date show promise that both methods can be evolved into practical tools for analysis of complex, large-scale distributed systems and prediction of dynamic behavior.

9. References

[1] K. Mills, and C. Dabrowski, Investigating Global Behavior in Computing Grids, *Lecture Notes in Computer Science* **4124**, 120─136 (2006).

[2] D. Carr, How Google Works, *Baseline Magazine*, July 6, 2006, [Web Page], http://www.baselinemag.com, [accessed 7/15/08].

[3] D. Raffo, Grid, redundancy and home-cooked management help site survive, *Byte and Switch*, November 22, 2006.

[4] B. Chun, and E. Culler, User-centric performance analysis of market-based cluster batch schedulers, *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid*, Berlin Germany, May 2002, p. 30.

[5] C. Yeo, and R. Buyya, Service level agreement based allocation of cluster resources: handling penalty to enhance utility, *Proceedings of the 7th IEEE International Conference on Cluster Computing*, Boston, USA, September 2005, 27–30.

[6] K. Mills, and C. Dabrowski, Can Economics-based Resource Allocation Prove Effective in a Computation Marketplace? *Journal of Grid Computing* **6** (3), 291–311 (2008).

[7] C. Dabrowski, and F. Hunt, Using Markov Chain Analysis to Study Dynamic Behavior in Large-Scale Grid Systems, *Seventh Australasian Symposium on Grid Computing and e-Research (AUSGRID 2009)*, Wellington, New Zealand, January 2009.

[8] J. Wu, and F. Deng, Finite Horizon Optimal Control of Networked Control Systems with Markov Delays, *Proceedings of the Sixth World Congress on Intelligent Control and Automation*, Dalian, China, October 2006, 4513–4517.

[9] D. Feng, D. Wencai, and L. Zhi, New Smith Predictor and Nonlinear Control for Networked Control Systems, *Proceedings of the International MultiConference of Engineers and Computer Scientists* (Vol. II), Hong Kong, March 2009, 1148–1153.

[10] J. Kemeny, and J. Snell, *Finite Markov Chains*, New York, Springer (1976).

[11] M. Siegle, On Efficient Markovian Modelling, *Proceedings of the QMIPS Workshop on Stochastic Petri Nets*, Sophia Antipolis, France, November 1992, 213–225.

[12] P. Buchholz, Hierarchical Markovian Models: Symmetries and Reduction, *Performance Evaluation* **22** (1), 93–100 (1995).

[13] B. Aupperle, and J. Meyer, State space generation for degradable multiprocessor systems, *Twenty-First International Symposium on Fault-Tolerant Computing, 1991* (FTCS-21), Digest of Papers, June 1991, 308–315.

[14] W. Sanders, and J. Meyer, Reduced base model construction methods for stochastic activity networks, *IEEE Journal on Selected Areas in Communications*, Special Issue on Computer-Aided Modeling Analysis, and Design of Communication Networks **9 (**1), 25–36 (1991).

[15] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, Stochastic Well-Formed Colored Nets and Symmetric Modeling Applications, *IEEE Transactions on Computers* **42** (11), 1343–1360 (1993).

[16] W. Obal, and W. Sanders, Measure-adaptive state-space construction, *Performance Evaluation* **44** (1–4), 237–258 (2001).

[17] M. Jacobi, and O. Gornerup, A Dual Eigenvector Condition for Strong Lumpability of Markov Chains, submitted to Arxiv preprint arXiv:0710.1986.

[18] P. Schweitzer, Perturbation Theory and Finite Markov Chains, *Journal of Applied* Probability **5 (**2), 401–413 (1968).

[19] F. Delebecque, A Reduction Process for Perturbed Markov Chains, *SIAM Journal of Applied Mathematics* **43**, 325–250 (1983).

[20] C. Meyer, Stochastic Complementation, Uncoupling Markov Chains, and the Theory of Nearly Reducible Systems, *SIAM Review* **31** (2), 240–272 (1989).

[21] W. Stewart, and M. Dekker, *Numerical Solution of Markov Chains*, Princeton University Press, Princeton, New Jersey, 1994.

[22] Y. Ho, A Survey of the Perturbation Analysis of Discrete Event Dynamic Systems, *Annals of Operations Research* **3**, 393–402 (1985).

[23] R. Suri, Perturbation Analysis: The State of the Art and Research Issues Explained via the GI/G/l Queue, *Proceedings of the IEEE* **77** (1), 114–138 (1989).

[24] Y. Ho, and S. Li, Extensions of infinitesimal perturbation analysis, *IEEE Transactions on Automation Control* **AC-33** (5), 427–438 (1988).

[25] X. Cao, Basic Ideas for Event-Based Optimization of Markov Systems, *Discrete Event Dynamic Systems: Theory and Applications* **15**, 169–197 (2005).

[26] X. Cao, and J. Zhang, Event-Based Optimization of Markov Systems, *IEEE Transactions on Automatic Control* **53** (4), 1076–1082 (2008).

[27] M. Benzi, and M. Tuma, A parallel solver for large-scale Markov chains, *Applied Numerical Mathematics* **41**, 135–153 (2002).

[28] D. Fox, Block Cutpoint Decomposition for Markovian Queueing Systems, *Applied Stochastic Models and Data Analysis* **4** (2), 101–114 (1988).

[29] A. Gambin, P. Kryzanowski, and P. Pokarowski, Aggregation Algorithms for Perturbed Markov Chains With Applications To Networks Modeling, *SIAM Journal of Scientific Computation* **31** (1), 45–73 (2008).

[30] G. Hachtel, E. Macii, A. Pardo, and F. Somenzi, Markovian Analysis of Large Finite State Machines, *IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems* **15** (12), 1479–1493 (1996).

[31] E. Solan, and N. Vielle, Perturbed Markov Chains, Journal of Applied Probability **40**, 107–122 (2003).

[32] S. Tsukiyama, I. Shirakawa, H. Ozaki, and H. Ariyoshi, An Algorithm to Enumerate All Cut Sets of a Graph in Linear Time per Cutset, *Journal of the Association of Computing Machinery* **27** (4), 619–632 (1980).

[33] S. Provan, and M. Ball, Computing Network Reliability in Time Polynomial in the Number of Cuts, *Operations Research* **32** (3), 516–526 (1996).

[34] D. Karger, and C. Stein, A New Approach to the Minimum Cut Problem, *Journal of the ACM* **43**, 601–640 (1996).

[35] H. Lin, S. Kuo, and F. Yeh, Minimal cutset enumeration and network reliability evaluation by recursive merge and BDD, *Proceedings of the Eighth IEEE International Symposium on Computers and Communications*, Kemer-Antalya, Turkey, June 2003, 1341–1346.

[36] Z. Tang, and J. Dugan, Minimal cut set/sequence generation for dynamic fault trees, *Proceedings of the 2004 Annual Symposium on Reliability and Maintainability (RAMS)*, Los Angeles, CA, January, 2004, 207–213.

[37] L. Ford, and D. Fulkerson, Flows in Networks, Princeton University Press, Princeton, NJ, (1962).

[38] S. Even, *Graph Algorithms*, Computer Science Press, Rockville, MD (1979).

[39] A. Goldberg, and R. Tarjan, A New Approach to the Maximum-Flow Problem, *Journal of the Association for Computing Machinery* **35 (**4), 921–940 (1988).

[40] K. Mills, J. Filliben, D. Cho, E. Schwartz, and D. Genin, Study *of Proposed Internet Congestion-Control Mechanisms*, NIST Special Publication 500–282, National Institute of Standards and Technology: Gaithersburg, MD, May 2010.

[41] D. Rosenberg, E. Solan, and N. Vielle, Approximating A Sequence of Observations By A Simple Process, *The Annals of Statistics* **32** (6), 2742–2775 (2004).

[42] C. Dabrowski, and F. Hunt, Markov Chain Analysis for Large-Scale Grid Systems, *Interagency Report 7566*, National Institute of Standards and Technology: Gaithersburg, MD, April 2009.

[43] Measurement Science for Complex Systems, National Institute of Standards and Technology, http://www.nist.gov/itl/antd/emergent_behavior.cfm. November 2010.

[44] J. Hayakawa, S. Tsukiyama, and H. Ariyoshi, Generation of Minimal Separating Sets of Graphs, *IEICE Transaction Fundamentals* **E82-A** (5), 775–783 (1999).

[45] L. Fratta, and U. Montanari, A Vertex Elimination Algorithm for Enumerating all Simple Paths in a Graph, *Networks* **5**, 151–177 (1975).

[46] B. Banerjee, and B. Chandrasekaran, A framework for planning multiple paths in free space, *Proceedings of the 25th Army Science Conference*, Orlando, FL, November, 2006.

[47] D. Karger, A Randomized Fully Polynomial Time Approximation Scheme for the All-Terminal Network Reliability Problem, *SIAM Review* **43** (3), 499–522 (2001).

[48] L. Yan, H. Taha, and T. Landers, A Recursive Approach for Enumerating Minimal Cutsets in a Network, *IEEE Transactions on Reliability* **43** (3), 383–387 (1994).

[49] RW136: Markov Chain Transition Matrix, National Institute of Standards and Technology, [Web page], http://math.nist.gov/MatrixMarket/data/NEP/mvmrwk/rw136.html#set, [accessed 8/09].

[50] F. Hunt, A Monte Carlo Approach To The Approximation of Invariant Measures, *Random and Computational Dynamics* **2** (1), 111–122 (1994).

[51] R. Jensen, and E. Jessup, Statistical Properties of the Circle Map, *Journal of Statistical Physics* **43** (1–2), 369–389 (1986).

[52] A. Boyarksy, A matrix method for estimating the Liapunov exponent of one-dimensional systems, *Journal of Statistical Physics* **50** (1–2), 213–229 (1988).

[53] S. Grossmann, and S. Thomae, Invariant Distributions and Stationary Correlations of One-Dimensional Discrete Processes. *Zeitschrift für Naturforschung*, **32a**, 1353–1363 (1997).

[54] R. Tarjan, Depth-first Search and Liner Gaph Algortihms, *SIAM Journal of Computation* **1** (2), 146–160 (1972).

[55] N. Curet, J. DeVinney, and M. Gaston, An Efficient Network Flow Code for Finding all Minimum Cost *s-t C*utsets, *Computers & Operations Research* **29**, 205–219 (2002).

[56] A. Balcioglu, and K. Wood, Enumerating Near-Min s-t Cuts, in *Network Interdiction and Stochastic Integer Programming*, D. L. Woodruff, ed., Kluwer Academic Publishers, 21–49, 2003.

[57] S. Karlin, *A First Course in Stochastic Processes*, Academic Press, New York, 1966.

[58] J. Hunter, Mathematical Techniques of Applied Probability, Vol. I, Academic Press, New York (1983) pp. 144−145.

Appendix A. Node Contraction Algorithm

/* Procedure for generating minimal s-t cut set of a directed graph. Procedure accepts a TPM and identifiers for the 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 combined vertices. It ensures start and absorbing vertices are never in the same combined vertex. Procedure repeats until two vertices are left, and minimal s-t 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 so, unmark proposed vertex, and select again */
            contractionConflict = checkContractionConflict ( toBeContracted,  intoVertex, contractionRecord,
                                                                    startState, AbsorbingState);
            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);
     proposedState = linked [nthState];

    return proposedState;

} // procedure getContractedInto
```

/* Checks to see record of contraction indicates that StartState and Absorbing State have been contracted into the same vertex. If so, returns TRUE. Accepts identifier of vertex *toBeContracted*, vertex into which contraction occurs (*intoVertex*), and start and target states for s-t cut set. Current contraction record (*contractionRecord*) is an array for which index value identifies vertex that has been contracted and array element value identifies what which vertex the contracted vertex has been taken up into. */

```
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
```

```
/* Retrieves and returns cut set that exists between two remaining vertexs. */

procedure getCutSet (boolean notContractible [*], integer contractionRecord)
       returning cutSet
{
  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 from [51] using [50]. 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.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	
1	0.68	0.02	0	0.02	0.02	0.04	0.04	0	0.02	0	0.02	0	0.04	0.04	0.02	0	0.02	0.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04	0.02	0	0	0.02	0.02	0	0.02	0	0	0.02	0.02	0.02	0	0.02	0	0	0.02	0.02	0.02	0.02	0.04	0	0.06	0.02	0	0.6	2
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	5
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	6
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	7
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	8
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	9
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	10
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	11
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	12
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	13
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	14
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	15
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	16
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	17
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	18
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	19
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	20
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	21
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	22
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	23
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.02	0.04	0	0.04	0	0	0.02	0.02	0.02	0.84	24
25	0.36	0	0.04	0	0	0.04	0.04	0.04	0.02	0	0.04	0	0.04	0.02	0.02	0.06	0	0.02	0.06	0.02	0.02	0	0.02	0	0.02	0	0.02	0.04	0	0	0	0.02	0.02	0	0	0	0	0	0.02	0	0	0	0	0	0	0	0	0	0	25
26	0.64	0.02	0	0.02	0.04	0	0.02	0	0	0.04	0.02	0.04	0	0	0.04	0.02	0	0	0.02	0	0.06	0.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26
27	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27
28	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28
29	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	29
30	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30
31	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	31
32	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32
33	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33
34	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34
35	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35
36	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36
37	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	37
38	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38
39	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	39
40	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40
41	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	41
42	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42
43	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	43
44	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	44
45	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45
46	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46
47	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47
48	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
49	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49
50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	50
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	

Non-zero TPM elements are listed below.

(1,1)	0.680000	(24,50)	0.840000	(48,1)	1.000000
(1,2)	0.020000	(25,1)	0.360000	(49,1)	1.000000
(1,4)	0.020000	(25,3)	0.040000	(50,50)	1.000000
(1,5)	0.020000	(25,6)	0.040000		
(1,6)	0.040000	(25,7)	0.040000		
(1,7)	0.040000	(25,8)	0.040000		
(1,9)	0.020000	(25,9)	0.020000		
(1,11)	0.020000	(25,11)	0.040000		
(1,13)	0.040000	(25,13)	0.040000		
(1,14)	0.040000	(25,14)	0.020000		
(1,15)	0.020000	(25,15)	0.020000		
(1,17)	0.020000	(25,16)	0.060000		
(1,18)	0.020000	(25,18)	0.020000		
(2,24)	0.040000	(25,19)	0.060000		
(2,25)	0.020000	(25,20)	0.020000		
(2,28)	0.020000	(25,21)	0.020000		
(2,29)	0.020000	(25,23)	0.020000		
(2,31)	0.020000	(25,25)	0.020000		
(2,34)	0.020000	(25,27)	0.020000		
(2,35)	0.020000	(25,28)	0.040000		
(2,36)	0.020000	(25,33)	0.020000		
(2,38)	0.020000	(25,34)	0.020000		
(2,41)	0.020000	(25,40)	0.020000		
(2,42)	0.020000	(26,1)	0.640000		
(2,43)	0.020000	(26,2)	0.020000		
(2,44)	0.020000	(26,4)	0.020000		
(2,45)	0.040000	(26,5)	0.040000		
(2,47)	0.060000	(26,7)	0.020000		
(2,48)	0.020000	(26,10)	0.040000		
(2,50)	0.600000	(26,11)	0.020000		
(3,50)	1.000000	(26,12)	0.040000		
(4,50)	1.000000	(26,15)	0.040000		
(5,50)	1.000000	(26,16)	0.020000		
(6,50)	1.000000	(26,19)	0.020000		
(7,50)	1.000000	(26,21)	0.060000		
(8,50)	1.000000	(26,22)	0.020000		
(9,50)	1.000000	(27,1)	1.000000		
(10,50)	1.000000	(28,1)	1.000000		
(11,50)	1.000000	(29,1)	1.000000		
(12,50)	1.000000	(30,1)	1.000000		
(13,50)	1.000000	(31,1)	1.000000		
(14,50)	1.000000	(32,1)	1.000000		
(15,50)	1.000000	(33,1)	1.000000		
(16,50)	1.000000	(34,1)	1.000000		
(17,50)	1.000000	(35,1)	1.000000		
(18,50)	1.000000	(36,1)	1.000000		
(19,50)	1.000000	(37,1)	1.000000		
(20,50)	1.000000	(38,1)	1.000000		
(21,50)	1.000000	(39,1)	1.000000		
(22,50)	1.000000	(40,1)	1.000000		
(23,50)	1.000000	(41,1)	1.000000		
(24,41)	0.020000	(42,1)	1.000000		
(24,42)	0.040000	(43,1)	1.000000		
(24,44)	0.040000	(44,1)	1.000000		
(24,47)	0.020000	(45,1)	1.000000		
(24,48)	0.020000	(46,1)	1.000000		
(24,49)	0.020000	(47,1)	1.000000		

B.2 Matrix 2

A matrix of order 50 generated from [52] using [50]. 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.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.48	0.52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.56	0.44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.46	0.54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.46	0.54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.58	0.42	0	0	0	0	0	0	0	0	0	0	0	0	0	6
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.58	0.42	0	0	0	0	0	0	0	0	0	0	0	7
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.54	0.46	0	0	0	0	0	0	0	0	0	8
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.54	0.46	0	0	0	0	0	0	0	9
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0.5	0	0	0	0	0	10
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.64	0.36	0	0	0	11
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.54	0.46	0	12
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	13
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4	0.6	14
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.38	0.62	0	15
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.48	0.52	0	0	16
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.54	0.46	0	17
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.42	0.58	18
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0.5	19
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.58	0.42	0	0	0	0	0	20
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.42	0.58	0	0	0	0	0	0	21
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.48	0.52	0	0	0	0	22
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0.5	0	0	0	0	0	0	0	0	0	23
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0.5	0	0	0	0	0	0	0	0	0	0	24
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.46	0.54	0	0	0	0	0	0	0	0	0	0	0	25
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0.6	0.2	0	0	0	0	0	0	0	0	0	0	0	0	26
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.22	0.58	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.22	0.58	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.36	0.5	0.14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	29
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.22	0.56	0.22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.36	0.42	0.22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	31
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.26	0.5	0.24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.22	0.44	0.34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.26	0.44	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0.64	0.16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.34	0.46	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0.26	0.5	0.24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	37
38	0	0	0	0	0	0	0	0	0	0	0	0	0.66	0.34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38
39	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	39
40	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40
41	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	41
42	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42
43	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	43
44	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	44
45	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45
46	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46
47	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47
48	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48
49	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49
50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	50
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	

Non-zero TPM elements are listed below.

(1,26)	0.480000	(28,34)	0.200000
(1,27)	0.520000	(29,30)	0.360000
(2,28)	0.560000	(29,31)	0.500000
(2,29)	0.440000	(29,32)	0.140000
(3,30)	0.460000	(30,28)	0.220000
(3,31)	0.540000	(30,29)	0.560000
(4,32)	0.400000	(30,30)	0.220000
(4,33)	0.600000	(31,26)	0.360000
(5,34)	0.460000	(31,27)	0.420000
(5,35)	0.540000	(31,28)	0.220000
(6,36)	0.580000	(32,24)	0.260000
(6,37)	0.420000	(32,25)	0.500000
(7,38)	0.580000	(32,26)	0.240000
(7,39)	0.420000	(33,22)	0.220000
(8,40)	0.540000	(33,23)	0.440000
(8,41)	0.460000	(33,24)	0.340000
(9,42)	0.540000	(34,20)	0.260000
(9,43)	0.460000	(34,21)	0.440000
(10,44)	0.500000	(34,22)	0.300000
(10,45)	0.500000	(35,18)	0.200000
(11,46)	0.640000	(35,19)	0.640000
(11,47)	0.360000	(35,20)	0.160000
(12,48)	0.540000	(36,16)	0.340000
(12,49)	0.460000	(36,17)	0.460000
(13,50)	1.000000	(36,18)	0.200000
(14,49)	0.400000	(37,14)	0.260000
(14,50)	0.600000	(37,15)	0.500000
(15,48)	0.380000	(37,16)	0.240000
(15,49)	0.620000	(38,13)	0.660000
(16,47)	0.480000	(38,14)	0.340000
(16,48)	0.520000	(39,12)	1.000000
(17,46)	0.540000	(40,11)	1.000000
(17,47)	0.460000	(41,10)	1.000000
(18,45)	0.420000	(42,9)	1.000000
(18,46)	0.580000	(43,8)	1.000000
(19,44)	0.500000	(44,7)	1.000000
(19,45)	0.500000	(45,6)	1.000000
(20,43)	0.580000	(46,5)	1.000000
(20,44)	0.420000	(47,4)	1.000000
(21,42)	0.420000	(48,3)	1.000000
(21,43)	0.580000	(49,2)	1.000000
(22,41)	0.480000	(50,50)	1.000000
(22,42)	0.520000		
(23,40)	0.500000		
(23,41)	0.500000		
(24,39)	0.500000		
(24,40)	0.500000		
(25,38)	0.460000		
(25,39)	0.540000		
(26,36)	0.200000		
(26,37)	0.600000		
(26,38)	0.200000		
(27,34)	0.220000		
(27,35)	0.580000		
(27,36)	0.200000		
(28,32)	0.220000		
(28,33)	0.580000		

B.3 Matrix 3

This matrix of order 50 is a subset of the 136 × 136 [49] described in Sec. 6.2.2. 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.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	
1	0	0.50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2	0.07	0	0.47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
3	0	0.13	0	0.43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
4	0	0	0.20	0	0.40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
5	0	0	0	0.27	0	0.37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5
6	0	0	0	0	0.33	0	0.33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6
7	0	0	0	0	0	0.40	0	0.30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7
8	0	0	0	0	0	0	0.47	0	0.27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8
9	0	0	0	0	0	0	0	0.53	0	0.23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9
10	0	0	0	0	0	0	0	0	0.60	0	0.20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10
11	0	0	0	0	0	0	0	0	0	0.67	0	0.17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11
12	0	0	0	0	0	0	0	0	0	0	0.73	0	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12
13	0	0	0	0	0	0	0	0	0	0	0	0.80	0	0.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13
14	0	0	0	0	0	0	0	0	0	0	0	0	0.87	0	0.07	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.07	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.93	0	0	0.03	0	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16
17	0	0.07	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.47	0	0	0	0	0	0	0	0	0	0	0	0	0	0.47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17
18	0	0	0.07	0	0	0	0	0	0	0	0	0	0	0	0	0	0.07	0	0.43	0	0	0	0	0	0	0	0	0	0	0	0	0	0.43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18
19	0	0	0	0.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0.10	0	0.40	0	0	0	0	0	0	0	0	0	0	0	0	0	0.40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	19
20	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0	0.13	0	0.37	0	0	0	0	0	0	0	0	0	0	0	0	0	0.37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20
21	0	0	0	0	0	0.17	0	0	0	0	0	0	0	0	0	0	0	0	0	0.17	0	0.33	0	0	0	0	0	0	0	0	0	0	0	0	0	0.33	0	0	0	0	0	0	0	0	0	0	0	0	0	21
22	0	0	0	0	0	0	0.20	0	0	0	0	0	0	0	0	0	0	0	0	0	0.20	0	0.30	0	0	0	0	0	0	0	0	0	0	0	0	0	0.30	0	0	0	0	0	0	0	0	0	0	0	0	22
23	0	0	0	0	0	0	0	0.23	0	0	0	0	0	0	0	0	0	0	0	0	0	0.23	0	0.27	0	0	0	0	0	0	0	0	0	0	0	0	0	0.27	0	0	0	0	0	0	0	0	0	0	0	23
24	0	0	0	0	0	0	0	0	0.27	0	0	0	0	0	0	0	0	0	0	0	0	0	0.27	0	0.23	0	0	0	0	0	0	0	0	0	0	0	0	0	0.23	0	0	0	0	0	0	0	0	0	0	24
25	0	0	0	0	0	0	0	0	0	0.30	0	0	0	0	0	0	0	0	0	0	0	0	0	0.30	0	0.20	0	0	0	0	0	0	0	0	0	0	0	0	0	0.20	0	0	0	0	0	0	0	0	0	25
26	0	0	0	0	0	0	0	0	0	0	0.33	0	0	0	0	0	0	0	0	0	0	0	0	0	0.33	0	0.17	0	0	0	0	0	0	0	0	0	0	0	0	0	0.17	0	0	0	0	0	0	0	0	26
27	0	0	0	0	0	0	0	0	0	0	0	0.37	0	0	0	0	0	0	0	0	0	0	0	0	0	0.37	0	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0	0.13	0	0	0	0	0	0	0	27
28	0	0	0	0	0	0	0	0	0	0	0	0	0.40	0	0	0	0	0	0	0	0	0	0	0	0	0	0.40	0	0.10	0	0	0	0	0	0	0	0	0	0	0	0	0	0.10	0	0	0	0	0	0	28
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0.43	0	0	0	0	0	0	0	0	0	0	0	0	0	0.43	0	0.07	0	0	0	0	0	0	0	0	0	0	0	0	0	0.07	0	0	0	0	0	29
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.47	0	0	0	0	0	0	0	0	0	0	0	0	0	0.47	0	0	0.03	0	0	0	0	0	0	0	0	0	0	0	0	0.03	0	0	0	0	30
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	31
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.43	0	0	0	0	0	0	0	0	0	0	0	0	0.43	0	0	0	0	32
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.10	0	0	0	0	0	0	0	0	0	0	0	0	0.10	0	0.40	0	0	0	0	0	0	0	0	0	0	0	0.40	0	0	0	33
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0.13	0	0.37	0	0	0	0	0	0	0	0	0	0	0	0.37	0	0	34
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.17	0	0	0	0	0	0	0	0	0	0	0	0	0.17	0	0.33	0	0	0	0	0	0	0	0	0	0	0	0.33	0	35
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.20	0	0	0	0	0	0	0	0	0	0	0	0	0.20	0	0.30	0	0	0	0	0	0	0	0	0	0	0	0.30	36
37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.32	0	0	0	0	0	0	0	0	0	0	0	0	0.32	0	0.36	0	0	0	0	0	0	0	0	0	0	0	37
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.35	0	0	0	0	0	0	0	0	0	0	0	0.35	0	0.30	0	0	0	0	0	0	0	0	0	0	0	38
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.38	0	0	0	0	0	0	0	0	0	0	0	0.38	0	0.25	0	0	0	0	0	0	0	0	0	0	39
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.40	0	0	0	0	0	0	0	0	0	0	0	0.40	0	0.20	0	0	0	0	0	0	0	0	0	40
41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.42	0	0	0	0	0	0	0	0	0	0	0	0.42	0	0.15	0	0	0	0	0	0	0	0	41
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.44	0	0	0	0	0	0	0	0	0	0	0	0.44	0	0.11	0	0	0	0	0	0	0	42
43	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.46	0	0	0	0	0	0	0	0	0	0	0	0.46	0	0.07	0	0	0	0	0	0	43
44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.48	0	0	0	0	0	0	0	0	0	0	0	0.48	0	0	0.03	0	0	0	0	44
45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.50	0	0	0	0	0	0	0	0	0	0	0	0.50	0	0	0	0	0	0	45
46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.33	0	0	0	0	0	0	0	0	0	0	0	0	0	0.67	0	0	0	46
47	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.21	0	0	0	0	0	0	0	0	0	0	0	0.21	0	0.58	0	0	47
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0	0	0	0	0	0	0	0	0	0.25	0	0.50	0	48
49	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.29	0	0	0	0	0	0	0	0	0	0	0	0	0.29	0	0.43	49
50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00	50
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	

Non-zero TPM elements are listed below.

(1,2) 0.500000	(20,19) 0.133333	(35,34) 0.166667
(1,17) 0.500000	(20,21) 0.366667	(35,36) 0.333333
(2,1) 0.066667	(20,35) 0.366667	(35,49) 0.333333
(2,3) 0.466667	(21,6) 0.166667	(36,22) 0.200000
(2,18) 0.466667	(21,20) 0.166667	(36,35) 0.200000
(3,2) 0.133333	(21,22) 0.333333	(36,37) 0.300000
(3,4) 0.433333	(21,36) 0.333333	(36,50) 0.300000
(3,19) 0.433333	(22,7) 0.200000	(37,23) 0.318182
(4,3) 0.200000	(22,21) 0.200000	(37,36) 0.318182
(4,5) 0.400000	(22,23) 0.300000	(37,38) 0.363636
(4,20) 0.400000	(22,37) 0.300000	(38,24) 0.347826
(5,4) 0.266667	(23,8) 0.233333	(38,37) 0.347826
(5,6) 0.366667	(23,22) 0.233333	(38,39) 0.304348
(5,21) 0.366667	(23,24) 0.266667	(39,25) 0.375000
(6,5) 0.333333	(23,38) 0.266667	(39,38) 0.375000
(6,7) 0.333333	(24,9) 0.266667	(39,40) 0.250000
(6,22) 0.333333	(24,23) 0.266667	(40,26) 0.400000
(7,6) 0.400000	(24,25) 0.233333	(40,39) 0.400000
(7,8) 0.300000	(24,39) 0.233333	(40,41) 0.200000
(7,23) 0.300000	(25,10) 0.300000	(41,27) 0.423077
(8,7) 0.466667	(25,24) 0.300000	(41,40) 0.423077
(8,9) 0.266667	(25,26) 0.200000	(41,42) 0.153846
(8,24) 0.266667	(25,40) 0.200000	(42,28) 0.444444
(9,8) 0.533333	(26,11) 0.333333	(42,41) 0.444444
(9,10) 0.233333	(26,25) 0.333333	(42,43) 0.111111
(9,25) 0.233333	(26,27) 0.166667	(43,29) 0.464286
(10,9) 0.600000	(26,41) 0.166667	(43,42) 0.464286
(10,11) 0.200000	(27,12) 0.366667	(43,44) 0.071429
(10,26) 0.200000	(27,26) 0.366667	(44,30) 0.482759
(11,10) 0.666667	(27,28) 0.133333	(44,43) 0.482759
(11,12) 0.166667	(27,42) 0.133333	(44,46) 0.034483
(11,27) 0.166667	(28,13) 0.400000	(45,31) 0.500000
(12,11) 0.733333	(28,27) 0.400000	(45,44) 0.500000
(12,13) 0.133333	(28,29) 0.100000	(46,33) 0.333333
(12,28) 0.133333	(28,43) 0.100000	(46,47) 0.666667
(13,12) 0.800000	(29,14) 0.433333	(47,34) 0.210526
(13,14) 0.100000	(29,28) 0.433333	(47,46) 0.210526
(13,29) 0.100000	(29,30) 0.066667	(47,48) 0.578947
(14,13) 0.866667	(29,44) 0.066667	(48,35) 0.250000
(14,15) 0.066667	(30,15) 0.466667	(48,47) 0.250000
(14,30) 0.066667	(30,29) 0.466667	(48,49) 0.500000
(15,14) 0.933333	(30,32) 0.033333	(49,36) 0.285714
(15,17) 0.033333	(30,45) 0.033333	(49,48) 0.285714
(15,31) 0.033333	(31,16) 0.500000	(49,50) 0.428571
(16,15) 1.000000	(31,30) 0.500000	(50,50) 1.000000
(17,2) 0.066667	(32,18) 0.133333	
(17,18) 0.466667	(32,33) 0.433333	
(17,32) 0.466667	(32,46) 0.433333	
(18,3) 0.066667	(33,19) 0.100000	
(18,17) 0.066667	(33,32) 0.100000	
(18,19) 0.433333	(33,34) 0.400000	
(18,33) 0.433333	(33,47) 0.400000	
(19,4) 0.100000	(34,20) 0.133333	
(19,18) 0.100000	(34,33) 0.133333	
(19,20) 0.400000	(34,35) 0.366667	
(19,34) 0.400000	(34,48) 0.366667	
(20,5) 0.133333	(35,21) 0.166667	

B.4 Matrix 4

A matrix of order 50 generated from [53] using [50]. 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.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	
1	0.51	0.48	0.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	0.38	0.43	0.19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
3	0	0	0	0	0.13	0.30	0.38	0.19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
4	0	0	0	0	0	0	0	0.10	0.22	0.17	0.23	0.15	0.08	0.06	0.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.05	0.04	0.02	0	0	0	0	0	0.01	0.04	0.03	0.08	0.06	0.13	0.16	0.24	0.15	0	0	0	0	0	0	0	0	0	5
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0.27	0.38	0.26	0	0	0	0	0	0	6
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.21	0.60	0.19	0	0	0	0	7
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.55	0.45	0	0	0	8
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.58	0.42	0	0	9
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.69	0.31	0	10
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	11
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.66	0.34	12
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	13
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	14
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	15
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	16
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	17
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	18
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	19
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	20
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	21
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	22
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	23
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	24
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	25
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	26
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	27
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	28
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.63	0.37	0	29
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	30
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.65	0.35	0	0	0	0	31
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.48	0.53	0	0	32
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.48	0.52	0	0	0	33
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.16	0.52	0.33	0	0	0	0	34
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.08	0.32	0.37	0.23	0	0	0	0	0	0	35
36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.06	0.02	0.03	0.02	0.01	0	0.01	0	0	0.02	0.03	0.05	0.05	0.10	0.16	0.22	0.23	0	0	0	0	0	0	0	0	0	36
37	0	0	0	0	0	0	0	0.07	0.33	0.13	0.18	0.12	0.08	0.07	0.03	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	37
38	0	0	0	0	0.09	0.37	0.38	0.16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38
39	0	0	0.37	0.43	0.20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	39
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	40
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	

Non-zero TPM elements are listed below.

(1,1)	0.508333	(24,40)	1.000000
(1,2)	0.483333	(25,40)	1.000000
(1,3)	0.008333	(26,40)	1.000000
(2,3)	0.383333	(27,40)	1.000000
(2,4)	0.425000	(28,40)	1.000000
(2,5)	0.191667	(29,39)	0.633333
(3,5)	0.133333	(29,40)	0.366667
(3,6)	0.300000	(30,39)	1.000000
(3,7)	0.375000	(31,38)	0.650000
(3,8)	0.191667	(31,39)	0.350000
(4,8)	0.100000	(32,37)	0.475000
(4,9)	0.216667	(32,38)	0.525000
(4,10)	0.166667	(33,36)	0.483333
(4,11)	0.225000	(33,37)	0.516667
(4,12)	0.150000	(34,34)	0.158333
(4,13)	0.075000	(34,35)	0.516667
(4,14)	0.058333	(34,36)	0.325000
(4,15)	0.008333	(35,31)	0.083333
(5,15)	0.050000	(35,32)	0.316667
(5,16)	0.041667	(35,33)	0.366667
(5,17)	0.016667	(35,34)	0.233333
(5,23)	0.008333	(36,15)	0.058333
(5,24)	0.041667	(36,16)	0.016667
(5,25)	0.033333	(36,17)	0.033333
(5,26)	0.075000	(36,18)	0.016667
(5,27)	0.058333	(36,19)	0.008333
(5,28)	0.125000	(36,21)	0.008333
(5,29)	0.158333	(36,24)	0.016667
(5,30)	0.241667	(36,25)	0.033333
(5,31)	0.150000	(36,26)	0.050000
(6,31)	0.091667	(36,27)	0.050000
(6,32)	0.266667	(36,28)	0.100000
(6,33)	0.383333	(36,29)	0.158333
(6,34)	0.258333	(36,30)	0.216667
(7,34)	0.208333	(36,31)	0.233333
(7,35)	0.600000	(37,8)	0.066667
(7,36)	0.191667	(37,9)	0.333333
(8,36)	0.550000	(37,10)	0.125000
(8,37)	0.450000	(37,11)	0.175000
(9,37)	0.583333	(37,12)	0.116667
(9,38)	0.416667	(37,13)	0.083333
(10,38)	0.691667	(37,14)	0.066667
(10,39)	0.308333	(37,15)	0.033333
(11,39)	1.000000	(38,5)	0.091667
(12,39)	0.658333	(38,6)	0.366667
(12,40)	0.341667	(38,7)	0.383333
(13,40)	1.000000	(38,8)	0.158333
(14,40)	1.000000	(39,3)	0.366667
(15,40)	1.000000	(39,4)	0.433333
(16,40)	1.000000	(39,5)	0.200000
(17,40)	1.000000	(40,40)	1.000000
(18,40)	1.000000		
(19,40)	1.000000		
(20,40)	1.000000		
(21,40)	1.000000		
(22,40)	1.000000		
(23,40)	1.000000		

APPENDIX C. Perturbation Method for the Theoretical Model

The perturbation method employed in this paper is similar to that of [7]. A number of important differences exist, though, so the precise perturbation procedure for this paper will be described below. This perturbation method is designed to determine the long-term effects of decreasing (or increasing) certain transition probabilities from a particular state while proportionally increasing (decreasing) other transition probabilities from that state. This proportional increase (decrease) is necessary to preserve the stochasticity of the TPMs. The work of [7] examines perturbations of combinations of two rows, but for the purposes of this paper, only perturbations affecting a single row, i.e. affecting transitions out of a single state, will be considered for the sake of simplicity. Restricting to these single perturbations makes it somewhat more difficult to precisely model the behavior of the large-scale grid system since inducing different execution paths in the large-scale system often results in changes to the transition probabilities for multiple states. Despite this shortcoming, though, the perturbation method used here does appear to at least qualitatively predict the effect of key changes to the large-scale system.

With this perturbation method, each of the transient states will be selected to have transitions from that state perturbed. The perturbation algorithm iterates through each of the different rows corresponding to the transient states, and the current state to be perturbed is given by the *perturbation row*, r. Within that row, the algorithm iteratively selects an entry to decrease, known as the *decrease column*, $c^{\downarrow}$, as well as an entry to increase, known as the *increase column*, $c^{\uparrow}$. Here, $c^{\downarrow}$ and $c^{\uparrow}$ are required to be allowable transitions for the grid system, i.e. the entries $(r, c^{\downarrow})$ and $(r, c^{\uparrow})$ must be non-zero in at least one of the TPMs.

In the *primary decrease perturbation method*, the $(r, c^{\downarrow})$ entry will be decreased down to zero by an increment v in all of the TPMs. Meanwhile, each decrease by v must be accompanied by an increase of v distributed across the other entries in row r so that the row sum will still equal 1. A weighted portion of that increase will be added to the entry in the increase column, $(r, c^{\uparrow})$. The weight, w, for the increase was varied between 0.1 and 0.9, but a weight of 0.5 was predominantly used. The entry $(r, c^{\uparrow})$ was then increased by $w * v$ at each step of the perturbation. The remainder of the increase was then distributed to the other entries in row r in an amount proportional to the original value of the entry. The new transition probabilities derived from these incremental perturbations are summarized below

$$p_{rj}^{(\text{new})} = \begin{cases} p_{rj}^{(\text{old})} - v & \text{if } j = c^{\downarrow} \\ p_{rj}^{(\text{old})} + w * v & \text{if } j = c^{\uparrow} \\ p_{rj}^{(\text{old})} + (1 - w) * \frac{p_{rj}^{(\text{old})}}{\sum_{k \neq c^{\downarrow}, c^{\uparrow}} p_{rk}^{(\text{old})}} & \text{else} \end{cases}$$

There are a couple of important situations, though, where the values of $p_{rj}^{(\text{new})}$ slightly deviate from those described above. The first case is when $p_{rc^{\downarrow}}^{(\text{old})} - v$ is negative for some TPM. In this event, the value of v used for that TPM is set to equal the old transition probability, $p_{rc^{\downarrow}}^{(\text{old})}$, and this new value for v is distributed as an increase among the other entries in the row as described previously. The other special case occurs when there are only two non-zero entries in a particular row, e.g. the only allowable transitions from the state *Initial* are the self-transition and the transition to *Discovering*. In this event, the *increase column* will be increased by v rather than by $w * v$ since there are no other non-zero entries to distribute the increase among. Notice that there is never a special case of increasing the *increase column* by too much, i.e. of making that entry greater than

1. This is not an issue because the increase to the entry $(r, c^{\uparrow})$ is equal to $w * v$ for $w \leq 1$ and v is capped by the value of the $(r, c^{\downarrow})$ entry, where the sum of the entries $(r, c^{\downarrow})$ and $(r, c^{\uparrow})$ is less than or equal to 1 since the TPMs are stochastic.

A second type of perturbation method, the *primary increase perturbation method*, was also used where the primary focus was on increasing a particular transition up to be just less than 1 by increments of v while distributing a weighted decrease to the other non-zero entries in that row.[1] For this perturbation method, the new transition probabilities are as follows

$$p_{rj}^{(\text{new})} = \begin{cases} p_{rj}^{(\text{old})} + v & \text{if } j = c^{\uparrow} \\ p_{rj}^{(\text{old})} - w * v & \text{if } j = c^{\downarrow} \\ p_{rj}^{(\text{old})} - (1-w) * \frac{p_{rj}^{(\text{old})}}{\sum_{k \neq c^{\downarrow}, c^{\uparrow}} p_{rk}^{(\text{old})}} & \text{else} \end{cases}$$

Similar to the first type of perturbation, there are a few situations where the values of $p_{rj}^{(\text{new})}$ slightly deviate from those described. The first case occurs when $p_{rc^{\uparrow}}^{(\text{old})} + v$ is greater than 1 for some TPM. In this case, the value for v is set to $1 - p_{rc^{\uparrow}}^{(\text{old})}$ so that $p_{rc^{\uparrow}}^{(\text{new})}$ will equal 1 and the decrease by v will be distributed across the other entries in that row in the same manner described previously. The special case when there are only two non-zero entries in the perturbation row is handled in the same fashion as outlined above. Finally, there is an additional situation to consider for this type of perturbation. For this type of perturbation, it is possible for $w * v$ to be greater than the $(r, c^{\downarrow})$ entry so that the weighted decrease would perturb that entry to be negative for a particular TPM. In this event, the weight w is modified for that specific TPM so that $w * v = p_{rc^{\downarrow}}^{(\text{old})}$; then, the $(r, c^{\downarrow})$ entry is perturbed down to zero while the other entries in that row take on a larger proportion of the decrease by v. Additionally, it is possible that $(1 - w) * v$, the amount of decrease to be distributed throughout the rest of the row, is greater than the total values in that row; thus, the standard method for perturbing those entries would cause them to become negative. To remedy this situation, each of the entries in the row other than $(r, c^{\uparrow})$ and $(r, c^{\downarrow})$ are perturbed to zero resulting in a decrease by $\sum_{k \neq c^{\downarrow}, c^{\uparrow}} p_{rk}^{(\text{old})}$. Then the remainder of the decrease is distributed to $p_{rc^{\downarrow}}$ so that $p_{rc^{\downarrow}}^{(\text{new})} = p_{rc^{\downarrow}}^{(\text{old})} - v + \sum_{k \neq c^{\downarrow}, c^{\uparrow}} p_{rk}^{(\text{old})}$.

Now, using the perturbed matrices created by the appropriate perturbation algorithm, the resulting Markov chain was utilized to determine the proportion of tasks completed by this new system. Additionally, the perturbed matrices were used to find the theoretical approximation to the proportion of tasks completed using Equation 6. The value for proportion of tasks completed for the Markov model and the theoretical model were then recorded together with a summary value for the entry $(r, c^{\downarrow})$ (or $(r, c^{\uparrow})$, depending on the perturbation method used) from the perturbed TPMs. This summary value was calculated as a weighted average of the $(r, c^{\downarrow})$ (or $(r, c^{\uparrow})$) entries across all the TPMs, using the formula described by [7] for creating a summary matrix. Perturbations were performed using both the primary decrease and primary increase methods for every allowable combination of $r, c^{\downarrow}$, and $c^{\uparrow}$ where r ranged among the transient states. For each of these perturbations, the proportion of tasks completed was then plotted against the varying summary values for the perturbed entry. The figures shown in section 6 of this report show examples of these plots together with large-scale simulation data for these perturbations.

[1] The transition probabilities were not perturbed to equal one because this often caused a drastic change in the proportion of tasks completed that was not reflective of the change that occurred when the transition probability was just fractionally smaller.

It is important to note that the large-scale simulation data was obtained by manually adjusting different aspects of the grid simulation such as the length of time required to negotiate an SLA, etc. in order to mimic the effect of decreasing (or increasing) a particular transition. As mentioned earlier, these changes often affected transition probabilities other than the particular transitions to be perturbed, and thus the perturbed TPMs are not an exact reflection of the TPMs that would be derived for the altered large-scale simulation. As a result, with the simple perturbation method used here, the data from the perturbed Markov model does not match that of the perturbed large-scale simulation as well as in the unperturbed case. The qualitative behavior does appear to match, however, and thus the perturbed Markov model provides a useful tool for determining which state transitions have the greatest deleterious effects on the proportion of tasks completed by the grid system.