USING MARKOV CHAIN AND GRAPH THEORY CONCEPTS TO ANALYZE BEHAVIOR IN COMPLEX DISTRIBUTED SYSTEMS

Christopher Dabrowski[(a)] and Fern Hunt[(b)]

U.S. National Institute of Standards and Technology

[(a)]cdabrowski@nist.gov, [(b)]fhunt@nist.gov

ABSTRACT

We describe how a Discrete Time Markov chain simulation and graph theory concepts can be used together to efficiently analyze behavior of complex distributed systems. Specifically, the paper shows how minimal s-t cut set analysis can be used to identify state transitions in a directed graph of a time-inhomogeneous Markov chain, which when suitably perturbed, lead to performance degradations in the system being modeled. These state transitions can be then be related to failure scenarios in which system performance declines catastrophically in the target system being modeled. Using a large-scale simulation of the grid system, we provide examples of the use of this approach to identify failure scenarios. Preliminary experiments are reported that show this approach can be applied to problems of significant size. The approach described here combines techniques whose use together to analyze dynamic system behavior has not previously been reported.

Keywords: time-inhomogeneous Discrete Time Markov chain; distributed system; minimal s-t cut set.

1. INTRODUCTION

In large-scale, dynamic distributed systems, such as computing grids, the interactions of many independent components can lead to emergent system-wide behaviors with unforeseen, often detrimental, outcomes (Mills and Dabrowski 2008). To ensure availability and reliability of computing services in such environments, new techniques will be needed to rapidly assess trends and predict changes in system behavior caused by such factors as shifts in workload, modifications to system configurations, policy changes, or failures.

In earlier work (Dabrowski and Hunt 2009), we described a succinct Discrete Time Markov chain (DTMC) representation for analyzing the behavior of a grid computing system in order to identify potential failure scenarios in which system-wide performance collapses. In this representation, the stochastic characteristics of Markov chains were used to summarize the evolving state of a system, in which dozens of users and grid service providers interacted to process over 1000 grid computing tasks over simulated time durations (Mills and Dabrowski 2008). To capture change in system behavior over time, the DTMC representation was made *time inhomogeneous*–also referred to as *piecewise homogeneous* (Rosenberg, Solan, and Vielle 2004)–in which a set of transition probability matrices (TPMs) was used to model successive time periods. The time-inhomogeneous TPM set could be perturbed by systematically changing the values of related state transition probabilities to examine alternative system execution paths. State transitions were deemed *critical state transitions* if they could be perturbed to cause system performance to decline drastically. Once identified, these critical transitions could then be related to events such as faults, policy changes, and workload shifts, in order to describe failure scenarios in a target system being modeled. The perturbed TPM set could be used to simulate the rate at which performance declines in response to such events and to establish thresholds, beyond which increased incidence of failure caused performance collapse. This initial approach, however, required exhaustive search of the TPM set in order to find failure scenarios.

To overcome this limitation, this paper extends (Dabrowski and Hunt 2009), to analyze the DTMC as a directed graph and to use minimal s-t cut set analysis (Tsukiyama, Shirakawa, Ozaki, and Ariyoshi 1980) to identify critical state transitions, which if perturbed, reveal potential performance collapses in the target system being modeled. We show that the use of minimal s-t cut set analysis reduces the computation needed to find critical transitions, and can thus be applied to more complex problems in comparison to the exhaustive search methods used in our initial approach (Dabrowski and Hunt 2009). Further, we show that minimal s-t cut analysis can also find combinations of critical transitions that represent more complicated failure scenarios, which our initial approach also could not do. In experiments, our new approach is applied to analyze grid system simulations with different durations and workloads. We assess the use of minimal s-t cut set analysis to predict failure scenarios, using a detailed, large-scale grid simulation as a proxy for a real-world system. The application of minimal cut set analysis to a grid system parallels our use of this method to analyze dynamic behavior in cloud computing systems, which we report in (Dabrowski and Hunt 2011). The use of this method in two different domains is essential to investigating the generality of the approach.

This paper also considers whether minimal s-t cut set analysis can be applied to large problems. Since larger directed graph problems can potentially contain many minimal s-t cut sets, we investigate the use of a cut set identification algorithm that can be bounded to run within reasonable time limits. We evaluate the effectiveness of this algorithm in identifying critical transitions in DTMCs with up to 50 states and 160 state transitions. To our knowledge, the use of minimal s-t cut set analysis to guide perturbation of a time-inhomogeneous DTMC has not previously been studied as an approach for analyzing dynamic behavior in complex distributed systems.

This paper is organized as follows. Section 2 reviews related work on using Markov chains to analyze dynamic systems. Section 3 summarizes the DTMC for the grid system example used here. Section 4 defines minimal s-t cut sets and describes their use in finding critical state transitions. Section 5 presents an algorithm for computing minimal s-t cut sets in large DTMC graphs and analyzes its performance. Section 6 discusses future work and concludes.

2. RELATED WORK

The method discussed in this paper is distinguishable from the well-known use of DTMCs to provide quantitative measures of system performance and reliability, which we review in (Dabrowski and Hunt 2009). Instead of measuring system reliability, we use DTMCs to examine alternative execution paths in dynamic systems in order to identify failure scenarios.

Both perturbation analysis and graph theory have previously been applied to DTMCs, but for different purposes than we intend. Perturbation analysis of DTMCs has been the topic of theoretical (Schweitzer 1968; Hassin and Haviv 1992) and computational study (Meyer 1989; Stewart and Dekker, 1994). Other researchers have used system performance gradients that are based on key decision parameters to perturb Markov models (Ho and Li 1988; Suri 1989; Cao and Zhang 2008). While gradient-based approaches demonstrated potential in modeling performance change, some issues involving computation of gradients required further research to fully resolve (Cao and Zhang 2008). Also, gradient-based approaches appear to be geared for system optimization, rather than for examining alternative execution paths to identify situations in which performance degrades.

Graph-theoretic methods have also been used previously to study dynamic behavior in Markov chain models. For example, graph decomposition has been used to calculate stationary probability distribution vectors of Markov chains (Benzi and Tuma 2002; Gambin, Kryzanowski and Pokarski 2008; as well as to measure how perturbation affects stationary distributions (Solan and Vielle 2003). Minimal cut set analysis has been used on topology graphs of avionics system components to identify the shortest sequence of component failures (Tang and Dugan 2004). However, these applications of graph theory have been targeted for analysis of specific subsystems in their respective domains, rather than using minimal s-t cut set analysis to identify global failure scenarios in the manner we envision. Finally, there are maximum-flow algorithms (Ford and Fulkerson 1962; Goldberg and Tarjan 1988), well-known graph-theoretic methods that find s-t cut sets on the basis of flow levels. These algorithms could potentially be used to identify critical state transitions. However, because these algorithms use flows, they are distinguishable from Markov chain approaches and so best merit separate investigation.

3. THE DISCRETE TIME MARKOV CHAIN

The DTMC model of a grid system was developed by observing a large-scale grid computing simulation (Mills and Dabrowski 2008). This section overviews the DTMC model, with full details in (Dabrowski and Hunt 2009). The DTMC model of the grid system simulates the progress of over 1000 computing tasks from the time they are submitted by a user to the grid for execution to the time they either complete or fail. Figure 1 shows a state diagram of this system, which describes the lifecycle of a single task. This model has 7 states: an *Initial* state, where a task remains prior to submission; a *Discovering* state, during which service discovery directories are accessed to locate grid service providers who are able to execute the task; a *Negotiating* state during which a Service Level Agreement (SLA) for task execution is negotiated with a provider; a *Waiting* state in which tasks reside that are temporarily unsuccessful in discovery or negotiation; and a *Monitoring* phase in which a provider who has entered into an SLA executes a task by a deadline. Transitions between states, shown in Figure 1 by arrows, represent actions taken by the grid system to process a task. All tasks eventually enter either the *Tasks Completed* or *Tasks Failed* state, which are the *absorbing states* of the Markov chain, because once entered, a task cannot leave. A Markov chain with these characteristics is called an *absorbing chain*.

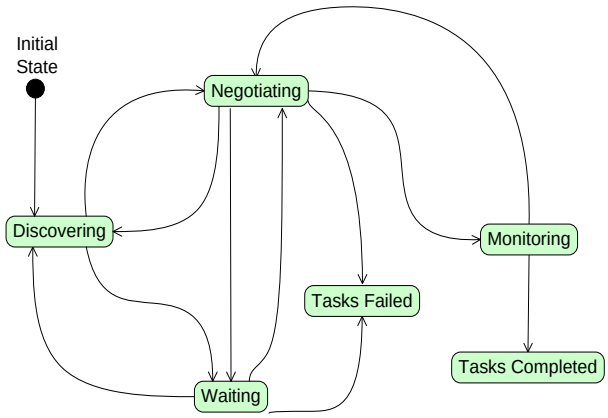


Figure 1: State model of grid computing system.

The large-scale grid simulation was observed over extended durations to accumulate frequencies for the state transitions shown in Figure 1, compute transition probabilities, and form TPMs. We computed transition probabilities by determining where state transitions

occur in the large-scale model code and inserting counters at those places. State transition probabilities were derived as follows. Given states s_i, s_j, $i,j = 1 \ldots n$ where n=7, p_{ij}, is the probability of transitioning from state i to state j, written as $s_i \rightarrow s_j$. This probability is estimated by calculating the frequency of $s_i \rightarrow s_j$, or f_{ij}, and dividing by the sum of the frequencies of s_i to all other states s_k, as shown in equation (1)

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

Here i and j may be equal, to allow for self transitions, which are counted if the task process remained in a state longer than a discrete time step, chosen to be 85 s. The resulting TPM is a 7 × 7 stochastic matrix, where rows stand for the state the transition originates from, or the *from state*, *i*, and columns represent states the transition goes to, or the *to state*, *j*. Figure 2 shows an example of such a TPM. Each element in this TPM contains a p_{ij}, where i and j are the *from* and *to* states, respectively. As in any stochastic TPM, the transition values of all row elements must sum to 1.0.

(a)

	Initial	Wait	Disc	Ngt	Mon	Comp	Failed
Initial	0.9697	0	0.0303	0	0	0	0
Wait	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
Failed	0	0	0	0	0	0	1.0

(b)

	Initial	Wait	Disc	Ngt	Mon	Comp	Failed
Initial	0.9997	0	0.0003	0	0	0	0
Wait	0	0.7612	0.0460	0.1911	0	0	0.0017
Disc	0	0.0686	0.6084	0.3230	0	0	0
Ngt	0	0.2401	0.0062	0.2378	0.4801	0	0.0358
Mon	0	0	0	0.0007	0.9902	0.009	0
Comp	0	0	0	0	0	1.0	0
Failed	0	0	0	0	0	0	1.0

Figure 2: (a, b) Summary TPMs for the grid system over (a) 8- and (b) 640-hour durations. The summary TPMs are weighted averages of their component time period TPMs, in which the weight of each time period TPM is determined by the relative number of transitions in the time period.

Separate observations were made to create two cases: (a) one in which the system executes for 8 hours with varying workload; and (b) a 640-hour execution that reaches near steady state. To create time-inhomogeneous representations for the two cases, the total duration of each was divided into equal periods of 7200 s and a TPM was computed for each period. Figure 2 shows the summary TPMs for the two cases, which are weighted averages of their respective time period TPMs. The weights are based on the relative number of transitions in each period.

3.1. Simulating System Behavior

To simulate system behavior over time, a well-known DTMC method was employed, which we refer to as *Markov simulation*. For further description, see (Hunt, Morrison, and Dabrowski 2011; Dabrowski and Hunt 2009). In Markov simulation, multiplication of time period TPMs is used to advance the system state in discrete time steps of a fixed duration, h. Here, h = 85 s. Since a time period covers a duration of d_{period} = 7200 s, each time-period TPM is made to represent S= d_{period}/h, or 85, steps. Thus, an 8-hour Markov simulation, with a 2-hour period for residual clean-up, covers 5 time periods, consisting of a total of 425 time steps. Correspondingly, a 640-hour Markov simulation with a clean-up period covers 321 time periods with over 27, 000 time steps.

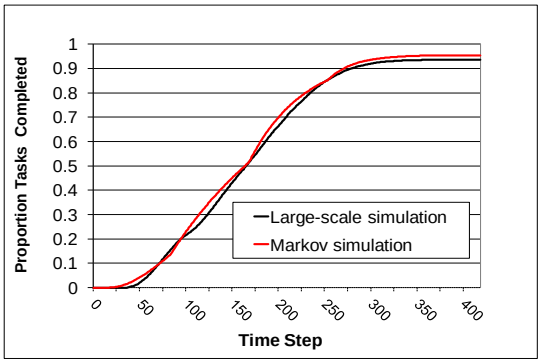
In Markov simulation, the state of the system can be summarized at any time step in an n-element state vector v, where n is the number of states in the related Markov chain. Each of the n elements in v represents the proportion of tasks in one of the n states of the DTMC. For the Markov chain of the grid system, the n=7 elements are ordered so as to correspond to the states in Figure 1. Thus, the first element in v contains the proportion of tasks in the *Initial* state, the second contains the proportion of tasks in the *Waiting* state, and so forth. In Markov simulation, the vector v represents the system state at different time steps, such that the vector v_m represents the system state at time step m. To evolve the system state by one discrete time step, the vector v_m is multiplied by the TPM, Q^{tp}, for the applicable time period tp to produce a new system state v_{m+1}, as shown in equation (2):

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

where T indicates a matrix transpose. Starting with v_1, which represents a system state with a value of 1.0 for the *Initial* state (see Figure 1) and 0 for all other states (i.e., all tasks begin in the *Initial* state), equation (2) is repeated for 425 time steps to evolve the system state over 8 simulated hours. This results in a system state vector, v_{425} at the end of the simulated 8 hours. To simulate 640 hours, equation (2) is repeated for 27, 000 time steps to produce a state vector, v_{27000}, at the end of 640 hours. In both cases, repeated application of equation (2) causes the proportion of tasks to be distributed over the 6 states other than the *Initial* state (i.e., all states have transitioned out of *Initial*). In an absorbing chain, tasks eventually transition into one of the absorbing states, i.e., the *Tasks Completed* or *Failed* states, where they remain permanently. Thus, a measure of the performance of a system is the proportion of tasks that enter the *Tasks Completed* state, because this absorbing state represents tasks that have succeeded. On the other hand, a performance collapse may be

simulated either when a large proportion of tasks enter the *Task Failed* state, or when they are otherwise prevented from entering *Tasks Completed*. Figure 3 shows that *Markov simulation* of the grid system closely approximates the performance of a large-scale simulation in both the 8- and 640-hour cases in terms of proportion of tasks that enter the *Tasks Completed* state.

(a)



(b)

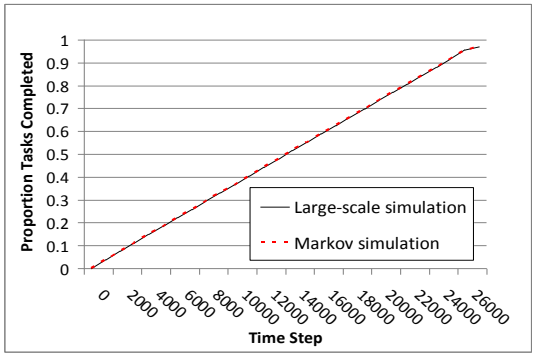


Figure 3: Performance of Markov chain and large-scale simulations as measured by *Tasks Completed* over: (a) 8 hours (5 time-period TPMs – 421 time steps with an extra cleanup period), and (b) 640 hours (321 time-period TPMs – 27000 time steps with cleanup). A time period represents 7200s and a time step represents 85 s.

3.2. Perturbing Critical State Transitions

To identify critical state transitions, we described a perturbation algorithm in (Dabrowski and Hunt 2009), which systematically raises and lowers all feasible combinations of non-zero state transition probabilities in individual rows of a TPM for a DTMC. The different combinations of changed values are then evaluated by Markov simulation in order to explore potential alternative system executions. This algorithm can be applied to exhaustively perturb all feasible combinations of state transition probabilities in all rows of a TPM. The algorithm outputs a set of individual critical state transitions, which when perturbed to extreme values, cause system performance to degrade drastically. We must omit the full description to the perturbation algorithm due to lack of space. In (Dabrowski and Hunt 2009), we showed that exhaustive application of this algorithm could replicate (with good agreement) scenarios in which performance drastically degraded in the large-scale grid simulation.

Figure 4 provides an example of a critical state transition, *Negotiating* → *Monitoring*, identified by the perturbation algorithm. The figure shows the impact of a set of related perturbations, in which lowering the probability of transition to 0 for *Negotiating* → *Monitoring* causes the proportion of *Tasks Completed* to fall to 0 in the Markov simulation (blue curves). The perturbation of this transition models a failure scenario in which negotiations for SLAs fail, due to events such as system-wide viruses which gradually affect all providers; hence, tasks cannot progress to the *Monitoring* state. Figure 4 also shows the result altering the target large-scale grid simulation (red curve), to randomly fail negotiations with systematically increased incidence. The figure shows that both the Markov and large-scale simulation curves exhibit low thresholds for the rate of successful negotiation, below which there is a sharp drop in *Tasks Completed*. In the Markov simulation this threshold is below 0.05, while in the large-scale simulation, the threshold is slightly higher at 0.15. However, both curves are sufficiently similar, so that the perturbation algorithm could be used to forecast that increased incidence of failed negotiation will eventually lead to a system performance collapse. In (Dabrowski and Hunt 2009), we provide the complete results of applying the perturbation algorithm. Though its computational cost prohibits use on large problems, the perturbation algorithm provides a baseline for assessing the use of minimal s-t cut set analysis.

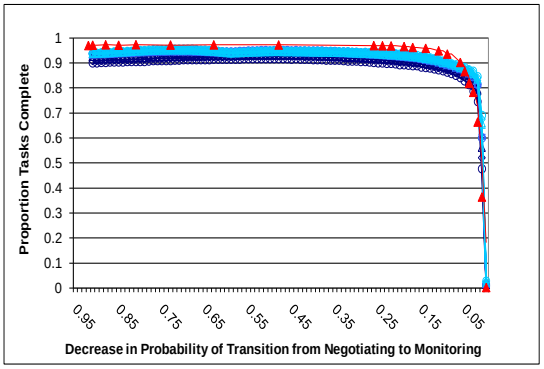


Figure 4: Perturbation of *Negotiating* State to reduce the probability of transition from *Negotiating* → *Monitoring* while raising the probability of transition from *Negotiating* → *Waiting* in the 640-hour case. The blue curve shows the proportion of *Tasks Completed* estimated by the perturbation algorithm. Large-scale simulation results are denoted by red triangles.

4. MINIMAL S-T CUT SET ANALYSIS IN A MARKOV CHAIN MODEL

This section describes how identifying minimal s-t cut sets on paths between an *Initial* state and a desired absorbing state can be used to identify critical state transitions in a DTMC, which if perturbed, lead to system performance degradations. In contrast to the perturbation algorithm, which can identify only single state transitions that are critical, minimal s-t cut set analysis identifies combinations of critical state

transitions, an important benefit for analysis of more complex problems. In Section 5, we describe an algorithm for finding minimal s-t cut sets and show its effectiveness for large problems. Our approach does not use flow levels to identify minimality, but instead uses cardinality and other factors discussed below.

4.1. Definitions

In graph theory, a graph G (V, E) consists of a set of vertices V connected by edges from the set E. A directed graph is a graph in which edges can be traversed in only one direction. A Markov chain is a directed graph, in which vertices correspond to states and directed edges correspond to state transitions. A directed path through this graph is a sequence of state transitions from one state to another. In this problem, the directed paths of most interest are non-cyclic paths that lead from the *Initial* state to one of the two absorbing states: *Tasks Completed* or *Tasks Failed*. This paper considers only paths to *Tasks Completed*. Figure 5 shows two such paths.

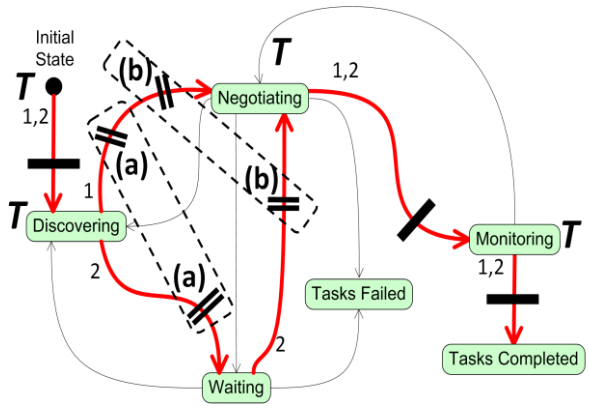


Figure 5: There are 2 directed paths (in red) from the *Initial* state to *Tasks Completed*, labeled 1 and 2. Three single-transition s-t cuts (minimal s-t cut sets consisting of one state transition) are marked by single bars. Two multiple transition s-t cuts (s-t cut sets with two transitions each) are marked by double bars: (a) *Discovering* → *Negotiating* and *Discovering* → *Waiting*; and (b) *Discovering* → *Negotiating* and *Waiting* → *Negotiating*. Trap states are denoted by ***T***.

A set of one or more edges, which if removed, disconnects all paths between two vertices *s* and *t* is referred to as an *s-t cut set* (Tsukiyama, Shirakawa, Ozaki, and Ariyoshi 1980). An s-t cut set is a *minimal s-t cut set* if removal of any edge from the cut set reconnects *s* and *t*. By finding minimal s-t cut sets consisting of state transitions that disconnect the *Initial* and *Tasks Completed* states, it is possible to know where reducing the related transition probabilities to 0 prevent tasks from completing. In this paper, minimal s-t cut sets with a single member will be referred to as *single-transition s-t cuts*, while those with more than one member are *multiple-transition s-t cuts*. State transitions that are members of a minimal s-t cut set are critical state transitions as defined above.

4.2. Identifying Minimal s-t Cut Sets in the Grid Markov Chain Model

In Figure 5, there are 3 single-transition s-t cuts: *Initial* → *Discovering*, *Negotiating* → *Monitoring*, and *Monitoring* → *Tasks Completed*. Figure 4 shows that reducing the probability of transition for *Negotiating* → *Monitoring* to 0 using Markov simulation causes the proportion of tasks reaching *Tasks Completed* to drop to 0. The same result occurs when the other two single-transition s-t cuts, *Initial* → *Discovering* and *Monitoring* → *Tasks Completed*, are similarly perturbed (see Dabrowski and Hunt 2009). Use of the exhaustive perturbation algorithm confirmed that the 3 single-transition s-t cuts identify state transitions, which if reduced to 0, cause the proportion of tasks reaching the *Tasks Completed* state to fall to 0 (see Section 5). These 3 single-transition s-t cuts are critical state transitions that clearly relate to resource allocation and task execution functions. Figure 5 also shows two multiple-transition s-t cuts, labeled (a) and (b), which disconnect all paths between the *Initial* from the *Tasks Completed* state. Both multiple-transition cuts consist of two transitions. In a multiple-transition s-t cut, lowering transition probabilities to 0 of all transitions in the cut set reduces the proportion of *Tasks Completed* to 0. Multiple-transition s-t cuts identify situations where a combination of state transitions is critical and together describe circumstances that degrade system performance. We return to multiple-transition s-t cuts in Section 5.

4.3. Identifying Trap States

The previous discussion considered only state transitions between different states. However, in a DTMC, a state may also transition to itself in the next discrete time step and remain in the same state. In this paper, this is referred to as a *self-transition*. If a self-transition probability is near 1, the task may stay in the state for a long time. Such a state effectively becomes a *trap state*. Figure 6 shows an example of how a *trap state* affects performance, when the self-transition probability of the *Discovering* state is raised to 1. As the self-transition probability approaches 1, tasks are increasingly stalled in *Discovering*, so that they cannot proceed to other states and complete by their deadlines. The evolution of *Discovering* into a trap state may correspond to a real-world failure scenario in which service discovery is impaired by directory failures, so that information about existing grid services cannot be retrieved. As the incidence of directory failures increases, the length of time to complete service discovery for all tasks also increases, until finally no task can progress beyond the discovery stage to begin negotiation. Figure 6 also shows how the large-scale simulation behaves when the equivalent failure is introduced. In the latter, the failure is modeled by systematically increasing the frequency of directory access failure. As in the example discussed in Section 3.2, the perturbation shown in Figure 6 predicts how this failure scenario impacts the large-scale simulation.

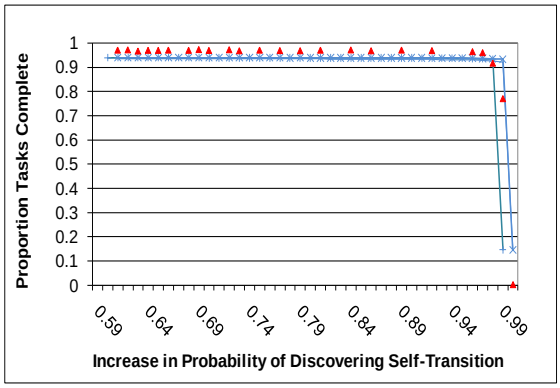


Figure 6: Perturbation of *Discovering* to increase the self-transition probability to 1, while decreasing the transition probability from *Discovering* to other states to 0 in 640-hour case. The blue curve shows the proportion of tasks completed as estimated by the perturbation algorithm. Values from the large-scale simulation are shown by red triangles.

A trap states is distinguishable from a permanent absorbing state, because the latter always has a self-transition probability of 1, while in the former, the transition probability varies. The concept of an s-t cut set can be extended to include vertices whose removal cuts all paths from *s* to *t*. A minimal set of such elements (edges and vertices) is a *minimal s-t separating set* (Hayakawa, Tsukiyama and Ariyoshi 1999) a topic we leave for future work.

5. PERFORMANCE OF MINIMAL S-T CUT SET ANALYSIS

This section shows that minimal s-t cut set analysis is an effective and efficient means of finding critical state transitions and trap states. The section also shows the potential applicability of this method to large problems. Section 5.1 first describes a well-known algorithm, known as the *node contraction algorithm*, which we adapted to find minimal s-t cut sets in a directed graph. The node contraction algorithm is considered probabilistic (Karger and Stein 1996), because it can find solutions with a probability which can be increased to a high level by repeatedly executing the algorithm. Though it is not guaranteed to find all minimal s-t cut sets, the computational cost of node contraction can be bounded, making it potentially applicable to large problems where use of exhaustive methods would be infeasible. The node contraction algorithm also finds critical transitions that are part of multiple transition s-t cuts, which the perturbation algorithm cannot find.

Section 5.2 shows effectiveness and efficiency of node contraction for the grid system problem. Here, the node contraction algorithm is able to find all individual critical transitions and trap states that were found using the perturbation algorithm, but at far less cost. Section 5.3 then reports experiments on the use of node contraction for finding critical state transitions in large Markov chain problems. To do this, the performance of the node contraction algorithm is tested by comparing it to the performance of an algorithm (Provan and Shier 1996) that, unlike node contraction, enumerates *all* minimal s-t cut sets in a directed graph, and thus finds all critical transitions. While, like other algorithms of this type, the time complexity of (Provan and Shier 1996) prohibits practical use on many large problems, the algorithm provides a good baseline for testing the effectiveness of node contraction. (The complexity of the minimal s-t cut set enumeration algorithm described in (Provan and Shier 1996) is $O\ |E|$ for each s-t cut set that exists, where $|E|$ is the number of edges in the graph). To examine the potential for scalability of minimal s-t cut set analysis, we wish to know what proportion of minimal s-t cut sets (and thus critical transitions) can be found by node contraction in large problems and the related computational cost.

5.1. Overview of the Node Contraction Algorithm

In this section, we summarize our implementation of the node contraction algorithm, with pseudo-code in (Dabrowski, Hunt and Morrison 2010). Though the time complexity of node contraction algorithms for directed graphs has not been studied, efficient versions of this algorithm for undirected graphs find a minimum cut with a complexity of $O\ |V|^2$, where $|V|$ is the number of vertices in the graph (Karger and Stein 1996). While this cost is significant, the algorithm can be used on large problems by controlling the number of executions, as will be discussed in Section 5.3.

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 result of each contraction is recorded. 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. The node contraction algorithm was modified for an absorbing Markov chain problem to prevent the two vertices representing the *Initial* state, *s*, and the *Tasks Completed* absorbing state, *t*, from being contracted into 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 a minimal s-t cut set of state transitions, which if removed, disconnect the *Initial* and absorbing state (*Tasks 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 is

considered 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 originate in one state. Perturbation then need be applied only to the transitions in the s-t cut set, in order to generate curves of tasks completed, such as appear in Figures 4 and 6, and to identify performance thresholds.

5.2. Comparing the Perturbation Algorithm with Minimal s-t Cut Set Analysis

Table 1 compares the result of applying the perturbation algorithm described in Section 3 with the result of minimal s-t cut set analysis using node contraction, when both are used to identify individual critical state transitions and trap states. The perturbation algorithm was applied to the 5 rows representing non-absorbing states (labeled a-e) in the time period TPMs for the 8- and 640-hour cases. The combinations of row elements representing the transition probability being decreased and increased appear in the two leftmost columns. For each such combination of transitions, the next two columns show the proportion of *Tasks Completed* for the 8- and 640-hour cases as the transition probability being decreased falls to 0. The rightmost column indicates if the state transition being reduced corresponds to a single-transition s-t cut (see Figure 5).

Table 1 shows that all combinations where perturbation causes a decline in the proportion in *Tasks Completed* to 0 correspond to single- transition s-t cuts. There are 7 such combinations, and all correspond to single-transition s-t cuts that are verified by large-scale simulation. In no case, did node contraction find an s-t cut that did not correspond to such a drastic reduction. For instance, in Table 1(c), rows 10–12, when the probability of transition from the *Negotiating* state to *Monitoring* (i.e., *Negotiating* → *Monitoring*) is reduced to 0, the proportion of *Tasks Completed* falls to 0. This is shown in Figure 4. Figure 5 shows that *Negotiating* → *Monitoring* is a single-transition s-t cut.

Note that in Table 1(d), row 3, reducing the probability of *Monitoring* self-transition while raising the probability of *Monitoring* → *Negotiating* also caused a severe decline in the proportion of *Tasks Completed*. This happens because the probability of transition for *Monitoring* → *Tasks Completed* is very low (see Figure 2), and so the probability of *Monitoring* self-transition must be very high to ensure tasks remain in the *Monitoring* state long enough to eventually transition to *Tasks Completed*. Thus, reducing the probability of *Monitoring* self-transition to 0 while raising the probability of *Monitoring* → *Negotiating* prevents tasks from reaching *Tasks Completed*—and acts like a single-transition s-t cut on *Monitoring* → *Tasks Completed*. However, because the transition probability of *Monitoring* → *Tasks Completed* is not lowered to 0 by this perturbation, some tasks are able to complete. Table 1 also contains 3 rows that show only partial reductions (Table 1 (a), rows 5 and 6, and Table 1 (b), row 5). These correspond to the state transitions

Table 1: Correspondence of results of applying the perturbation algorithm to the TPMs for the 8- and 640-hour cases with single-transition s-t cuts found by the node contraction algorithm. The perturbations represented by individual rows correspond to single-transition s-t cuts in Figure 5 as follows: Table (c) rows 10–12 to *Negotiating* → *Monitoring*; Table (d) rows 3, 5, and 6 to *Monitoring* → *Tasks Completed*; and Table (e) row 1 to *Initial* → *Discovering*. Note also the explanation in the text for Table (d) row 3. Perturbations verified by large-scale simulation are bolded and shaded.

(a) *row* = *Discovering*

	Element reduced→0	Element raised	Proportion of *Tasks Complete* 8-hour	Proportion of *Tasks Complete* 640-hour	s-t cut exists
1	*Waiting*	*Discovering*	0.957	0.935	No
2	*Waiting*	*Negotiating*	0.959	0.935	No
3	*Discovering*	*Waiting*	0.939	0.935	No
4	*Discovering*	*Negotiating*	0.963	0.935	No
5	*Negotiating*	*Waiting*	0.894	0.933	No
6	*Negotiating*	*Discovering*	0.651	0.932	No

(b) *row* = *Waiting*

	Element reduced→0	Element raised	Proportion of *Tasks Complete* 8-hour	Proportion of *Tasks Complete* 640-hour	s-t cut exists
1	*Waiting*	*Discovering*	0.974	0.937	No
2	*Waiting*	*Negotiating*	0.981	0.939	No
3	*Discovering*	*Waiting*	0.937	0.934	No
4	*Discovering*	*Negotiating*	0.963	0.936	No
5	*Negotiating*	*Waiting*	0.818	0.843	No
6	*Negotiating*	*Discovering*	0.939	0.932	No

(c) *row* = *Negotiating*

1	*Waiting*	*Discovering*	0.974	0.937	No
2	*Waiting*	*Negotiating*	0.985	0.938	No
3	*Waiting*	*Monitoring*	1.000	0.939	No
4	*Discovering*	*Waiting*	0.954	0.935	No
5	*Discovering*	*Negotiating*	0.957	0.935	No
6	*Discovering*	*Monitoring*	0.967	0.936	No
7	*Negotiating*	*Waiting*	0.923	0.931	No
8	*Negotiating*	*Discovering*	0.941	0.933	No
9	*Negotiating*	*Monitoring*	0.988	0.938	No
10	*Monitoring*	*Waiting*	***0.000***	***0.000***	**Yes**
11	*Monitoring*	*Discovering*	***0.000***	***0.000***	**Yes**
12	*Monitoring*	*Negotiating*	***0.000***	***0.000***	**Yes**

(d) *row* = *Monitoring*

1	*Negotiating*	*Monitoring*	0.982	0.937	No
2	*Negotiating*	*Tasks Comp*	0.982	0.938	No
3	*Monitoring*	*Negotiating*	***0.028***	***0.186***	**Yes**
4	*Monitoring*	*Tasks Comp*	0.980	0.949	No
5	*Tasks Comp*	*Negotiating*	***0.001***	***0.006***	**Yes**
6	*Tasks Comp*	*Monitoring*	***0.002***	***0.016***	**Yes**

(e) *row* = *Initial*

1	*Discovering*	*Initial*	***0***	***0***	**Yes**
2	*Initial*	*Discovering*	0.970	0.988	No

In the two multiple transition s-t cuts in Figure 5, which were identified by node contraction, but could not be found by the perturbation algorithm of Section 3.

The perturbation algorithm was also applied to raise the self-transition probability of the 5 non-absorbing states in the grid model to 1. This perturbation caused the proportion of *Tasks Completed* to decline to 0 when applied to 4 of these states: *Initial*, *Discovering*, *Negotiating*, and *Monitoring* states. All 4 are trap states found through node contraction. The fifth state *Waiting*, is not a trap state; but is part of a state transition that is a member of both multiple-transition s-t cuts in Figure 5. Hence, if the self-transition probability of *Waiting* is raised toward 1, there is only a partial reduction in proportion of tasks completed.

Executing the exhaustive perturbation algorithm on non-absorbing rows of the grid model took 56 minutes in the 8-hour case and 4.5 hours in the 640-hour case. In comparison, node contraction needed less than 0.01 s to find all minimal s-t cut sets and trap states. In the 8-hour case, generating Markov simulation curves to reduce the proportion of *Tasks Completed* to 0 for all minimal s-t cut sets and trap states required 244 s, or 7 % of the 56 minutes needed by the perturbation algorithm. For the 640-hour case, these computations took 230 s, or 1.4 % of the 4.5 hours needed by the perturbation algorithm. Thus, minimal s-t cut set analysis needed two orders of magnitude less time than exhaustive application of the perturbation algorithm. All experiments were executed on a workstation with dual quad-core, 3.16GHz processors and 32 GB memory.

5.3. Using Node Contraction to Find Minimal s-t Cut Sets in Large Problems

This section reports the results of experiments on the use of the node contraction algorithm to find critical transitions in large, complex Markov chain models with many multiple transition minimal s-t cuts. These experiments compare the results of using the contraction algorithm to the results produced by the enumeration algorithm of (Provan and Shier 1996) which is guaranteed to find all minimal s-t cut sets and the critical transitions in these cut sets. Here, the criticality of transitions is estimated using measures we define for these experiments. The results show that, with some exceptions, the node contraction algorithm found a large proportion of the most critical cut sets in two orders of magnitude less time than did exhaustive enumeration. While further experiments are needed, these preliminary investigations suggest that minimal s-t cut set analysis can effectively identify critical transitions in large, complex Markov chain graphs as might be encountered in real-world problems.

5.3.1. Experimental Design

To perform these experiments, four Markov chain models were selected, each consisting of 40 or 50 states, from (Boyarsky 1988; Stewart 2004; Jensen and Jessup 1986) and single time-period TPMs were generated using (Hunt 1994). All four problems were originally ergodic chains, which were suitably modified to be absorbing chains. Though the matrices were sparse, these problems were large and complex, with a very sizable number of minimal s-t cut sets between the *Initial* and absorbing states ($> 4\times10^8$ for the largest; see Table 2.) In contrast to the grid system model, minimal s-t cut sets for these problems consisted of multiple state transitions, which could correspond to combinations of circumstances that impact system performance. In (Dabrowski, Hunt and Morrison 2010), the full description of all four Markov chain problems is provided, which we omit here due to lack of space.

To provide a baseline measure for the number of minimal s-t cut sets in these Markov chain graphs, we implemented the minimal s-t cut set enumeration algorithm of (Provan and Shier 1996), which lists all cut sets. To determine which minimal s-t cut sets were most critical, we selected ranking criteria based on the idea that the most critical cut sets will have a small number of state transitions. We chose this basis, because fewer transitions represent smaller combinations of circumstances that are more likely to occur and thus more likely to impact a system. (Note: this intuition is supported in the case of undirected graphs by the finding (Karger 2001) that small cut sets are more likely to disconnect undirected graphs, if edges of the graph that fail independently with a known probability. Also in (Dabrowski and Hunt 2011), we use this ranking criterion to analyze a DTMC for a cloud computing system.) We used this basis to choose 3 ranking criteria. The first criterion, Sort A, ranks minimal s-t cut sets by the fewest number of edges as the 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 few transitions higher). Hence, Sorts A and B are likely to identify minimal s-t cut sets in which small perturbations to the fewest number of state transitions are likely to produce the largest changes. The third ranking criterion, Sort C, uses the least number of edges as a primary sorting criterion and the highest total transition probability of edges as a secondary criterion. Sort C identifies cut sets consisting of state transitions that are more likely to be taken and therefore, if perturbed, more likely to affect system behavior.

5.3.2. Experimental Results

We applied the node contraction algorithm and the enumeration algorithm of (Provan and Shier 1996) to the four TPMs, ranked the minimal cut sets produced by each using the ranking criteria described above, and compared the results to determine the proportion of most highly ranked cut sets that the node contraction algorithm could find. With the exception of Matrix 1, Table 2 shows that, with 100,000 repetitions, node contraction generated 91.4 % of the top 100 ranked cut sets that were generated by the enumeration algorithm for all four TPMs under all three sorting criteria. The contraction algorithm produced these results in 1.3 % of

the time needed by the enumeration algorithm. This amounts to a two-order of magnitude improvement in time. 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. For Matrix 4, the node contraction algorithm could find all the top 100 under sort criteria B and C in about 15 minutes (as opposed to 156.1 hours through enumeration). Here, node contraction was successful despite the fact that Matrix 4 has over 4×10^8 minimal s-t cut sets.

Table 2: Comparison of minimal s-t cut sets generated by the enumeration algorithm of (Provan and Shier 1996) and by the node contraction algorithm. At 10,000 repetitions, 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) of the cut sets found by enumeration in 1.3 % of the time.

		Minimal s-t cut set enumeration		**Proportion (in %) of 100 top-ranked minimal s-t cut sets ranked by criteria A, B that were found by the node contraction algorithm**							
Number	Order	Number of cut sets	Time (in hours)	After 10,000 repetitions				After 100,000 repetitions			
				Time	Sort A	Sort B	Sort C	Time	Sort A	Sort B	Sort C
1	50	530,432	332 s	640 s	80	100	96	---	---	---	---
2	50	28,230,288	21.6	171 s	93	98	65	1710 s	99	100	99
3	50	27,242,634	36.0	218 s	67	100	100	2288 s	88	100	100
4	40	422,060,801	193.6	106 s	30	80	62	1051 s	37	100	100

However, in Matrix 4, the algorithm found only 37 of 100 high ranked minimal s-t cut sets under Sort A. Also, for Matrix 1, Table 2 shows that the node contraction algorithm had to run longer than the enumeration algorithm, before it began to produce a large percentage of highly-ranked cut sets. This difference in performance may be attributable in part to topological characteristics such as vertices (states) with large numbers of edges (state transitions), which increases vertex interconnectivity and impedes the contraction process. This exception suggests that in some cases where TPMs are small, it may be more efficient to enumerate cut sets than to generate them probabilistically. Despite these exceptions, the data shows that the node contraction algorithm can be used to find a high proportion of minimal s-t cut sets representing combinations of critical state transitions in larger Markov chains within reasonable time limits.

6. CONCLUSIONS AND FUTURE WORK

This paper has described an approach for using minimal s-t cut set analysis to guide perturbation of a time-inhomogeneous DTMC in order to understand the potential for failure in grid computing systems. The approach combines multiple techniques in a way not previously reported. In this approach, minimal s-t cut sets are computed for paths from the *Initial* to selected absorbing states in the directed graph of a DTMC. These cut sets can be used to identify critical state transitions, which if perturbed, reveal areas for potential performance degradation. The perturbation of critical state transitions in turn provides a basis to identify potential failure scenarios that could occur in the target system being modeled. By perturbing critical state transitions incrementally, it is possible to quantitatively measure performance degradation and to predict how the target system being modeled is likely to respond to increased incidence of failure. As we have shown, the stochastic character of the Markov chain representation of the system state enables modeling of systems having large numbers of tasks, while time inhomogeneity allows modeling of system evolution over time. Using a large-scale grid system as a proxy for a real-world system, we used the approach described here to identify failure scenarios in systems that process hundreds of tasks over different durations. Our results showed that minimal cut set analysis could be used to identify (in two-orders of magnitude less time) all of the failure scenarios found using exhaustive search techniques. In addition, this method also discovered failure scenarios that involved multiple state transitions, which the exhaustive search algorithm could not. To find critical state transitions in larger Markov chains, the paper has presented a probabilistic algorithm for minimal s-t cut set analysis. Experimental results in Section 5.3.2 show the potential of this algorithm for efficiently analyzing large, complex problems and finding related critical transitions that represent complicated circumstances.

To further evaluate the utility of minimal s-t cut set analysis, it will be necessary to carry out experiments in which DTMC representations are constructed for different problem domains, such as reported in (Dabrowski and Hunt 2011). The use of this approach will have to be further evaluated on larger problems as we have begun to do in this paper. As part of this effort, it will also be necessary to investigate other methods for finding minimal s-t cut sets in large, complex directed graphs. For instance, there are alternative approaches to node contraction, such as (Curet, DeVinney and Gaston 2000) which could be examined. Another possible method involves use of maximum-flow algorithms (Ford and Fulkerson 1962; Goldberg and Tarjan 1988) to find s-t cut sets that identify critical transitions. These algorithms find s-t cut sets on the basis of maximum flow and minimum capacity. Potentially, maximum-flow algorithms could be adapted to find cut sets and rank them on the basis of their nearness to maximum flow and minimum capacity, rather than the criteria described here. To enable such rankings, the work of (Curet, DeVinney and Gaston 2000; Balcioglu and Wood 2003) could be used. In addition, we are also investigating the use of methods that are not based on graph theory concepts to analyze dynamic behavior in complex systems. In (Hunt, Morrison and Dabrowski 2011), we describe the use of spectral methods for eigendecompositon to identify critical state transitions, and in (Dabrowski, Hunt and Morrison 2010) we employ this technique as a complementary method to

minimal s-t cut set analysis. Beyond this, it is our hope that this paper will provide useful ideas to other researchers studying dynamic behavior in complex systems, and that ultimately, the work will lead to the development of effective tools for this purpose.

REFERENCES

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

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

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

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

Curet, N., DeVinney, J. and Gaston, M., 2000. An Efficient Network Flow Code for Finding all Minimum Cost s-t Cutsets. *Computers and Operations Research*, 29, 205–219.

Dabrowski, C. and Hunt, F., 2009. Using Markov Chain Analysis to Study Dynamic Behavior in Large-Scale Grid Systems. *Proceedings of the Seventh Australasian Symposium on Grid Computing and e-Research–Volume 99*, pp. 29–40. January 21, Wellington (New Zealand).

Dabrowski, C., Hunt, F. and Morrison, K., 2010. *Improving the Efficiency of Markov Chain Analysis of Complex Distributed Systems.* National Institute of Standards and Technology, Interagency Report 7744.

Dabrowski, C. and Hunt, F., 2011. Identifying Failure Scenarios in Complex Systems by Perturbing Markov Chain Models. *Proceedings of the 2011 Pressure Vessels and Piping Division Conference*. July 17–21, Baltimore (Maryland, USA). In press.

Ford, L. and Fulkerson, D., 1962. *Flows in Networks*. Princeton: Princeton University Press.

Gambin, A., Kryzanowski, P. and Pokarowski, P., 2008. Aggregation Algorithms for Perturbed Markov Chains with Applications to Network Modeling. *SIAM Journal of Scientific Computation*, 31 (1), 45–77.

Goldberg, A. and Tarjan, R., 1988. A New Approach to the Maximum-Flow Problem. *Journal of the ACM*, 35 (4), 921–940.

Hassin, R. and Haviv, M., 1992. Mean Passage Times and Nearly Uncoupled Markov Chains. *SIAM Journal of Discrete Mathematics*, 5 (3), 386–397.

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

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

Hunt, F., 1994. A Monte Carlo Approach To The Approximation of Invariant Measures. *Random and Computational Dynamics*, 2 (1), 111–112.

Hunt, F., Morrison, K. and Dabrowski, C., 2011. Spectral Based Methods That Streamline the Search for Failure Scenarios in Large-Scale Distributed Systems. *Nineteenth IASTAD International Conference on Modeling and Simulation*. June 22–24, Crete (Greece). In press.

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

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

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

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

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

Provan, J. and Shier, D., 1996. A Paradigm for Listing (s,t)-cuts in Graphs. *Algorithmica*, 15, 351–372.

Rosenberg, D., Solan, E. and Vielle, N., 2004. Approximating a Sequence of Observations by a Simple Process. *The Annals of Statistics*, 32 (6), 2742–2775.

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

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

Stewart, G., 2004. *MVMRWK: Markov Chain Transition Probability Matrix*. National Institute of Standards and Technology. Available from: http://math.nist.gov/MatrixMarket/data/NEP/mvmrwk/rw136.html. [Accessed 27 June 2011]

Stewart, W. and Dekker, M., 1994. *Numerical Solution of Markov Chains*. Princeton: Princeton University Press.

Suri. R., 1989. 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.

Tang, Z. and Dugan, J., 2004. Minimal cut set/sequence generation for dynamic fault trees. *Proceedings of the 2004 Annual Symposium on Reliability and Maintainability***,** pp. 207–213. January 26–29, Los Alamitos (California USA).

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