On a Random Walk Survivability Problem With Arc Failures and Memory

Burak Büke[1] J. Cole Smith[2]
Sadie Thomas[3]

[1]School of Mathematics, University of Edinburgh
B.Buke@ed.ac.uk
[2]Department of Industrial Engineering, Clemson University
jcsmith@clemson.edu
[3]Department of Industrial and Systems Engineering, University of Florida
sadiethomas@ufl.edu

Abstract

Consider a directed network in which each arc can fail with some specified probability. An entity arrives on this network at a designated origin node and traverses the network in a random-walk fashion until it either terminates at a destination node, or until an arc fails while being traversed. We study the problem of assessing the probability that the random walk reaches the destination node, which we call the survival probability of the network. Complicating our analysis is the assumption that certain arcs have "memory," in the sense that after a memory arc is successfully traversed, it cannot fail on any subsequent traversal during the walk. We prove that this problem is #P-hard, provide methods for obtaining lower and upper bounds on the survival probability, and demonstrate the effectiveness of our bounding methods on randomly generated networks.

Keywords: Random walks on graphs, survival probability, computational complexity, network reliability, #P-hard, Markov chain, heuristics

1 Introduction

In this paper, we consider a network $G(V, A)$ having node set $V = \{1, \ldots, n\}$ and directed arc set A, where a reliability value $0 \leq r_{ij} \leq 1$ is associated with each arc $(i, j) \in A$. We examine some entity that conducts a random walk starting at node 1 (the origin node) and seeks to reach node n (the destination node). Define $\Delta_i = \{j \in V : (i, j) \in A\}$, and $\delta_i = |\Delta_i|$. Given that the entity is currently on node $i \in V$, the next arc traversed by the random walk is determined by selecting arc (i, j), where $j \in \Delta_i$, with probability $1/\delta_i$. We assume that there exists a directed path from node i to node n, for every $i \in V$. This assumption is made without loss of generality: If no path exists from some $i \in V$ to n, then we can equivalently add arc (i, n) with $r_{in} = 0$ to satisfy the assumption.

The *survival probability* of a network is the probability that a random walk hits node n before it encounters an arc that fails while being traversed. In a truly Markovian model, the probability that an arc fails is independent of the walk's prior movements, and in fact a simple conditioning argument can be employed to compute the survival probability (as we show in Section 3). However, in several settings for which random walk analyses are conducted, an arc that is successfully traversed once is known to be reliable for the duration of the walk. This interpretation arises, generally speaking, when the existence of arc (i, j) is uncertain, and r_{ij} refers to the probability that (i, j) exists. A walk that encounters arc (i, j) and successfully crosses it would confirm that the arc exists, and would guarantee its reliability in future traversals.

When an arc is known to be reliable after being successfully traversed once, we say that the arc has memory, or that it is a *memory arc*. A network may consist of a mixture of memory arcs and *memoryless arcs* (e.g., those arcs (i, j) that survive with probability r_{ij} each time they are traversed, independent of the number of times they have already been traversed). Note that any arc of the form (i, n) is automatically treated as a memoryless arc for simplicity, because the walk stops upon hitting node n and therefore cannot traverse any such arc multiple times. We let $A^m \subset A$ denote the set of memory arcs in our problem. The problem of calculating the survival probability is called the Random Walk Survivability problem with Memory (RWSM). Because of the arc memory property, simple Markov-chain methods are unsuitable for addressing the RWSM problem. In fact, there is no obvious way to compute a network's survival probability short of using an exponential-time algorithm. In this paper, we make three primary contributions. The first demonstrates that RWSM is in general #P-hard, and in particular, is at least as hard as enumerating all Hamiltonian paths in a directed network. The second provides two general approaches based on RWSM modifications, which (respectively) yield lower and upper bounds on the survival probability of a network. The third prescribes a mechanism for strategically constructing these modifications in order to produce tight bounds on survival probabilities, which we empirically demonstrate on a set of randomly generated test instances.

The graph theoretic literature is replete with applications in which random walks arise, or in which they approximate behavior on real networks. One classical application of random walks appears in the context of electricity grids [9], while more contemporary applications regard social and recommender networks [5, 17]. We refer the reader to [3, 4, 15] for a comprehensive treatment of random walks. To motivate this paper and illustrate the problem setting, consider the stochastic surveillance of a network. An intruder enters the network via some node, and without knowledge of the network structure, moves randomly in order to gain access to some target nodes. (The notion of a single possible origin and destination node for the random walk is made without loss of generality above, and can easily be extended to the case having multiple possible origin or destination nodes.) Each movement made by the intruder risks detection by the network owner. In one scenario, the network owner will detect (with perfect reliability) any arc that is being monitored. The intruder is unsure of which arcs are being monitored, and assesses the probability that arc (i, j) is being monitored as $1 - r_{ij}$. Hence, in this case, traversing (i, j) the first time is successful with probability r_{ij}; moreover, if the arc is traversed once, it will be safely traversed each time thereafter, and hence each arc is treated as a memory arc. In a different scenario, all arcs are monitored, but detection on arc (i, j) occurs with probability $(1 - r_{ij})$. In this case, detection on (i, j) is simply a (memoryless) Bernoulli process, and (i, j) is treated as a memoryless arc with reliability r_{ij}. (Of course, the situation in which arc monitoring and detection are both probabilities that are less than one is also of interest. This scenario does not satisfy the assumptions that we make in our model, although our model could be readily adapted to handle such a scenario.)

One notable application of random walks is the PageRank algorithm [6, 8], which is used to determine relative importances of web pages. This algorithm models the behavior of Internet users as a random walk on the network, where web pages are treated as nodes and the links between pages as directed arcs. Similarly, RWSM can be used to calculate the probability that a user safely reaches desired information without encountering undesirable content. This content could be malware in a security context, or could be a competitor's advertisement in a sales context. When a user clicks on a link to go from page i to j for the first time, the user encounters undesirable content with probability $1 - r_{ij}$. (Note that when the probability depends only on j, a simple transformation exists to convert the RWSM problem with imperfect arcs to one having imperfect nodes.) Once a link is safely opened, then it will be safely opened each time thereafter, which justifies the assumption of arcs having memory.

Although the problem we explore in this paper is new, several studies in the literature examine the probability that a network functions as desired. One particular class of problems explores a multicommodity flow network with unreliable arcs. The challenge in these problems is to evaluate the probability that there exists a path connecting each origin-destination pair after arcs fail. It is in general NP-hard to evaluate network reliability as defined in this manner [7], and other foundational reliability problems are also known to be #P-hard [22], as we prove the RWSM to be in this paper. We refer to [12, 13, 14, 18, 23] for studies that propose algorithms to compute network reliability. One related study of interest was performed by Shier [21], who approximates a different network reliability problem by examining paths of bounded length. As such, the approach in [21] relaxes the probability that certain arcs might fail in an origin-destination walk. By contrast, our approaches in this paper differ by constructing alternative restrictions and relaxations on survival probability, and are tailored toward mitigating the complexity induced by the presence of memory arcs.

The remainder of this paper is organized as follows. Section 2 states our main complexity result, showing that RWSM is #P-hard. Section 3 explores lower- and upper-bounding algorithms for RWSM. Then, we conduct computational experiments in Section 4 that show the tightness of the suggested bounding methods and illustrate the effectiveness of our recommended strategies to improve these bounds. Finally, we conclude the paper in Section 5 and discuss possible directions for future research.

2 Problem Complexity

We begin by stating our key complexity result: The RWSM is a #P-hard problem. The implication of this result is that if a polynomial-time algorithm existed to compute the survival probability for an RWSM instance, then it would be possible to solve very difficult counting problems. In particular, the following proof demonstrates that an efficient solution procedure to the RWSM problem would imply an efficient algorithm for determining the number of Hamiltonian paths that exist in a directed graph.

Theorem 1. *The RWSM problem is #P-hard.*

Proof. We demonstrate that the RWSM problem is at least as difficult as computing the number of Hamiltonian paths (HPs) in a directed graph with a specified origin and destination. The HP problem seeks a directed path between two designated nodes such that all nodes in the network are visited exactly once by the path, and is known to be strongly NP-complete [11]. Our transformation is from an arbitrary HP instance having nodes $V' = \{1, \ldots, \overline{n}\}$ and arc set A', with origin node 1 and destination node $\overline{n}$. Note that we can assume without loss of generality that no arcs exist of the form $(\overline{n}, i)$, for any $i = 1, \dots, \overline{n} - 1$.

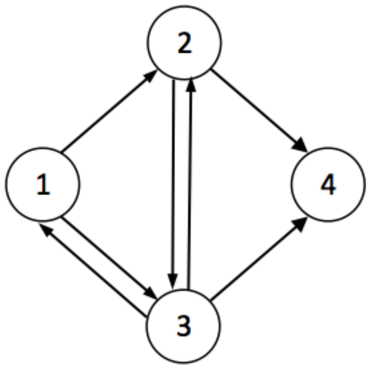
The RWSM network is initialized exactly as a copy of the HP instance network, with several modifications described as follows. For each HP node $i = 1, \dots, \overline{n}$, split node i into two nodes, i-*in* and i-*out*, and create a *split arc* directed from i-*in* to i-*out*. Every arc $(i, j) \in A'$ is replaced with a *temporary* arc directed from i-*out* to j-*in*; each temporary arc will later be replaced with a specialized subgraph. All split arcs have a common reliability value of r (which we will specify at the end of this proof), while the reliability of all other arcs in the network will equal 1. Node 1-*in* is the source, and node $\overline{n}$-*out* is the destination for RWSM.

The next part of our transformation ensures that, given a starting point of i-*out* (for $i < \overline{n}$), the probability of traversing to j-*in* is equal to a common value $1/d$, for all $(i, j) \in A'$. To achieve this, for each node $i = 1, \dots, \overline{n} - 1$, create $(\overline{n} - \delta_i)$ new temporary arcs from i-*out* to i-*in*. (For now, this operation creates parallel arcs in the graph whenever $\overline{n} - \delta_i \geq 2$. However, our transformation will ultimately ensure that no parallel arcs exist.) Now, every i-*out* node has out-degree $\overline{n}$, and each arc exiting an i-*out* node is a temporary arc. Figure 1 illustrates the transformation described thus far, with Figure 1(a) depicting the input instance, and Figure 1(b) depicting the network obtained after duplicating each node and adding the required split and temporary arcs.

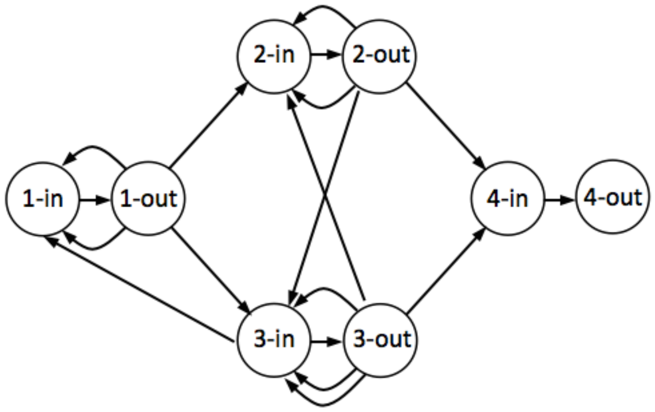
Next, we replace each temporary arc (i-*out*, j-*in*) where $(i, j) \in A'$ or $i = j$, with a *reducing subgraph* structure. A reducing subgraph structure is seeded by a chain of $k + 1$ arcs (i-*out*, ij_1), (ij_1, ij_2), $\dots$, (ij_k, j-*in*), where $ij_1, \dots, ij_k$ are k new nodes added to the network. (Note that a precise value of k will be specified at the end of this proof.) For each new node ij_κ, add an arc from ij_κ to node $\overline{n}$-*out*. Because each new arc added is not a split arc, it has 100% reliability. Given that the walk is currently visiting i-*out*, the probability that the walk chooses arc (i-*out*, ij_1) next (for some $j \in \Delta_i$) and reaches j-*in* (instead of going directly to $\overline{n}$-*out*) is $1/(\overline{n}2^k)$. Accordingly, we select $d = \overline{n}2^k$ in this transformation. Figure 1(c) illustrates an example of a reducing subgraph.

It is more convenient to compute the *failure probability* (given by 1 $-$ probability of successfully reaching node $\overline{n}$-*out*). We say that the walk takes a "step" each time it attempts to traverse a split arc. Define u_{sw} as the number of walks that (a) have s steps, (b) have visited w distinct i-*out* nodes, and (c) terminate at an i-*out* node that is visited exactly once in the walk (i.e., the walk is visiting an i-*out* node for the first time when it terminates). Note that this value regards the counting of all such walks without regard to arc reliabilities.

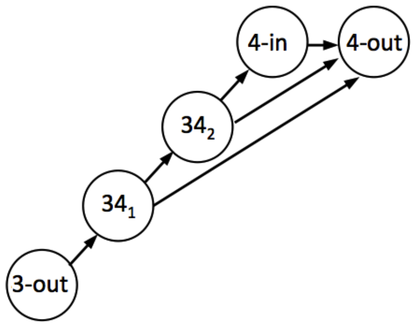
Now, defining f_s as the probability that the walk fails during step number s, the failure



(a) Original network



(b) Transformed network



(c) Reducing subgraph with two new nodes to replace arc (3-out, 4-in)

Figure 1: Sample graph transformation before reducing subgraphs.

probability is then given as

$$F = \sum_{s=1}^{\infty} f_s. \tag{1}$$

We can compute f_s as:

$$f_s = \sum_{w=1}^{\min\{s,\overline{n}\}} \frac{(u_{sw}(1-r)r^{w-1})}{d^{s-1}} = \frac{1-r}{d^{s-1}} \left(\sum_{w=1}^{\min\{s,\overline{n}\}} u_{sw} r^{w-1} \right). \tag{2}$$

Note that the $1/(d^{s-1})$ term captures the probability of taking a given s-step walk that ends at an *i-out* node. The $(1-r)r^{w-1}$ component term accounts for the probability of successfully making it across $(w-1)$ split arcs that are visited for the first time, but then failing on the last split arc visited. The u_{sw} term counts how many such unique walks exist. In particular, we are most interested in $u_{\overline{n}\overline{n}}$, which equals the number of $\overline{n}$-step walks that visit $\overline{n}$ distinct out-nodes; i.e., the number of Hamiltonian paths. Our goal is to select k and r appropriately so that the coefficient $u_{\overline{n}\overline{n}}$ can efficiently be recovered from the failure probability F.

To accomplish this goal, define $\alpha_{sw} = (u_{sw} r^{w-1})/d^{s-1}$, and consider the sequence α_{11}, α_{21}, α_{22}, α_{31}, ... We wish to choose k (which in turn determines d) and r such that:

$$r^{w-1}/d^{s-1} > \sum_{w'=w+1}^{s} \alpha_{sw'} + \sum_{s'=s+1}^{\infty} \sum_{w'=1}^{\min\{s',\overline{n}\}} \alpha_{s'w'} \quad \forall 1 \leq w \leq s \leq \overline{n}. \tag{3}$$

If (3) holds, then suppose that F is known, and consider the calculation of $F/(1-r)$, given by:

$$F/(1-r) = \sum_{s=1}^{\infty} \sum_{w=1}^{\min\{s,\overline{n}\}} \alpha_{sw}. \tag{4}$$

Then we can compute u_{sw} efficiently, $\forall 1 \leq w \leq s \leq \overline{n}$, by the following procedure. Initialize by setting $s = w = 1$ and $T_{1,1} = F/(1-r)$. Compute

$$u_{sw} = \left\lfloor \frac{T_{s,w}}{r^{w-1}/d^{s-1}} \right\rfloor, \tag{5}$$

and then set $\alpha_{sw} = (u_{sw} r^{w-1})/d^{s-1})$ and

$$T_{s,w} = \begin{cases} T_{s-1,s-1} - \alpha_{s-1,w-1}, & \text{if } s > 1 \text{ and } w = 1, \\ T_{s,w-1} - \alpha_{s,w-1}, & \text{if } 1 < w \leq s. \end{cases} \tag{6}$$

Note that (5) is valid due to the facts that (a) u_{sw} is an integer, (b) u_{sw} cannot be an integer greater than the value specified by (5), or else the remaining α-values in the summation specified in (4) must be negative, and (c) if u_{sw} were less than the value specified by (5), then the remaining sum of α-values would need to exceed r^{w-1}/d^{s-1}, which by (3) is impossible. Equation (6) updates $T_{s,w}$ after computing u_{sw}. We would then proceed by incrementing w if $w < s$, and otherwise (if $w = s$), by setting $s = s + 1$ and $w = 1$. We then compute u_{sw} via (5), and carry this process out until $u_{\overline{n}\overline{n}}$ is obtained. Therefore, given r and k such that (3) holds true, $u_{\overline{n}\overline{n}}$ (the number of Hamiltonian paths) can be efficiently computed, and thus the RWSM problem is #P-hard.

To find values for r and k that satisfy (3), we seek to find upper bounds on both terms appearing in the right-hand side of (3), and then guarantee that r^{w-1}/d^{s-1} exceeds the upper bounds on those terms, for $1 \leq w \leq s \leq n$. First note that $\sum_{w'=w+1}^{s} \alpha_{sw'}$ is the probability of taking an s-step walk with $w' > w$ unique steps. An upper bound on $\sum_{w'=w+1}^{s} \alpha_{sw'}$ is the probability of the walk not failing on the first w unique steps, multiplied by the probability of the walk making it across each of $s-1$ reducing subgraphs. The first of these values is simply given by r^w. The second of these values is $1/2^{k(s-1)}$, noting that each temporary arc is replaced with a reducing subgraph having k intermediate nodes, such that the walk's probability of going from *i-out* to *j-in* (and not to $\overline{n}$-*out*) on a reducing subgraph is $1/2^k$. Therefore, we have:

$$\sum_{w'=w+1}^{s} \alpha_{sw'} \leq r^w/2^{k(s-1)}. \tag{7}$$

The same logic gives $\sum_{s'=s+1}^{\infty}(\sum_{w'=1}^{\min\{s',\overline{n}\}} \alpha_{s'w'})$ is bounded from above by the probability of the walk successfully traversing s reducing subgraphs, so that:

$$\sum_{s'=s+1}^{\infty} \sum_{w'=1}^{\min\{s',\overline{n}\}} \alpha_{s'w'} \leq 1/2^{ks}. \tag{8}$$

Therefore, to satisfy (3), it is sufficient to satisfy the following, $\forall 1 \leq w \leq s \leq \overline{n}$:

$$r^{w-1}/d^{s-1} > r^w/2^{k(s-1)} + 1/2^{ks}.$$

By substituting $d = \overline{n}2^k$, this inequality becomes

$$r^{w-1}/(\overline{n}^{s-1}2^{k(s-1)}) > r^w/2^{k(s-1)} + 1/2^{ks} \quad \Leftrightarrow \tag{9}$$

$$r^{w-1}/\overline{n}^{s-1} > r^w + 1/2^k, \tag{10}$$

To achieve this, we find r and k such that (a) $r^{w-1}/(2\overline{n}^{s-1}) \geq r^w$ and (b) $r^{w-1}/(2\overline{n}^{s-1}) \geq 1/2^k$. Hence, noting that (a) and (b) can be satisfied for all s and w by considering the case of $s = w = \overline{n}$, we take

$$r = 1/(2\overline{n}^{\overline{n}-1}), \tag{11}$$

and choose the smallest integer k that satisfies the following, where $\beta = \log_2 \overline{n}$:

$$2^k > 2\overline{n}^{\overline{n}-1}/r^{\overline{n}-1} = 2(2^{\beta(\overline{n}-1)})(2(2^{\beta(\overline{n}-1)}))^{(\overline{n}-1)} = 2^{\overline{n}(\beta(\overline{n}-1)+1)}, \text{ i.e.,}$$
$$k = \lceil \overline{n}(\beta(\overline{n}-1)+1) \rceil + 1. \tag{12}$$

Note therefore that k is $O(\overline{n}^2 \log \overline{n})$.

Finally, we verify that the transformation is polynomial in terms of $\overline{n}$, the size of the original HP instance. To do this, we verify that the number of the transformed RWSM network's nodes is bounded by a polynomial function of $\overline{n}$ (which suffices because there are no parallel arcs in this network, and the number of arcs is no more than a square of the number of nodes), and that the numerical data requires a polynomial number of bits to be represented using a binary encoding. Observe that there are $2\overline{n} + k(\overline{n}-1)\overline{n}$ nodes in the transformed network after splitting all original nodes, adding additional temporary arcs to make the out-degree of each node equal to some common value n, and replacing each temporary arc with a k-node reducing subgraph. Noting that k is $O(\overline{n}^2 \log \overline{n})$, the number of RWSM nodes is $O(\overline{n}^4 \log \overline{n})$. Next, the only numerical data used is the reliability value r for the split arcs, which takes $O(\overline{n} \log \overline{n})$ bits to represent, noting the base-2 log of $1/r$ as computed by (11). Hence, the transformation is polynomial, and this completes the proof. □

3 Bounding Techniques

Computing the survival probability for a random walk having memory arcs is evidently very difficult. We thus develop in this section efficient methods to compute lower and upper bounds on the survival probability. Section 3.1 focuses on lower-bounding methods, where we convert a subset of memory arcs to memoryless arcs. In Section 3.2, we describe an upper-bounding scheme, wherein memory arcs are partitioned into clusters, and the traversal of one arc in a cluster ensures the reliability of all arcs in the cluster. Each modification serves to simplify the calculation of the survival probability, at the expense of underestimating (in Section 3.1) or overestimating (in Section 3.2) this probability. Section 3.3 summarizes our developments and discusses an implementation technique that we employ in our methods.

3.1 Lower-Bounding Technique

In this section, we first demonstrate how to obtain a lower bound on the survival probability by assuming that some (or all) of the memory arcs are actually memoryless. Then, we present a method for calculating the survival probability by using an expanded network having only memoryless arcs. The size of this expanded network grows exponentially as a function of the number of memory arcs, and therefore it is necessary to limit the number of arcs that have memory. One strategy would be to treat as memoryless any arc that could be traversed at most once due to graph connectivity, because the notion of memory on those arcs is not relevant. (In fact, acyclic graphs prohibit re-traversals of any arc, implying that memory can be ignored on all arcs, and hence that the network's survival probability can be determined in polynomial time.) In general, however, we require a more sophisticated method to most effectively choose which arcs are treated as memoryless.

First we prove the intuitive result that if a subset of memory arcs is changed to memoryless, the survival probability of this modified problem cannot increase. Suppose that a random walk is performed on a directed graph $G(V, A)$ having memory arcs $A^m \subseteq A$. Consider a walk w given by the sequence of nodes $i_1 - \cdots - i_z$, where $(i_j, i_{j+1}) \in A$ for $j = 1, \dots, z-1$, and where nodes may be repeated in the walk. Now define $N^{w,i}$ and $N^{w,(i,j)}$ as the number of times node i and arc (i, j) are visited during walk w, respectively, and let W be the set of walks starting at node 1 and terminating at node n. The probability that walk $w \in W$ is performed and successfully terminates at node n is given by

$$\left(\prod_{i \in V \setminus \{n\}} \frac{1}{{\delta_i}^{N^{w,i}}}\right) \left(\prod_{(i,j) \in A^m : N^{w,(i,j)} > 0} r_{ij}\right) \left(\prod_{(i,j) \in A \setminus A^m} {r_{ij}}^{N^{w,(i,j)}}\right), \tag{13}$$

where the first term corresponds to the probability of choosing walk w and the remainder corresponds to the survival probability of walk w. Summing over all possible walks in W yields the survival probability p_{A^m}:

$$\sum_{w \in W} \left(\prod_{i \in V \setminus \{n\}} \frac{1}{{\delta_i}^{N^{w,i}}}\right) \left(\prod_{(i,j) \in A^m : N^{w,(i,j)} > 0} r_{ij}\right) \left(\prod_{(i,j) \in A \setminus A^m} {r_{ij}}^{N^{w,(i,j)}}\right). \tag{14}$$

We are now ready to state our lemma.

Lemma 1. *Consider a directed graph $G(V, A)$ along with two alternative memory arc sets $A_1^m \subset A_2^m \subset A$. Then $p_{A_1^m} \leq p_{A_2^m}$.*

Proof. For the first instance (with memory arcs A_1^m) and any walk $w \in W$, we can write the probability of choosing walk w and successfully reaching node n as

$$\begin{aligned}&\left(\textstyle\prod_{i \in V \setminus \{n\}} \frac{1}{{\delta_i}^{N^{w,i}}}\right) \left(\textstyle\prod_{(i,j) \in A_2^m : N^{w,(i,j)} > 0} r_{ij}\right) \left(\textstyle\prod_{(i,j) \in A \setminus A_2^m} {r_{ij}}^{N^{w,(i,j)}}\right) \\ &\qquad\qquad \left(\textstyle\prod_{(i,j) \in A_2^m \setminus A_1^m : N^{w,(i,j)} > 0} {r_{ij}}^{N^{w,(i,j)}-1}\right) \\ &\leq \left(\textstyle\prod_{i \in V \setminus \{n\}} \frac{1}{{\delta_i}^{N^{w,i}}}\right) \left(\textstyle\prod_{(i,j) \in A_2^m : N^{w,(i,j)} > 0} r_{ij}\right) \left(\textstyle\prod_{(i,j) \in A \setminus A_2^m} {r_{ij}}^{N^{w,(i,j)}}\right).\end{aligned}$$

Summing over all $w \in W$ on both sides, we get $p_{A_1^m} \leq p_{A_2^m}$. □

Lemma 1 indicates that the survival probability for the case in which all arcs lack memory acts as a lower bound. To compute the survival probability in the memoryless case, let Y_t represent the position of the random walk without memory at time t if the walk has not yet failed; otherwise, Y_t represents a cemetery state, Θ, if the walk has failed by time t. Define

$$\phi(i) = \mathbb{P}(\exists t \geq 0 : Y_t = n | Y_0 = i).$$

By conditioning on our first step, for $i = 1, \dots, n-1$ we get

$$\begin{aligned} \phi(i) &= \textstyle\sum_{j:(i,j)\in A} \frac{r_{ij}}{\delta_i} \mathbb{P}(\exists t \geq 0 : Y_t = n | Y_0 = i, Y_1 = j) \\ &= \textstyle\sum_{j:(i,j)\in A} \frac{r_{ij}}{\delta_i} \mathbb{P}(\exists t \geq 0 : Y_t = n | Y_0 = j) \\ &= \textstyle\sum_{j:(i,j)\in A} \frac{r_{ij}}{\delta_i} \phi(j). \end{aligned}$$

The first equality is the result of conditioning on the first time step, the second equality follows by using the Markov property and time-homogeneity, and the third holds by substitution. Notice that we have $\phi(n) = 1$ by definition. Hence, the above result yields $n-1$ equations with $n-1$ variables.

To write the above equations in matrix form, define the $(n-1) \times (n-1)$ matrix Q with entries q_{ij} such that

$$q_{ij} = \begin{cases} \frac{r_{ij}}{\delta_i} & \text{if } (i,j) \in A, i \neq n, j \neq n \\ 0 & \text{otherwise} \end{cases} \qquad \forall i, j \in V \setminus \{n\}. \tag{15}$$

Similarly, define vector b as

$$b_i = \begin{cases} \frac{r_{in}}{\delta_i} & \text{if } (i,n) \in A \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in V \setminus \{n\}. \tag{16}$$

Then the solution of the above equations can be written as

$$\phi = (I - Q)^{-1} b,$$

where I is the $(n-1)$-dimensional identity matrix. Note that the Markov chain is absorbing due to the assumption that there exists a path from every node to node n. Therefore, $(I - Q)$ is invertible [10, 19].

Next, we give an expansion technique to convert a network having $m = |A^m|$ memory arcs into one having only memoryless arcs. We first make 2^m copies of the network G. Each copy G' is associated with a different subset $S' \subseteq A^m$ of memory arcs and the random walk takes place on G' if it has successfully crossed all arcs in S' (and none in $A^m \setminus S'$). All reliabilities corresponding to arcs in S' are perfect, because they have by assumption been successfully crossed at least once. Hence, when the random walk traverses a memory arc

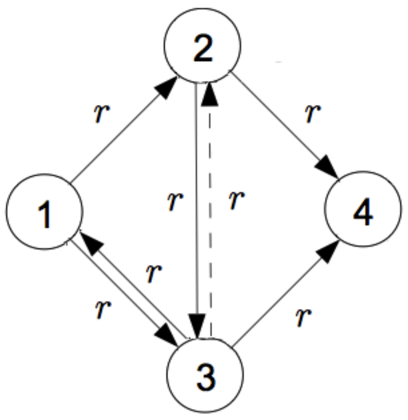


Figure 2: Four-node network in which only arc (3,2) has memory.

(i, j) for the first time, the walk moves from node i in one network G' (for which $(i, j) \notin S'$) to node j in another copy network G'' (whose visited arc set is $S'' = S' \cup \{(i, j)\}$). Note that the final node n need not be duplicated in this process, and hence the total number of nodes required in this transformation is given by $(n-1)2^m + 1$. Before we formalize this method, we illustrate the idea on the network given in Figure 2, where the walk seeks to reach node 4 from node 1. All arcs have equal reliability r in this example, and only arc (3,2) has memory. This network is equivalent to the network with memoryless arcs illustrated in Figure 3, i.e., when the walk crosses arc $(3, 2)$ the first time, it moves to a parallel network copy in which arc $(3, 2)$ has reliability 1. To find $\phi(i)$, we solve the following system of linear equations:

$$\left(\begin{array}{ccc|ccc} 1 & -r/2 & -r/2 & 0 & 0 & 0 \\ 0 & 1 & -r/2 & 0 & 0 & 0 \\ -r/3 & 0 & 1 & 0 & -r/3 & 0 \\ \hline 0 & 0 & 0 & 1 & -r/2 & -r/2 \\ 0 & 0 & 0 & 0 & 1 & -r/2 \\ 0 & 0 & 0 & -r/3 & -1/3 & 1 \end{array}\right) \begin{pmatrix} \phi(1) \\ \phi(2) \\ \phi(3) \\ \phi(1\text{-}1) \\ \phi(2\text{-}1) \\ \phi(3\text{-}1) \end{pmatrix} = \begin{pmatrix} 0 \\ r/2 \\ r/3 \\ 0 \\ r/2 \\ r/3 \end{pmatrix}.$$

This method thus requires the solution of an $(n-1)2^m \times (n-1)2^m$ system of equations. To develop a lower bound on the survival probability, we can choose $k \leq m$ arcs to have memory, treat all remaining arcs as memoryless, and employ the method given above.

To formalize the lower-bounding methodology, suppose a set $S \subseteq A^m$ of arcs is chosen to have memory and the complement of S $(A \backslash S)$ is assumed to be memoryless. If $|S| = k$,

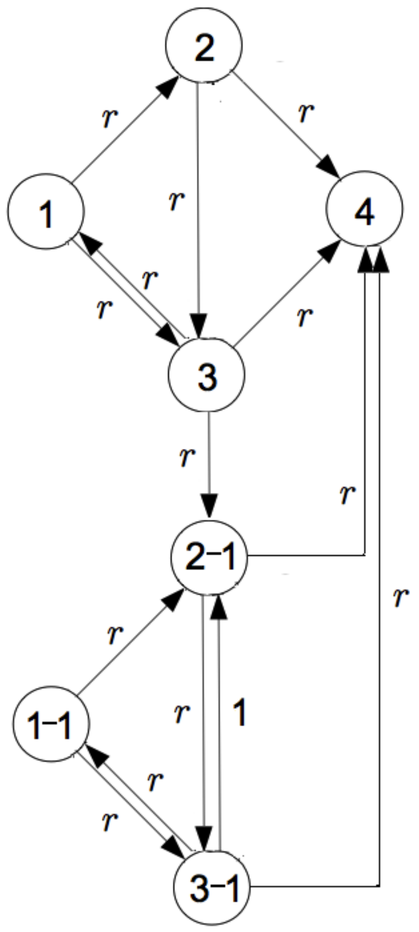


Figure 3: Equivalent memoryless network.

then S has 2^k subsets $S_1, \dots, S_{2^k}$, where $S_1 = \emptyset$ and $S_{2^k} = S$. Define the block matrix

$$\tilde{Q} = \begin{pmatrix} \tilde{Q}^{S_1,S_1} & \tilde{Q}^{S_1,S_2} & \dots & \tilde{Q}^{S_1,S_{2^k}} \\ \tilde{Q}^{S_2,S_1} & \tilde{Q}^{S_2,S_2} & \dots & \tilde{Q}^{S_2,S_{2^k}} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{Q}^{S_{2^k},S_1} & \tilde{Q}^{S_{2^k},S_2} & \dots & \tilde{Q}^{S_{2^k},S_{2^k}} \end{pmatrix}.$$

For $1 \le u \le 2^k, 1 \le v \le 2^k$, matrix $\tilde{Q}^{S_u,S_v}$ is an $(n-1) \times (n-1)$ matrix that has entries $\tilde{q}_{ij}^{S_u,S_v}$ defined as

$$\tilde{q}_{ij}^{S_u,S_v} = \begin{cases} \frac{1}{\delta_i} & \text{if } S_u = S_v,\ (i,j) \in S_u \\ \frac{r_{ij}}{\delta_i} & \text{if } S_u = S_v,\ (i,j) \in A \backslash S \\ \frac{r_{ij}}{\delta_i} & \text{if } S_v = S_u \cup \{(i,j)\},\ (i,j) \in S \backslash S_u \\ 0 & \text{otherwise.} \end{cases}$$

Next, define $\tilde{b}$ as a column vector with $(n-1)2^k$ entries. Let $j(i) = i \bmod (n-1)$ if i is not a multiple of $n-1$, and $j(i) = (n-1)$ otherwise. The $\tilde{b}$-vector is given as:

$$\tilde{b}_i = \begin{cases} \frac{r_{j(i)n}}{\delta_i} & \text{if } (j(i), n) \in A \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

Then, by solving $\phi = (I - \tilde{Q})^{-1} \tilde{b}$, a lower bound is given by $\phi(1)$, where I is the $(n-1)2^k$-dimensional identity matrix.

The method described above yields a valid lower bound for any choice of $S \subset A^m$. However, we can possibly improve this lower bound by strategically choosing arcs in S. Consider the survival probability calculation in which there exists only one memory arc, i.e., $A^m = \{(i_1, j_1)\}$ for some arc $(i_1, j_1) \in A$. The following analysis quantifies the increase in survival probability by having memory on this arc. To simplify our notation we define

$$\begin{aligned} \rho(i) &= \mathbb{P}(\text{walk hits node } n \text{ without crossing arc } (i_1, j_1) | Y_0 = i) \\ \psi(i) &= \mathbb{P}(\text{walk attempts to cross arc } (i_1, j_1) \text{ before hitting node } n | Y_0 = i). \end{aligned}$$

Notice that $\psi(i)$ calculates the probability that the walk hits node i_1 and attempts to traverse (i_1, j_1), without accounting for failure or success while crossing arc (i_1, j_1). To calculate $\rho(i)$ and $\psi(i)$ for each $i \in V \setminus \{n\}$, we define matrix $\tilde{C}_{i_1 j_1}$ such that $\tilde{c}_{i_1 j_1} = 0$ and $\tilde{c}_{ij} = q_{ij}$ as defined above when $(i,j) \in A \setminus \{(i_1, j_1)\}$. Define b as in (16) and define vector c^{i_1} such that $c_{i_1}^{i_1} = 1/\delta_{i_1}$ and $c_i^{i_1} = 0$ for $i \in V \setminus \{i_1, n\}$. Then, we can calculate ρ and ψ as

$$\begin{aligned} \rho &= (I - \tilde{C}_{i_1 j_1})^{-1} b \\ \psi &= (I - \tilde{C}_{i_1 j_1})^{-1} c^{i_1}. \end{aligned}$$

Let $\phi(1,t)$ be the probability of reaching the destination node n after crossing arc (i_1, j_1) exactly t times given that the walk starts from node 1. Hence,

$$\phi(1) = \sum_{t=0}^{\infty} \phi(1,t). \tag{18}$$

From the definition, $\phi(1,0) = \rho(1)$. For $\phi(1,1)$, the walk must cross (i_1, j_1) once, and then arrive at node n without attempting to cross (i_1, j_1) again. Hence,

$$\phi(1,1) = \psi(1) r_{i_1 j_1} \rho(j_1).$$

Similarly, we can calculate $\phi(1,t)$ as

$$\phi(1,t) = \psi(1) r_{i_1 j_1} \psi(j_1)^{t-1} \rho(j_1),\ t \geq 2. \tag{19}$$

Replacing the expression given by (19) in (18) yields

$$\phi(1) = \rho(1) + \frac{r_{i_1 j_1} \psi(1) \rho(j_1)}{1 - \psi(j_1)}. \tag{20}$$

Lemma 2. *The increase in the survival probability by having memory on arc* (i_1, j_1) *is*

$$\Delta\phi_{(i_1,j_1)} = \frac{r_{i_1 j_1}(1 - r_{i_1 j_1}) \psi(1) \psi(j_1) \rho(j_1)}{(1 - \psi(j_1))(1 - r_{i_1 j_1} \psi(j_1))}. \tag{21}$$

Proof. The survival probability when arc (i_1, j_1) has memory is given by (20). If the arc were not to have memory, $\phi(1,0)$ and $\phi(1,1)$ would not change, but for $t \geq 2$, we would modify (19) as

$$\phi(1,t) = \psi(1) r_{i_1 j_1}^t \psi(j_1)^{t-1} \rho(j_1),$$

and hence the total probability would be

$$\rho(1) + \frac{r_{i_1 j_1} \psi(1) \rho(j_1)}{1 - r_{i_1 j_1} \psi(j_1)}. \tag{22}$$

Subtracting (22) from (20), the result follows. □

Lemma 2 quantifies the gain by having memory on a given arc, while all other arcs are memoryless. As a heuristic means to select a set of $k < m$ arcs that have memory, we calculate (21) for each arc, sort the arcs in nonincreasing order of these values, and select the first k arcs to have memory.

3.2 Upper-Bounding Technique

In Section 3.1 we obtained a lower bound on the survival probability by assuming that a subset of the arcs remain unreliable, even after they are successfully crossed. By contrast, we obtain an upper bound in this section by assuming that some arcs may become fully reliable even before they are crossed.

Toward this objective, we partition the set of memory arcs A^m into k clusters $A_1, \dots, A_k$. Then, we assume that once one arc in a cluster is traversed, all other arcs in the same cluster become fully reliable. The following lemma formally states the key upper-bounding principle used in this section.

Lemma 3. *Let W be the set of all origin-destination walks on G, and let $H_w \subseteq \{A_1, \dots, A_k\}$ be the set of all clusters such that walk $w \in W$ visits at least one arc in the cluster. Define $\gamma^{h,w}$ as the first arc crossed in cluster $A_h \in H_w$ during walk $w \in W$. Then*

$$\sum_{w \in W} \left(\prod_{i \in V \setminus \{n\}} \frac{1}{\delta_i^{N^{w,i}}} \right) \left(\prod_{A_h \in H_w} r_{\gamma^{h,w}} \right) \left(\prod_{(i,j) \in A \setminus A^m} r_{ij}^{N^{w,(i,j)}} \right) \tag{23}$$

is an upper bound on the survival probability.

Proof. The result follows by comparing (14) and (23). □

Because (23) requires enumerating all origin-destination walks, calculating the upper bound using this formula is generally impractical. However, using the intuition behind the upper-bounding clusters, let $T = \{1, \dots, k\}$ and $T_1, \dots, T_{2^k}$ be the subsets of T. We now define the following block matrix:

$$\tilde{B} = \begin{pmatrix} \tilde{B}^{T_1,T_1} & \tilde{B}^{T_1,T_2} & \dots & \tilde{B}^{T_1,T_{2^k}} \\ \tilde{B}^{T_2,T_1} & \tilde{B}^{T_2,T_2} & \dots & \tilde{B}^{T_2,T_{2^k}} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{B}^{T_{2^k},T_1} & \tilde{B}^{T_{2^k},T_2} & \dots & \tilde{B}^{T_{2^k},T_{2^k}} \end{pmatrix}.$$

Define $U_l = \bigcup_{i \in T_l} A_i$. For $1 \le u \le k,\ 1 \le v \le k$, matrix $\tilde{B}^{T_u,T_v}$ is an $(n-1) \times (n-1)$ matrix with entries $\tilde{b}_{ij}^{T_u,T_v}$ defined as

$$\tilde{b}_{ij}^{T_u,T_v} = \begin{cases} \frac{1}{\delta_i} & \text{if } T_u = T_v,\ (i,j) \in U_u \\ \frac{r_{ij}}{\delta_i} & \text{if } T_u = T_v,\ (i,j) \in A \setminus A^m \\ \frac{r_{ij}}{\delta_i} & \text{if } \exists l \in T : U_v = U_u \cup A_l,\ (i,j) \in A_l, \text{ and } A_l \cap U_u = \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

Defining the $(n-1)2^k$-dimensional column vector $\tilde{b}$ as in (17), we set $\phi = (I - \tilde{B})^{-1}\tilde{b}$ and obtain an upper bound of $\phi(1)$ on the survival probability.

Similar to the lower-bounding method, the upper-bounding method is valid for any partitioning of the set of memory arcs. The next two subsections detail approaches for improving the upper bound based on our choice of the partitioning scheme.

3.2.1 Strategic clustering approaches

Following the same principle given in Section 3.1, we will attempt to partition arcs into clusters, so that the overestimation in the upper bound caused by placing multiple arcs in a common cluster is mitigated. Accordingly, we devise a metric that assesses the overestimation in the upper bound due to placing distinct arcs (i_1, j_1) and (i_2, j_2) in the same cluster. Toward this goal, suppose that no arc aside from (i_1, j_1) and (i_2, j_2) has memory. We can write the survival probability as follows:

$$\begin{aligned}\phi(1) = \; & \mathbb{P}(\text{hitting } n \text{ without crossing } (i_1, j_1) \text{ or } (i_2, j_2) | Y_0 = 1) \\ & + \mathbb{P}(\text{hitting } n \text{ after crossing only } (i_1, j_1) | Y_0 = 1) \\ & + \mathbb{P}(\text{hitting } n \text{ after crossing only } (i_2, j_2) | Y_0 = 1) \\ & + \mathbb{P}(\text{hitting } n \text{ after crossing } (i_1, j_1) \text{ first and then } (i_2, j_2) | Y_0 = 1) \\ & + \mathbb{P}(\text{hitting } n \text{ after crossing } (i_2, j_2) \text{ first and then } (i_1, j_1) | Y_0 = 1).\end{aligned} \tag{24}$$

We seek to compute the difference in (24) when (i_1, j_1) and (i_2, j_2) are assigned to the same cluster, and when they are assigned to different clusters. In both cases, the first three terms of (24) will be identical. For simplicity in notation, we replace $e_1 = (i_1, j_1)$ and $e_2 = (i_2, j_2)$ where convenient below. Define P_1^s and P_2^s to be the fourth and fifth terms of (24), respectively, when e_1 and e_2 are assigned to the same cluster, and define P_1^d and P_2^d analogously when e_1 and e_2 are assigned to different clusters. For the fourth term of (24), we have

$$\begin{aligned}P_1^s = \; & \mathbb{P}(\text{hitting } n \text{ after crossing } e_1 \text{ first and then } e_2 | Y_0 = 1) \\ = \; & \mathbb{P}(\text{attempting to cross } e_1 \text{ before crossing } e_2 | Y_0 = 1) r_{e_1} \\ & \times \mathbb{P}(\text{crossing } e_2 \text{ and then hitting } n | Y_0 = j_1,\ r_{e_1} := 1),\end{aligned} \tag{25}$$

where the notation $r_{e_1} := 1$ indicates that the reliability of arc e_1 is set to 1 in the calculation of this probability term. When these arcs are in different clusters, we have

$$\begin{aligned}P_1^d = \; & \mathbb{P}(\text{attempting to cross } e_1 \text{ before crossing } e_2 | Y_0 = 1) r_{e_1} r_{e_2} \\ & \times \mathbb{P}(\text{crossing } e_2 \text{ and then hitting } n | Y_0 = j_1,\ r_{e_1} := 1).\end{aligned} \tag{26}$$

The values for P_2^s and P_2^d are similarly defined for the case in which the walk reaches n after crossing e_2 before crossing e_1 by swapping e_1 and e_2 in (25) and (26). To calculate these values, we must compute for each $i \in V \setminus \{n\}$:

$$\begin{aligned}
\alpha_{e_1\setminus e_2}(i) &= \mathbb{P}(\text{attempting to cross } e_1 \text{ without crossing } e_2 | Y_0 = i) \\
\psi_{e_2|e_1}(i) &= \mathbb{P}(\text{attempting to cross } e_2 | Y_0 = i,\ r_{e_1} := 1) \\
\lambda_{e_1,e_2}(i) &= \mathbb{P}(\text{walk hits node } n | Y_0 = i,\ r_{e_1} := 1,\ r_{e_2} := 1).
\end{aligned}$$

In computing the α-values, the walk should fail if arc e_2 or any arc of the form (i, n), for $i \in V$, is traversed. The walk should terminate if it attempts to traverse e_1. Similar logic holds for ψ, except that because e_1 was already crossed, its reliability should be perfect, and if the walk attempts to traverse e_2, then the walk ends successfully. Finally, the λ calculation simply assumes that arc reliabilities for e_1 and e_2 are perfect, and that the walk terminates at node n. Therefore, to calculate these values, we require slight modifications of the $(n-1)\times(n-1)$ matrix Q derived in Section 3.1. Define matrix $\tilde{C}_{A_0}^{A_1}$ exactly as matrix Q, but where all arcs in A_0 have reliability 0 and all arcs in A_1 have reliability 1. For example, in the computation of ψ, we will use $\tilde{C}_{e_2}^{e_1}$, which indicates that the entry corresponding to e_1 is set to $1/\delta_{i_1}$ (a perfect-reliability arc), and the entry corresponding to e_2 is set to 0 (a result of the walk terminating when it attempts to traverse this arc). Recall that vector c^i is the zero vector except where $c_i^i = 1/\delta_i$. Using the conditioning argument in Section 3.1, we can write

$$\begin{aligned}
\alpha_{e_1\setminus e_2} &= (I - \tilde{C}_{e_1,e_2})^{-1} c^{i_1} \\
\psi_{e_2|e_1} &= (I - \tilde{C}_{e_2}^{e_1})^{-1} c^{i_2} \\
\lambda_{e_1,e_2} &= (I - \tilde{C}^{e_1,e_2})^{-1} b,
\end{aligned}$$

where b is given by (16). Using this notation, we can write

$$\begin{aligned}
P_1^s + P_2^s &= \alpha_{e_1\setminus e_2}(1) r_{e_1} \psi_{e_2|e_1}(j_1) \lambda_{e_1,e_2}(j_2) + \alpha_{e_2\setminus e_1}(1) r_{e_2} \psi_{e_1|e_2}(j_2) \lambda_{e_1,e_2}(j_1) \\
P_1^d + P_2^d &= \alpha_{e_1\setminus e_2}(1) r_{e_1} r_{e_2} \psi_{e_2|e_1}(j_1) \lambda_{e_1,e_2}(j_2) + \alpha_{e_2\setminus e_1}(1) r_{e_2} r_{e_1} \psi_{e_1|e_2}(j_2) \lambda_{e_1,e_2}(j_1).
\end{aligned}$$

Taking the difference, we can quantify the *loss coefficient* ℓ_{12}, i.e., the upper-bound increase due to assigning e_1 and e_2 to the same cluster, as

$$\begin{aligned}
\ell_{12} = (P_1^s + P_2^s) - (P_1^d + P_2^d) = \alpha_{e_1\setminus e_2}(1) r_{e_1} (1 - r_{e_2}) \psi_{e_2|e_1}(j_1) \lambda_{e_1,e_2}(j_2) \\
+ \alpha_{e_2\setminus e_1}(1) r_{e_2} (1 - r_{e_1}) \psi_{e_1|e_2}(j_2) \lambda_{e_1,e_2}(j_1). \quad (27)
\end{aligned}$$

Given ℓ_{uv}, for each pair of arcs (i_u, j_u) and (i_v, j_v), we suggest the following clustering algorithm. Define P_A as the set of arcs that have been assigned to a cluster, P_U as the set of unassigned arcs, and k ($\leq m$) as the number of clusters used in the upper-bounding scheme. The first phase of our algorithm seeds each cluster with a single arc. We begin this phase by placing arc (i_1, j_1) (arbitrarily) in the first cluster, and move (i_1, j_1) from P_U to P_A (as we do each time an arc is assigned to a cluster). For each subsequent cluster, we use a max-min approach to determine the first arc to be assigned to the cluster. For each arc $(i_v, j_v) \in P_U$, we compute ℓ_{uv} over all arcs $(i_u, j_u) \in P_A$. We then let $v^* \in \text{argmax}_{v \in P_U} \{\min_{u \in P_A}\{\ell_{uv}\}\}$, i.e., placing arc (i_{v^*}, j_{v^*}) in a cluster with any other assigned arc results in the maximum loss coefficient over all unassigned arcs, even if (i_{v^*}, j_{v^*}) is assigned to a cluster that minimizes its loss coefficient. Arc (i_{v^*}, j_{v^*}) becomes the first arc assigned to the next empty cluster. Once each cluster is nonempty, we assign all remaining arcs in P_U to clusters as follows. While P_U is nonempty, we find an arc $(i_u, j_u) \in P_U$ and cluster A_h that minimizes the sum of ℓ_{uv} over all arcs (i_v, j_v) already assigned to A_h, and place arc (i_u, j_u) into cluster A_h. The procedure repeats until all arcs are assigned to a cluster.

3.2.2 Reducing upper-bound overestimation within clusters

Observe that for each cluster A_h and arc $(i,j) \in A_h$, the foregoing method accounts only for the probability of failing on (i,j) if it is the first arc traversed in A_h. However, it is possible to tighten the upper-bound estimate by examining the maximum probability that the walk does not fail on any arc in A_h, starting from arc (i,j). In this manner, we can possibly (depending on the problem instance) account for other arc reliabilities in the cluster aside from r_{ij}.

Let $\gamma^{h,w} \in A_h$ be the first arc traversed in cluster A_h during walk w, and define $W_{\gamma^{h,w}}$ as the set of all origin-destination walks that cross arc $\gamma^{h,w}$, and do so before any other arc in A_h is traversed. Then, we can write the second product in (13) as

$$\prod_{(i,j)\in A^m : N^{w,(i,j)}>0} r_{ij} = \prod_{A_h \in H_w} \omega_{A_h}, \text{ where } \omega_{A_h} = \prod_{(i,j)\in A_h : N^{w,(i,j)}>0} r_{ij}, \quad \forall A_h \in H_w.$$

Lemma 3 bounds ω_{A_h} from above by $r_{\gamma^{h,w}}$. We attempt to improve this bound by computing the maximum possible value of ω_{A_h} over all walks that visit $\gamma^{h,w}$ before any other arc in cluster A^h. Toward this goal, consider the following inequality:

$$\begin{aligned}
\prod_{(i,j)\in A_h : N^{w,(i,j)}>0} r_{ij} &\leq \max_{\bar{w} \in W_{\gamma^{h,w}}} \left\{ \prod_{(i,j)\in A_h : N^{\bar{w},(i,j)}>0} r_{ij} \right\} \\
&= \exp\left(\max_{\bar{w} \in W_{\gamma^{h,w}}} \left\{ \sum_{(i,j)\in A_h : N^{\bar{w},(i,j)}>0} \ln r_{ij} \right\} \right) \\
&= \exp\left(- \min_{\bar{w} \in W_{\gamma^{h,w}}} \left\{ \sum_{(i,j)\in A_h : N^{\bar{w},(i,j)}>0} - \ln r_{ij} \right\} \right). \qquad (28)
\end{aligned}$$

Inequality (28) suggests a possible improvement to the foregoing upper bound, wherein all arcs in A_h become perfectly reliable after arc $\gamma^{h,w} = (i,j)$ was initially traversed. In particular, we can revise the reliability of arc $\gamma^{h,w}$ to take on the right-hand side of (28) (which is no more than $r_{\gamma^{h,w}}$), thus incorporating the reliability of other arcs within A_h that must inevitably be traversed before reaching node n, given that the walk enters A_h via $\gamma^{h,w}$. The challenge remains to compute this right-hand side value efficiently.

The minimization problem appearing in (28) corresponds to the optimal objective function value of a shortest-path problem from the to-node of arc $\gamma^{h,w}$ to node n, multiplied by $r_{\gamma^{h,w}}$. Intuitively, this shortest-path cost corresponds to the most reliable path from the to-node of $\gamma^{h,w}$ to node n, given that all arc reliabilities are perfect except for those belonging to A_h. Thus, we seek the shortest-path cost from the to-node of $\gamma^{h,w}$ and node n on the same network $G(V,A)$ defined for the RWSM instance, where each arc $(i,j) \in A_h$ for which $r_{ij} > 0$ has a cost of $-\ln r_{ij}$, each arc $(i,j) \in A_h$ for which $r_{ij} = 0$ is deleted, and all other arcs have a cost of zero.

Instead of individually computing each such shortest path, an alternative approach employs (backwards) Dijkstra's algorithm, which computes optimal shortest-path costs for all nodes in V to node n in time proportional to $O(|A| + n \log n)$ [2]. Hence, our strategy computes the shortest-path cost, θ_i^h, from every node i to node n for each cluster A_h (generating the appropriate arc costs for A_h). The reliability value for arc $(i,j) \in A_h$ is then revised to r_{ij} multiplied by $\exp(-\theta_j^h)$.

3.3 The Bounding Algorithms

Now, we state the bounding algorithms formally using the ideas developed in the previous sections. In the discussion below, I denotes the identity matrix having appropriate dimensions. The bounding algorithms are outlined in Algorithms 1 and 2.

The algorithm to calculate lower and upper bounds for the survival probability requires the solution of many linear systems of equations. A close investigation of these systems shows that the matrices to be inverted have similar structure, differing by only one or two entries. Hence, the inverse of the related matrices can be calculated by updating a previous inverse, which takes only $O(n^2)$ operations. To achieve this, we rely on the following lemma.

Lemma 4. ***(Miller [16])*** *If G is invertible and H is rank one, then*

$$(G+H)^{-1} = G^{-1} - \frac{1}{1+g} G^{-1} H G^{-1}, \tag{29}$$

where $g = trace(HG^{-1})$.

Algorithm 1 Finding a lower bound on the survival probability.

```
Require: Network $G(V, A)$ with reliabilities, set of memory arcs $A^m$, and $k_1$ (the number of memory arcs used to calculate the lower bound)
  Set $b$ as in (16)
  for all $(i, j) \in A^m$ do
    Set $\tilde{C}_{ij}$ and $c^i$ as in Section 3.1
    Set $\rho \leftarrow (I - \tilde{C}_{ij})^{-1} b$ and $\psi \leftarrow (I - \tilde{C}_{ij})^{-1} c^i$
    Set $\Delta\phi_{(i,j)}$ as in (21)
  end for
  Sort $\Delta\phi_{(i,j)}$ in nonincreasing order and choose the first $k_1$ arcs to belong to set $S$ (to have memory).
  Set $\tilde{Q}$ and $\tilde{b}$ as in Section 3.1
  $\phi \leftarrow (I - \tilde{Q})^{-1} \tilde{b}$
  return Lower Bound: $\phi(1)$
```

For a given $(i, j) \in A^m$, we define the rank-one $(n-1) \times (n-1)$ matrix $\tilde{D}^{ij}(x)$ having all zero entries except for $d_{ij} = x$. Suppose that we have $F = (I - Q)^{-1}$ with entries f_{kl}. Then

$$\begin{aligned}(I - \tilde{C}_{ij})^{-1} &= (I - Q + \tilde{D}^{ij}(\tfrac{r_{ij}}{\delta_i}))^{-1} \\ &= (I - Q)^{-1} - \tfrac{1}{1+g}(I - Q)^{-1}\tilde{D}^{ij}(\tfrac{r_{ij}}{\delta_i})(I - Q)^{-1},\end{aligned}$$

where $g = r_{ij} f_{ji}/\delta_i$ and

$$(I - Q)^{-1}\tilde{D}^{ij}(r_{ij})(I - Q)^{-1} = \frac{r_{ij}}{\delta_i}\begin{pmatrix} f_{1i}f_{j1} & f_{1i}f_{j2} & \cdots & f_{1i}f_{j(n-1)} \\ f_{2i}f_{j1} & f_{2i}f_{j2} & \cdots & f_{2i}f_{j(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ f_{(n-1)i}f_{j1} & f_{(n-1)i}f_{j2} & \cdots & f_{(n-1)i}f_{j(n-1)} \end{pmatrix}.$$

The other inverse matrices needed in the bounding algorithm can be iteratively updated in this manner. For example, given $e_1 = (i_1, j_1) \in A^m$ and $e_2 = (i_2, j_2) \in A^m$:

$$\begin{aligned}(I - \tilde{C}_{e_1}^{e_2})^{-1} &= (I - \tilde{C}_{e_1} + \tilde{D}^{e_2}(\tfrac{r_{i_2 j_2} - 1}{\delta_{i_2}}))^{-1} \\ &= (I - \tilde{C}_{e_1})^{-1} - \tfrac{1}{1+g}(I - \tilde{C}_{e_1})^{-1}\tilde{D}^{e_2}(\tfrac{r_{i_2 j_2} - 1}{\delta_{i_2}})(I - \tilde{C}_{e_1})^{-1},\end{aligned}$$

where g is defined accordingly.

While calculating the lower and upper bounds, we need to invert $n2^k \times n2^k$ matrices, $I - \tilde{Q}$ and $I - \tilde{B}$, which would evidently require $O(n^3 2^{3k})$ operations. However, note that $\tilde{Q}^{S_u, S_v}$ is a matrix of all zeros if $S_u \not\subseteq S_v$, and a similar result holds for $\tilde{B}^{T_u, T_v}$. Hence, both matrices are block upper-diagonal matrices, and both $I - \tilde{Q}$ and $I - \tilde{B}$ can be inverted using $O(n^3 2^{2k})$ operations. This time can be further reduced using Lemma 4.

Algorithm 2 Finding an upper bound on the survival probability.

```
Require: Network $G(V,A)$ with reliabilities, set of memory arcs $A^m$, $k_2$ (the number of
  clusters used to calculate the upper bound)
  Set $T_1 = \cdots = T_{k_2} = \emptyset$, $T = \emptyset$ ($T$ represents the arcs classified so far, i.e., $T = \bigcup_{l=1}^{k_2} T_l$)
  {Assign one arc to each cluster}
  Assign $(i_1, j_1) \in T_1,\ (i_1, j_1) \in T$
  $k \leftarrow 2$
  for all $k = 2, \ldots, k_2$ do
    for all $e_1 = (i_1, j_1) \in A^m \backslash T,\ e_2 = (i_2, j_2) \in T$ do
      Set $\alpha_{e_2 \backslash e_1}, \alpha_{e_1 \backslash e_2}, \psi_{e_1|e_2}, \psi_{e_2|e_1}$ and $\lambda_{e_1,e_2}$ as in Section 3.2.
      Compute $\ell_{e_1,e_2}$ as in (27)
    end for
    Choose $(i^*, j^*) \in \arg\max_{(i,j)\in A^m \backslash T}\{\min_{(u,v)\in T} \ell_{(i,j),(u,v)}\}$
    Assign $(i^*, j^*)$ to $T_k$ and $T$
  end for
  {Assign the remaining arcs to the clusters}
  while $T \neq A^m$ do
    for all $(i,j) \in A^m \backslash T$ and $1 \leq k \leq k_2$ do
      Set $\ell^k_{(i,j)} \leftarrow \sum_{(u,v)\in T_k} \ell_{(i,j),(u,v)}$
    end for
    Choose $(i^*, j^*, k^*) \in \arg \min\limits_{(i,j)\in T \backslash A^m} \left\{ \min\limits_{k=1,\ldots,k_2} \ell^k_{(i,j)} \right\}$
    Assign $(i^*, j^*)$ to $T_{k^*}$ and $T$
  end while
  {Calculate an upper bound based on the arc clustering found above}
  Set $\tilde{B}$ and $\tilde{b}$ as in Section 3.2
  $\phi \leftarrow (I - \tilde{B})^{-1}\tilde{b}$
  return Upper Bound: $\phi(1)$
```

4 Computational Results

Now, we turn our attention to an empirical analysis of the concepts introduced in Section 3. Given a network $G(V, A)$, along with a set of arc reliabilities, recall from Lemma 1 that the survival probability for the memoryless-arc case is no more than that for the case in which arcs have memory. We first empirically quantify the impact of having memory in our first set of experiments in Section 4.1, along with the effect of varying reliability values and arc density. In Section 4.2, we study the tightness of the lower and upper bounds presented in Sections 3.1 and 3.2, respectively. We perform all experiments on randomly generated networks using Algorithm 3. This algorithm places an arc between each possible node pair with probability p (where p is a pre-specified network density parameter), and assigns a reliability value uniformly generated on the interval $[l, u]$ for each arc. We require that there is at least one path between the origin and the destination node to avoid trivialities. If a generated network does not satisfy this property, we reject the network and generate another one.

4.1 Impact of Arc Memory on Survival Probability

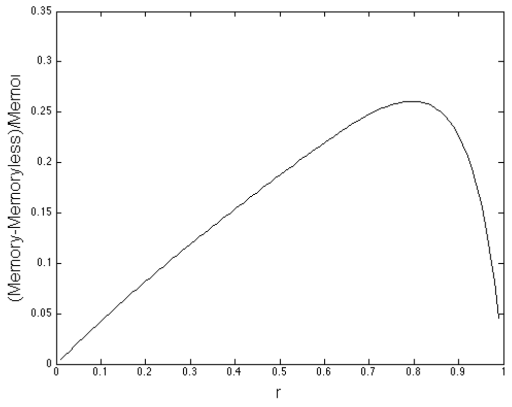
Recall that calculating the (exact) survival probability requires the solution of an $(n-1)2^m$ square system of equations, where n is the number of nodes and m is the number of memory arcs. Thus, in order to perform calculations on dense networks, we initially consider seven-node networks in which each arc has a common reliability value. To explore the relationship between arc reliability and survival probability with respect to memory, we examine the relative survival probabilities of the cases in which arcs have memory and lack memory. Specifically, define the *impact* of memory to be the difference in survival probabilities with and without memory, divided by the survival probability of the memory case. Table 1 and Figure 4 illustrate the impact of memory as a function of r. When $r = 0$ and $r = 1$, the survival probability is given by 0 and 1, respectively, for both memory and memoryless cases. Hence, arc memory has no impact for these boundary cases. We expect the impact of memory to initially increase with r, and then decrease as r approaches 1. We observe that the maximum impact generally occurs when r is relatively close to 1. To magnify the impact of having memory, our subsequent analysis will thus focus on common arc reliability $r = 0.9$.

Algorithm 3 Generating random graphs for testing the proposed methods.

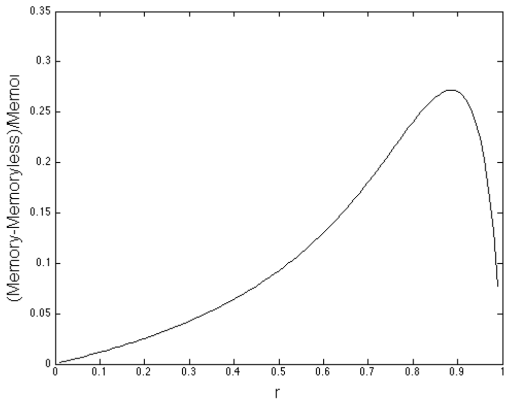
```
Require: Set of nodes ($V$), network density parameter ($p$), reliability lower bound ($l$) and
  reliability upper bound ($u$)
  Set pathflag = false
  while pathflag = false do
    $A \leftarrow \emptyset$
    for all $i = 1, \ldots, |V|$ and $j = 1, \ldots, |V|$ do
      Generate an arc from node $i$ to node $j$ with probability $p$
      if arc $(i, j)$ is generated then
        Set $A \leftarrow A \cup \{(i, j)\}$
        Generate $r_{ij}$ uniformly between $l$ and $u$
      else
        Set $r_{ij} \leftarrow 0$
      end if
    end for
    Set $Q$ and $b$ as in (15) and (16)
    if a path from 1 to $n$ exists using arcs $A$ then
      $\phi(1) \leftarrow (I - Q)^{-1}b$
      Set pathflag = true
    end if
  end while
  return $G(V, A)$ and $r$
```

	$r = 0.25$		$r = 0.50$		$r = 0.75$	
Arcs	Memory	Memoryless	Memory	Memoryless	Memory	Memoryless
10	1.007×10^{-1}	1.000×10^{-1}	2.559×10^{-1}	2.500×10^{-1}	5.189×10^{-1}	5.000×10^{-1}
10	5.907×10^{-2}	5.845×10^{-2}	1.545×10^{-1}	1.477×10^{-1}	3.595×10^{-1}	3.220×10^{-1}
10	2.138×10^{-2}	2.051×10^{-2}	1.083×10^{-1}	1.000×10^{-1}	3.439×10^{-1}	3.102×10^{-1}
12	5.248×10^{-3}	4.775×10^{-3}	5.222×10^{-2}	4.455×10^{-2}	2.478×10^{-1}	2.478×10^{-1}
12	1.806×10^{-3}	1.572×10^{-3}	2.326×10^{-2}	1.767×10^{-2}	1.586×10^{-1}	1.080×10^{-1}
12	1.358×10^{-1}	1.347×10^{-1}	3.044×10^{-1}	3.009×10^{-1}	5.503×10^{-1}	5.380×10^{-1}
14	1.405×10^{-3}	1.405×10^{-3}	1.574×10^{-2}	1.370×10^{-2}	1.043×10^{-1}	7.692×10^{-2}
14	2.670×10^{-3}	2.458×10^{-3}	2.941×10^{-2}	2.511×10^{-2}	1.685×10^{-1}	1.336×10^{-1}
14	1.330×10^{-1}	1.327×10^{-1}	2.936×10^{-1}	2.911×10^{-1}	5.246×10^{-1}	5.134×10^{-1}
16	1.841×10^{-2}	1.721×10^{-2}	9.204×10^{-2}	8.271×10^{-2}	2.939×10^{-1}	2.604×10^{-1}
16	2.033×10^{-2}	1.981×10^{-2}	9.467×10^{-2}	9.001×10^{-2}	2.919×10^{-1}	2.693×10^{-1}
16	5.511×10^{-2}	5.490×10^{-2}	1.341×10^{-1}	1.313×10^{-1}	2.999×10^{-1}	2.794×10^{-1}
18	5.511×10^{-2}	5.486×10^{-2}	1.332×10^{-1}	1.304×10^{-1}	2.913×10^{-1}	2.735×10^{-1}
18	9.544×10^{-4}	9.113×10^{-4}	1.100×10^{-2}	9.949×10^{-3}	7.288×10^{-2}	5.949×10^{-2}
18	9.329×10^{-2}	9.308×10^{-2}	2.175×10^{-1}	2.155×10^{-1}	4.201×10^{-1}	4.084×10^{-1}
20	1.252×10^{-2}	1.222×10^{-2}	6.558×10^{-2}	6.242×10^{-2}	2.303×10^{-1}	2.124×10^{-1}
20	5.805×10^{-3}	5.678×10^{-3}	2.896×10^{-2}	2.751×10^{-2}	1.077×10^{-1}	9.478×10^{-2}
20	5.444×10^{-2}	5.433×10^{-2}	1.277×10^{-1}	1.263×10^{-1}	2.715×10^{-1}	2.589×10^{-1}

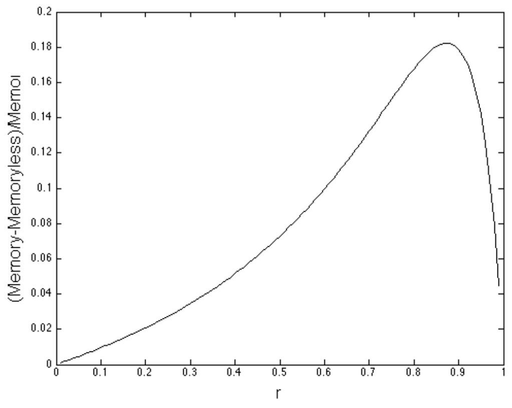
Table 1: Survival probabilities for randomly generated seven-node networks.



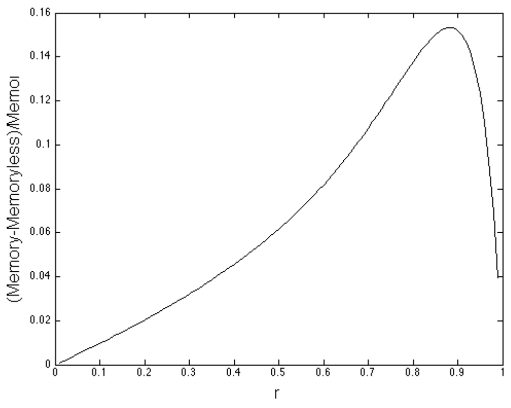
(a) A 7-node, 12-arc network



(b) A 7-node, 14-arc network



(c) A 7-node, 16-arc network



(d) A 7-node, 18-arc network

Figure 4: Ratio of survival probabilities (given by (memory-memoryless)/memoryless) as a function of arc reliabilities.

4.2 Computational Results of the Bounding Techniques

The second topic we investigate regards the effectiveness of the lower- and upper-bounding techniques introduced in Section 3 in estimating survival probability. Table 2 and Figures 5 and 6 present the performance of our lower- and upper-bounding techniques on six different networks having various sizes, and Table 3 presents the total computation time required to obtain bounds for these networks. These instances are randomly generated using Algorithm 3 with two caveats. One, the probability of generating an arc of the form (i, n) is $p = 0.3$ for the less-dense instances (where $(n, m) = (8, 18), (12, 30)$, and $(20, 84)$) and is $p = 0.4$ for the denser instances (where $(n, m) = (8, 26), (12, 42)$, and $(20, 110)$). Two, recall that a common value of r is set to 0.9 in order to magnify the impact of arc memory. The arcs of the form (i, n) may only be crossed once and the walk terminates after one of these arcs is crossed. Hence, we treat these arcs as memoryless and assume that all other arcs in the original network have memory.

The column labeled **LB Memory Arcs/Clusters** in Table 2 indicates the number of memory arcs in the lower-bounding case (k_1) or the number of clusters (k_2) in the upper-bounding case. In the following text, we thus refer to k $(= k_1 = k_2)$ as the number of memory arcs or clusters. Recall that Algorithms 1 and 2 include methods to specify the composition of the memory arcs or clusters: We refer to the use of these methods as "strategic selection" for the lower-bounding scheme and "strategic clustering" for the upper-bounding scheme. An alternative mechanism would simply randomly select arcs to have memory in the lower-bounding scheme, and randomly partition the arcs into k clusters in the upper-bounding scheme. Hence, the fourth and fifth columns correspond to lower-bound values when the memory arcs are chosen strategically and randomly, respectively. Similarly, the sixth and seventh columns respectively correspond to strategic and random clustering methods for obtaining upper bounds. Table 2 shows that the bounds become progressively tighter as we increase k and employ the strategic methods developed in Section 3.

Table 3 indicates that the computation times for strategically and randomly chosen arcs or clusters are comparable. This indicates that the major computational burden lies in solving the resulting set of equations, and that the computational effort required for strategic selection takes a negligible amount of time. For large k, the computational time roughly doubles if k increases by one: As a result, the computation times for computing bounds with $k = 20$ are approximately 32 times longer than when $k = 15$. Slightly more computational effort is required to calculate upper bounds than lower bounds, because the matrix to be inverted when solving the required system of equations is denser.

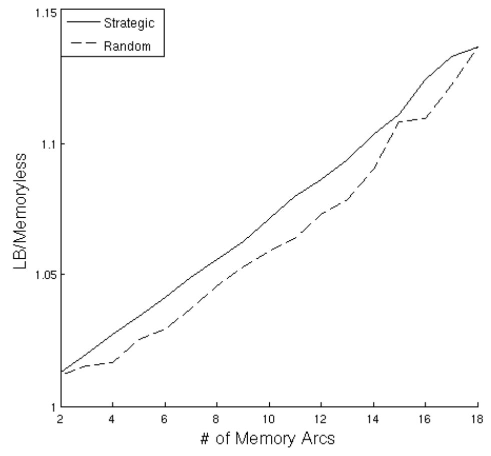
To further investigate the effect of strategic selection, we compared the bounds obtained

Nodes	Arcs	LB Memory Arcs/ Clusters	LB Strategic	LB Random	UB Strategic	UB Random
8	18	0	4.634×10^{-1}	4.634×10^{-1}	—	—
		5	4.794×10^{-1}	4.753×10^{-1}	6.549×10^{-1}	6.663×10^{-1}
		10	4.965×10^{-1}	4.907×10^{-1}	5.707×10^{-1}	5.824×10^{-1}
		15	5.149×10^{-1}	5.135×10^{-1}	5.380×10^{-1}	5.427×10^{-1}
		18	5.268×10^{-1}	5.268×10^{-1}	5.268×10^{-1}	5.268×10^{-1}
8	26	0	5.131×10^{-1}	5.131×10^{-1}	—	—
		5	5.211×10^{-1}	5.187×10^{-1}	6.686×10^{-1}	6.790×10^{-1}
		10	5.273×10^{-1}	5.239×10^{-1}	5.977×10^{-1}	6.048×10^{-1}
		15	5.338×10^{-1}	5.290×10^{-1}	5.695×10^{-1}	5.765×10^{-1}
		20	5.399×10^{-1}	5.381×10^{-1}	5.570×10^{-1}	5.635×10^{-1}
12	30	0	1.375×10^{-1}	1.375×10^{-1}	—	—
		5	1.496×10^{-1}	1.423×10^{-1}	5.452×10^{-1}	5.569×10^{-1}
		10	1.617×10^{-1}	1.509×10^{-1}	3.729×10^{-1}	3.814×10^{-1}
		15	1.799×10^{-1}	1.655×10^{-1}	2.874×10^{-1}	3.095×10^{-1}
		20	1.876×10^{-1}	1.791×10^{-1}	2.475×10^{-1}	2.666×10^{-1}
12	42	0	2.100×10^{-1}	2.100×10^{-1}	—	—
		5	2.202×10^{-1}	2.150×10^{-1}	5.560×10^{-1}	5.668×10^{-1}
		10	2.251×10^{-1}	2.163×10^{-1}	4.055×10^{-1}	4.178×10^{-1}
		15	2.291×10^{-1}	2.191×10^{-1}	3.359×10^{-1}	3.473×10^{-1}
		20	2.330×10^{-1}	2.235×10^{-1}	2.999×10^{-1}	3.099×10^{-1}
20	84	0	2.338×10^{-1}	2.338×10^{-1}	—	—
		5	2.406×10^{-1}	2.350×10^{-1}	5.565×10^{-1}	5.683×10^{-1}
		10	2.424×10^{-1}	2.357×10^{-1}	4.106×10^{-1}	4.318×10^{-1}
		15	2.437×10^{-1}	2.374×10^{-1}	3.451×10^{-1}	3.652×10^{-1}
		20	2.449×10^{-1}	2.380×10^{-1}	3.128×10^{-1}	3.277×10^{-1}
20	110	0	2.697×10^{-1}	2.697×10^{-1}	—	—
		5	2.733×10^{-1}	2.701×10^{-1}	5.678×10^{-1}	5.824×10^{-1}
		10	2.745×10^{-1}	2.716×10^{-1}	4.347×10^{-1}	4.515×10^{-1}
		15	2.755×10^{-1}	2.711×10^{-1}	3.748×10^{-1}	3.886×10^{-1}
		20	2.761×10^{-1}	2.713×10^{-1}	3.440×10^{-1}	3.578×10^{-1}

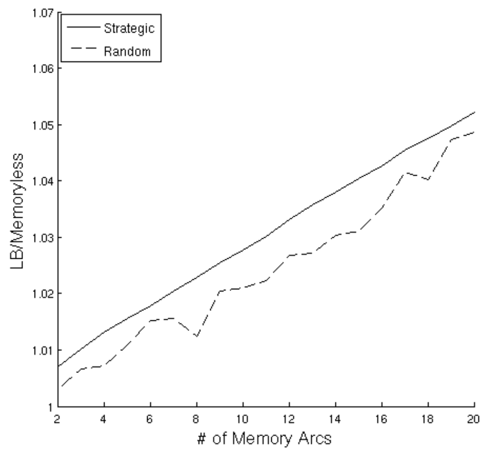
Table 2: Bound comparison using strategic and random selection/clustering methods.

Nodes	Arcs	LB Memory Arcs/ Clusters	LB Strategic	LB Random	UB Strategic	UB Random
8	18	0	≈ 0	≈ 0	—	—
		5	≈ 0	≈ 0	0.03	≈ 0
		10	0.03	0.03	0.06	0.03
		15	1.12	1.13	1.32	1.28
		18	9.59	9.56	11.23	11.11
8	26	0	≈ 0	≈ 0	—	—
		5	≈ 0	≈ 0	0.06	≈ 0
		10	0.04	0.03	0.12	0.05
		15	1.14	1.13	1.45	1.41
		20	40.81	40.80	51.36	51.12
12	30	0	≈ 0	≈ 0	—	—
		5	≈ 0	≈ 0	0.1	≈ 0
		10	0.04	0.04	0.14	0.05
		15	1.38	1.39	1.80	1.69
		20	49.03	48.99	60.93	60.65
12	42	0	≈ 0	≈ 0	—	—
		5	≈ 0	≈ 0	0.18	≈ 0
		10	0.04	0.04	0.25	0.05
		15	1.39	1.39	2.13	1.94
		20	48.99	49.24	66.41	66.31
20	84	0	≈ 0	≈ 0	—	—
		5	0.01	≈ 0	1.08	≈ 0
		10	0.07	0.05	1.22	0.08
		15	1.93	1.91	4.04	2.84
		20	65.74	65.62	101.91	101.13
20	110	0	≈ 0	≈ 0	—	—
		5	0.01	≈ 0	1.87	≈ 0
		10	0.06	0.06	2.06	0.09
		15	1.93	1.89	5.21	3.15
		20	65.77	66.33	114.43	112.27

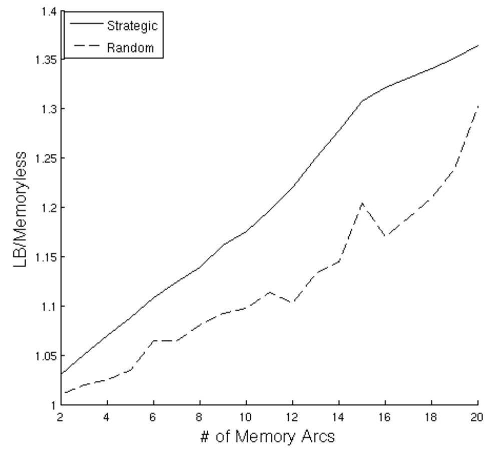
Table 3: Computation times for obtaining lower and upper bounds (in seconds).



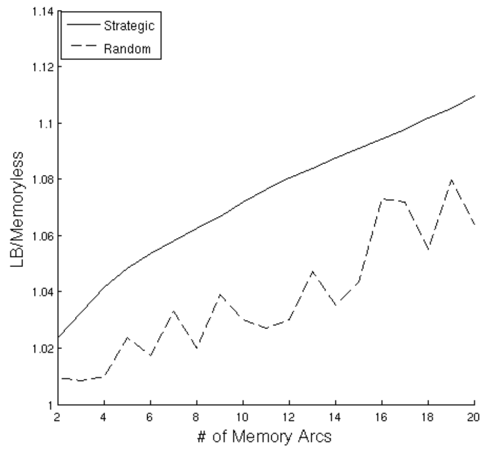
(a) 8 nodes with 18 arcs



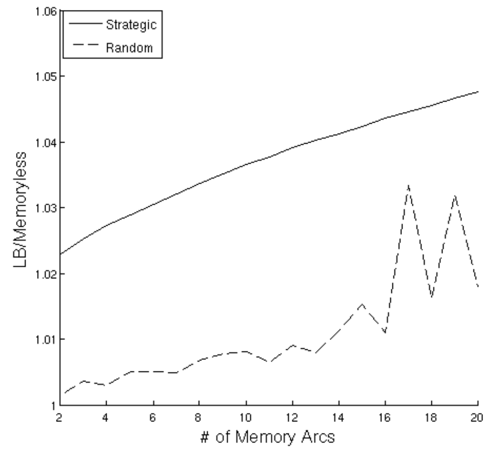
(b) 8 nodes with 26 arcs



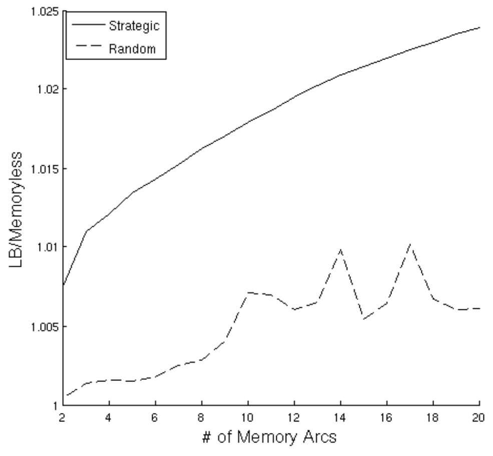
(c) 12 nodes with 30 arcs



(d) 12 nodes with 42 arcs

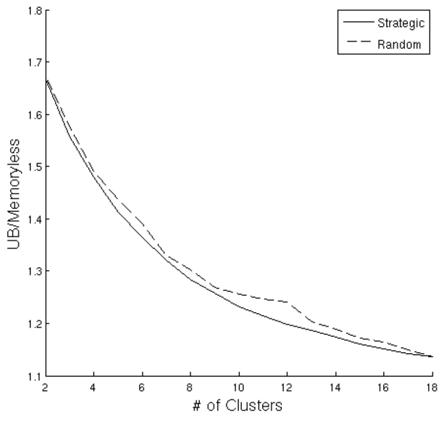


(e) 20 nodes with 84 arcs

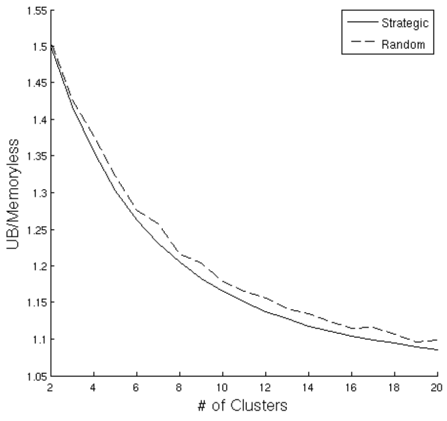


(f) 20 nodes with 110 arcs

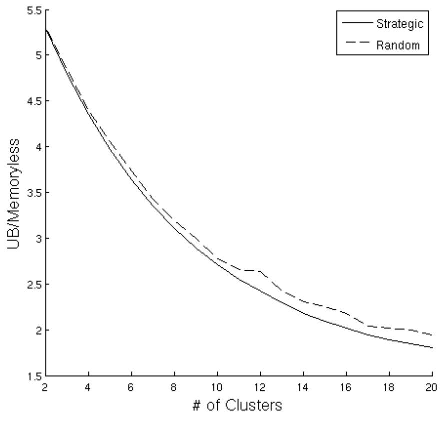
Figure 5: Lower bounds with strategically and randomly chosen arcs.



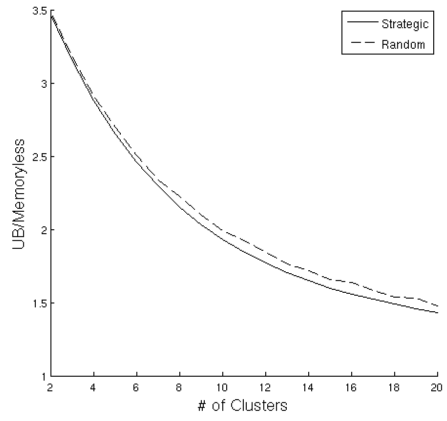
(a) 8 nodes with 18 arcs



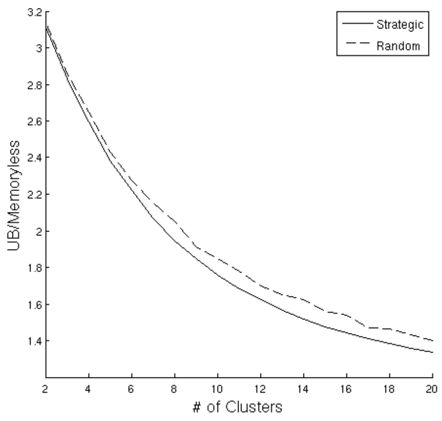
(b) 8 nodes with 26 arcs



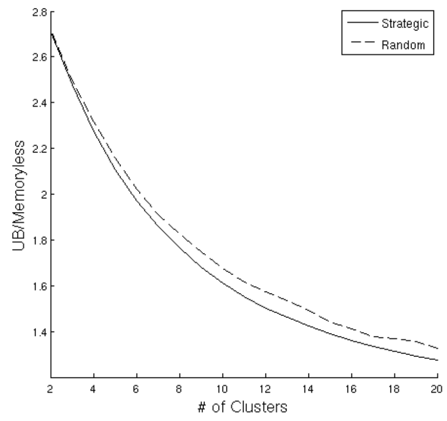
(c) 12 nodes with 30 arcs



(d) 12 nodes with 42 arcs



(e) 20 nodes with 84 arcs



(f) 20 nodes with 110 arcs

Figure 6: Upper bounds with strategically and randomly chosen clusters.

Nodes	Total Arcs	LB Memory Arcs	LB_{str}	$E(LB_{\text{rand}})$	$LB_{\text{str}} - LB_{\text{rand}}$ min	$LB_{\text{str}} - LB_{\text{rand}}$ max
8	22	6	6.508×10^{-1}	6.467×10^{-1}	1.66×10^{-3}	5.78×10^{-3}
		12	6.556×10^{-1}	6.522×10^{-1}	1.41×10^{-3}	5.07×10^{-3}
		18	6.595×10^{-1}	6.576×10^{-1}	6.60×10^{-4}	3.16×10^{-3}
12	34	6	5.154×10^{-1}	5.120×10^{-1}	2.04×10^{-3}	4.36×10^{-3}
		12	5.220×10^{-1}	5.172×10^{-1}	3.20×10^{-3}	5.94×10^{-3}
		18	5.274×10^{-1}	5.226×10^{-1}	2.67×10^{-3}	6.38×10^{-3}
16	44	6	3.899×10^{-1}	3.858×10^{-1}	1.54×10^{-3}	5.59×10^{-3}
		12	3.945×10^{-1}	3.893×10^{-1}	3.77×10^{-3}	6.68×10^{-3}
		18	3.983×10^{-1}	3.930×10^{-1}	3.03×10^{-3}	7.96×10^{-3}

Table 4: Lower-bound comparison for strategic and random methods.

from strategically chosen arcs and clusters with bounds obtained from 20 different instances of randomly chosen sets of arcs and clusters. The results of our experiments are presented in Tables 4 and 5. In these tables, **Total Arcs** refers to the number of arcs with memory in the original network (i.e., those arcs in A that are not of the form (i, n)), and **LB Memory Arcs** refers to the chosen value for k (the number of arcs that have memory in our lower-bound computation). For Table 4, column $\mathbf{LB}_{\text{str}}$ gives the lower bound obtained by the strategic arc selection method, and $\mathbf{E}(\mathbf{LB}_{\text{rand}})$ states the average lower bound obtained by random arc selection. Finally, in the last two columns, $\mathbf{LB}_{\text{str}} - \mathbf{LB}_{\text{rand}}$ states the minimum and maximum difference between the bounds given by the strategic and random strategies over all 20 bounds obtained by the random strategy. The columns in Table 5 are analogously defined. Interestingly, for every instance tested, the strategic bounds (both lower and upper) are better than all 20 bounds provided by their randomized counterparts. These results attest to the importance of cleverly choosing arcs and clusters for use in the given bounding schemes.

So far, the computational results presented on the strategic upper-bounding technique do not employ the improvement method described in Section 3.2.2, which revises the arc reliabilities via the solution of a series of shortest-path problems. In Tables 6–7, we present experiments that quantify how well this technique performs. Table 6 presents the upper-bound values for networks having a varying number of clusters, and Table 7 shows the computation time required to obtain these results. When the number of clusters is chosen to be small, the reduction in upper-bound values is significant, and the computational times do not change significantly. However, as the number of clusters increases, the reduction

Nodes	Total Arcs	Clusters	UB_{str}	$E(UB_{\text{rand}})$	$UB_{\text{rand}} - UB_{\text{str}}$ min	max
8	22	6	6.989×10^{-1}	7.102×10^{-1}	4.21×10^{-3}	2.82×10^{-2}
		12	6.715×10^{-1}	6.788×10^{-1}	3.71×10^{-3}	1.35×10^{-2}
		18	6.638×10^{-1}	6.680×10^{-1}	1.68×10^{-3}	1.03×10^{-2}
12	34	6	6.358×10^{-1}	6.487×10^{-1}	8.31×10^{-3}	2.46×10^{-2}
		12	5.733×10^{-1}	5.885×10^{-1}	1.03×10^{-2}	2.27×10^{-2}
		18	5.537×10^{-1}	5.639×10^{-1}	5.64×10^{-3}	1.81×10^{-2}
16	44	6	5.683×10^{-1}	5.849×10^{-1}	1.16×10^{-2}	2.23×10^{-2}
		12	4.792×10^{-1}	4.945×10^{-1}	1.17×10^{-2}	2.21×10^{-2}
		18	4.461×10^{-1}	4.589×10^{-1}	7.95×10^{-3}	1.74×10^{-2}

Table 5: Upper-bound comparison for strategic and random methods.

methods do not have any effect, which appears to be due to two trends. One, there typically exists a path from any given node to the destination that does not visit any arcs in the same cluster (i.e., the shortest-path costs in the network described in Section 3.2.2 are zero). Two, the calculated upper bound starts to approach the actual survival probability as the number of clusters increases, and hence the maximum possible upper-bound improvement is modest at best.

Also, as expected, the strategic upper-bounding technique nearly always outperforms the random upper-bounding technique, even after the improvement method is applied to both techniques. The only exception occurs when there are two clusters, and neither technique seems to be conclusively better than the other. This behavior is no surprise, given that the bounds are unlikely to be tight in this case regardless of the cluster choice. However, observe from Table 7 that the computational times for the random technique are far less than those corresponding to the strategic technique. An alternative experiment could then randomly generate multiple clusters, compute a bound for each cluster (optionally aided by the improvement method), and retain the best bound resulting from those clusters. We refer to this approach as the multiple-restart random technique. The results of this experiment are summarized in Tables 8 (bound quality) and 9 (computational time). Our multiple-restart experiments were performed by using 100 restarts and recording the computational time and best bounds obtained after 1, 10, 50, and 100 restarts.

These tables show that once again, when only two clusters are allowed, the multiple-restart random technique generally provides better bounds than the strategic technique, when the improvement method is used for both. In particular, only 10 restarts are needed to

Nodes	Total Arcs	Clusters	UB_{str} w/o Improvement	UB_{str} w/ Improvement	UB_{rand} w/o Improvement	UB_{rand} w/ Improvement
8	20	2	7.878×10^{-1}	7.850×10^{-1}	7.888×10^{-1}	7.865×10^{-1}
		4	7.290×10^{-1}	7.290×10^{-1}	7.375×10^{-1}	7.375×10^{-1}
		6	7.006×10^{-1}	7.006×10^{-1}	7.140×10^{-1}	7.139×10^{-1}
		8	6.840×10^{-1}	6.840×10^{-1}	6.955×10^{-1}	6.955×10^{-1}
		10	6.732×10^{-1}	6.732×10^{-1}	6.851×10^{-1}	6.851×10^{-1}
12	32	2	7.836×10^{-1}	7.720×10^{-1}	7.873×10^{-1}	7.873×10^{-1}
		4	7.185×10^{-1}	7.171×10^{-1}	7.257×10^{-1}	7.247×10^{-1}
		6	6.805×10^{-1}	6.805×10^{-1}	6.863×10^{-1}	6.834×10^{-1}
		8	6.552×10^{-1}	6.552×10^{-1}	6.617×10^{-1}	6.617×10^{-1}
		10	6.380×10^{-1}	6.380×10^{-1}	6.513×10^{-1}	6.513×10^{-1}
16	58	2	7.398×10^{-1}	7.295×10^{-1}	7.407×10^{-1}	7.407×10^{-1}
		4	6.364×10^{-1}	6.364×10^{-1}	6.439×10^{-1}	6.439×10^{-1}
		6	5.659×10^{-1}	5.659×10^{-1}	5.818×10^{-1}	5.818×10^{-1}
		8	5.171×10^{-1}	5.171×10^{-1}	5.398×10^{-1}	5.398×10^{-1}
		10	4.853×10^{-1}	4.853×10^{-1}	5.032×10^{-1}	5.032×10^{-1}

Table 6: The effect of shortest path improvement on upper bounds for strategic and random methods on networks of different size.

consistently improve upon the strategic technique, while still using less time than the strategic technique. However, when more than two clusters are used, the bounds obtained by the strategic technique are at least as tight as those obtained by the multiple-restart random technique (where both use the improvement method), even with 100 restarts, with the only tie coming in the 16-node, 58-arc instance with four clusters. Furthermore, the strategic technique with improvement uses less computational time than the 100-restart method, especially as the number of clusters increases. We therefore recommend the strategic technique with improvement when generating upper bounds, using as many clusters as possible within computational limits in order to obtain the best bound possible.

5 Conclusion

In this work, we study the survival probability for random walks on a network. The key feature of the networks we study is that some arcs may fail the first time they are crossed by the walk, but they are assumed to be perfectly reliable each subsequent time they are

Nodes	Total Arcs	Clusters	UB_{str} w/o Improvement	UB_{str} w/ Improvement	UB_{rand} w/o Improvement	UB_{rand} w/ Improvement
8	20	2	0.06	0.06	≈0	≈0
		4	0.03	0.03	≈0	≈0
		6	0.01	0.01	0.01	0.01
		8	0.04	0.05	0.02	0.02
		10	0.10	0.14	0.09	0.10
12	32	2	0.11	0.11	≈0	≈0
		4	0.04	0.05	≈0	0.01
		6	0.05	0.06	≈0	≈0
		8	0.06	0.06	0.01	0.01
		10	0.07	0.09	0.02	0.03
16	58	2	0.38	0.38	≈0	0.01
		4	0.37	0.37	≈0	≈0
		6	0.38	0.4	≈0	≈0
		8	0.4	0.43	0.02	0.02
		10	0.56	0.61	0.06	0.07

Table 7: The effect of shortest path improvement on upper-bound computation times (in seconds).

Nodes	Total Arcs	Clusters	UB_{str} w/o Improvement	UB_{str} w/ Improvement	UB_{rand} w/o Improvement				UB_{rand} w/ Improvement			
					Number of restarts				Number of restarts			
					1	10	50	100	1	10	50	100
8	20	2	7.88	7.85	7.89	7.86	7.85	7.83	7.87	7.73	7.65	7.65
		4	7.29	7.29	7.38	7.34	7.33	7.31	7.38	7.34	7.33	7.31
		6	7.01	7.01	7.14	7.05	7.05	7.05	7.14	7.03	7.03	7.03
		8	6.84	6.84	6.96	6.90	6.90	6.87	6.96	6.90	6.90	6.87
		10	6.73	6.73	6.85	6.76	6.76	6.74	6.85	6.76	6.76	6.74
12	32	2	7.84	7.72	7.87	7.85	7.84	7.84	7.87	7.64	7.63	7.57
		4	7.18	7.17	7.26	7.26	7.24	7.22	7.25	7.25	7.21	7.20
		6	6.80	6.80	6.86	6.86	6.86	6.86	6.83	6.83	6.83	6.83
		8	6.55	6.55	6.62	6.61	6.61	6.61	6.62	6.61	6.60	6.58
		10	6.38	6.38	6.51	6.50	6.44	6.44	6.51	6.48	6.43	6.43
16	58	2	7.40	7.30	7.41	7.41	7.41	7.40	7.41	7.20	7.20	7.15
		4	6.36	6.36	6.44	6.42	6.41	6.41	6.44	6.40	6.40	6.36
		6	5.66	5.66	5.82	5.77	5.74	5.74	5.82	5.77	5.74	5.74
		8	5.17	5.17	5.40	5.32	5.31	5.29	5.40	5.32	5.29	5.29
		10	4.85	4.85	5.03	4.96	4.96	4.95	5.03	4.96	4.96	4.95

Table 8: Bounds obtained from the multiple-restart random technique (all bounds multiplied by 10).

Nodes	Total Arcs	Clusters	UB_{str} w/o Improvement	UB_{str} w/ Improvement	UB_{rand} w/o Improvement				UB_{rand} w/ Improvement			
					Number of restarts				Number of restarts			
					1	10	50	100	1	10	50	100
8	20	2	0.06	0.06	≈0	≈0	0.03	0.08	≈0	0.02	0.09	0.16
		4	0.03	0.03	≈0	≈0	0.01	0.03	≈0	0.03	0.09	0.17
		6	0.01	0.01	0.01	0.02	0.1	0.17	≈0	0.02	0.16	0.31
		8	0.02	0.02	≈0	0.04	0.27	0.51	≈0	0.09	0.53	0.99
		10	0.04	0.05	0.02	0.25	1.29	2.43	0.02	0.39	1.95	3.72
12	32	2	0.11	0.11	≈0	≈0	0.04	0.06	≈0	0.01	0.07	0.14
		4	0.04	0.05	≈0	0.01	0.05	0.08	0.01	0.02	0.14	0.31
		6	0.05	0.06	≈0	0.03	0.07	0.15	≈0	0.05	0.28	0.55
		8	0.06	0.06	0.01	0.06	0.36	0.73	0.01	0.12	0.68	1.26
		10	0.07	0.09	0.02	0.29	1.41	2.76	0.03	0.48	2.28	4.48
16	58	2	0.38	0.38	≈0	0.04	0.19	0.25	0.01	0.04	0.2	0.43
		4	0.37	0.37	≈0	0.01	0.09	0.15	≈0	0.1	0.49	0.97
		6	0.38	0.4	≈0	≈0	0.22	0.38	≈0	0.2	0.83	1.7
		8	0.4	0.43	0.02	0.2	0.85	1.75	0.02	0.32	1.75	3.79
		10	0.56	0.61	0.06	0.64	3.46	6.76	0.07	1.24	5.83	11.5

Table 9: Computational times (in seconds) required to execute the multiple-restart random technique.

crossed. We investigate the computation of the probability that the entity survives the walk and reaches the destination node before failing on any arc. When all arcs are memoryless, i.e., each arc fails with the same probability each time it is crossed, calculating the survival probability is polynomially solvable in a straightforward manner, which requires inverting an $(n-1)\times(n-1)$ matrix. However, when arcs exist that have memory, we prove that calculating survival probability is #P-hard. In particular, this task is at least as hard as calculating the number of Hamiltonian paths in a directed graph. We then propose efficient lower- and upper-bounding techniques to estimate the survival probability. The lower-bounding technique assumes that only a subset of memory arcs actually have memory. The upper-bounding technique clusters arcs so that all arcs in a cluster become fully reliable when one arc in the cluster is traversed. Both bounds converge to the true survival probability as the number of memory arcs or clusters increase, at the expense of an exponential increase in computational effort. Our numerical experiments demonstrate that these bounds are quite effective.

Several research challenges exist that are related to the model studied in this paper. First, we note that within the confines of the problem considered in this paper, our greedy method for selecting memory arcs could possibly be improved by examining more complex interactions among other arcs selected to have memory. (An analogous challenge regards the clustering problem for the upper-bounding method.) Next, an alternative model may involve entities that do not fail on arcs, as assumed in this paper, but will eventually reach an absorbing state. The objective could be to minimize or maximize the amount of time that this entity requires to reach the end state (see [1, 20] for studies in an ecological context). Finally, the examination of models having memory arcs can be studied in many other contexts aside from the one given here, which involves a simple random walk. For instance, instead of considering a purely random walk, it may be interesting to consider an entity that is actively seeking to find the destination node, and chooses each subsequent arc to traverse based on a simple stochastic algorithm related to arc reliabilities and memory as to which arcs have already been traversed. From a network design perspective, there may exist a network owner that seeks to aid an entity using a random walk that wishes to find node n. The network owner may have a budget of k links that can be "revealed" (shown a priori to be reliable or faulty), or perhaps fortified (guaranteed to be reliable). The optimization challenge of determining which k out of m links should be revealed (or, alternatively, fortified) is an interesting challenge that we leave for future research.

Acknowledgments

The authors are grateful for the thorough and insightful remarks of an anonymous referee and an associate editor. Dr. Büke gratefully acknowledges the support of Engineering and Physical Research Council under grant EP/I017127/1 (Mathematics for Vast Digital Resources). Dr. Smith gratefully acknowledge the support of the National Science Foundation under grant CMMI-1100765, the Defense Threat Reduction Agency under grant HDTRA-10-01-0050, the Air Force Office of Scientific Research under grant FA9550-12-1-0353, and the Office of Naval Research under grant N000141310036.

References

[1] M.A. Acevedo, J.A. Sefair, J.C. Smith, and R.J. Fletcher, Jr., Conservation with uncertainty: Identifying protection strategies under worst-case disturbance events, Technical report, University of Florida, School of Natural Resources and Environment, Gainesville, FL, 2014.

[2] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, Network flows: Theory, algorithms, and applications, Prentice Hall, Englewood Cliffs, NJ, 1993.

[3] D. Aldous and J. Fill, Reversible Markov chains and random walks on graphs, webpage. http://stat-www.berkeley.edu/pub/users/aldous/RWG/book.html (last checked 14 January 2014).

[4] B. Bollobás, Modern graph theory Vol. 184 of *Graduate Texts in Mathematics*, Springer, New York, 1998.

[5] M. Brand, A random walks perspective on maximizing satisfaction and profit, SIAM Int Conference Data Mining, Newport Beach, CA, SIAM, April 2005, pp. 12–19.

[6] S. Brin and L. Page, The anatomy of a large-scale hypertextual Web search engine, Comput Networks ISDN Syst 30 (1998), 107–117.

[7] C.J. Colbourn, The combinatorics of network reliability, Oxford University Press, New York, NY, 1987.

[8] A. Das Sarma, A.R. Molla, G. Pandurangan, and E. Upfal, Fast distributed PageRank computation, Theoret Comput Sci 561(B) (2015), 113–121.

[9] P.G. Doyle and J.L. Snell, Random walks and electrical networks, Mathematical Association of America, Washington, DC, 1984.

[10] B.D. Ewald, J. Humpherys, and J.M. West, Computing expected transition events in reducible Markov chains, SIAM J Matrix Anal Appl 31 (2009), 1040–1054.

[11] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, W. H. Freeman & Co., Princeton, NJ, 1979.

[12] J.S. Lin, C.C. Jane, and J. Yuan, On reliability evaluation of a capacitated flow network in terms of minimal path sets, Networks 25 (1995), 131–138.

[13] Y.K. Lin, A simple algorithm for reliability evaluation of a stochastic-flow network with node failure, Comput Oper Res 28 (2001), 1277–1285.

[14] Y.K. Lin and J. Yuan, A new algorithm to generate d-minimal paths in a multistate flow network with non-integer arc capacities, Int J Reliab, Qual Saf Eng 5 (1998), 269–285.

[15] L. Lovász, "Random walks on graphs: A survey," Combinatorics, Paul Erdös is eighty, D. Miklós, V.T. Sós, and T. Szönyi (Editors), Janos Bolyai Society Mathematical Studies, Budapest, Hungary, 1996, Vol. 2, pp. 353–397.

[16] K.S. Miller, On the inverse of the sum of matrices, Math Magazine 54 (1981), 67–72.

[17] M.E.J. Newman, A measure of betweenness centrality based on random walks, Social Networks 27 (2005), 39–54.

[18] Z. Peixin and Z. Xin, A survey on reliability evaluation of stochastic-flow networks in terms of minimal paths, Int Conference Informat Eng Comput Sci 2009 (ICIECS2009), December 2009, pp. 1–4.

[19] S.M. Ross, Introduction to probability models, Academic Press, Inc., Boston, MA, 2010.

[20] J.A. Sefair, J.C. Smith, M.A. Acevedo, and R.J. Fletcher, Jr., A three-stage model and algorithm for maximizing weighted expected hitting time with application to conservation planning, Technical report, University of Florida, Department of Industrial and Systems Engineering, Gainesville, FL, 2014.

[21] D.R. Shier, Network reliability and algebraic structures, Oxford University Press, Oxford, NY, 1991.

[22] L.G. Valiant, The complexity of enumeration and reliability problems, SIAM J Comput 8 (1979), 410–421.

[23] W.C. Yeh, A revised layered-network algorithm to search for all d-minpaths of a limited-flow acyclic network, IEEE Trans Reliab 47 (1998), 436–442.