University of Arkansas, Fayetteville
ScholarWorks@UARK

Graduate Theses and Dissertations

12-2022

Using Reinforcement Learning to Improve Network Reliability through Optimal Resource Allocation

Henley Wells
University of Arkansas, Fayetteville

Follow this and additional works at: https://scholarworks.uark.edu/etd

 Part of the Industrial Engineering Commons, Industrial Technology Commons, and the Systems Engineering Commons

Citation

Wells, H. (2022). Using Reinforcement Learning to Improve Network Reliability through Optimal Resource Allocation. *Graduate Theses and Dissertations* Retrieved from https://scholarworks.uark.edu/etd/4714

This Thesis is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu, uarepos@uark.edu.

Using Reinforcement Learning to Improve Network Reliability through Optimal Resource Allocation

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Industrial Engineering

by

Henley Wells
University of Arkansas
Bachelor of Science in Industrial Engineering, 2020

December 2022
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

_____

Edward A. Pohl, Ph.D.
Committee Chair

_____

Haitao Liao, Ph.D.
Committee Member

_____

Kelly Sullivan, Ph.D.
Committee Member

Abstract

Networks provide a variety of critical services to society (e.g. power grid, telecommunication, water, transportation) but are prone to disruption. With this motivation, we study a sequential decision problem in which an initial network is improved over time (e.g., by adding or increasing the reliability of edges) and rewards are gained over time as a function of the network's all-terminal reliability. The actions during each time period are limited due to availability of resources such as time, money, or labor. To solve this problem, we utilized a Deep Reinforcement Learning (DRL) approach implemented within OpenAI-Gym using Stable Baselines. A Proximal Policy Optimization (PPO) was used to identify the edge to be improved or a new edge to be added based on the current state of the network and the available budget. To calculate the network's all-terminal reliability, a reliability polynomial was employed. To understand how the model behaves under a variety of conditions, we explored numerous network configurations with different initial link reliability, added link reliability, number of nodes, and budget structures. We conclude with a discussion of insights gained from our set of designed experiments.

Table of Contents

Introduction

Networks are used in all facets of our lives. Ensuring networks operate properly is critical so that everyone can go about their daily activities such as driving on a network of roads, utilizing a communication network, and using appliances in their house which are supplied by the power network. These are just a few of the infrastructure networks on which everyone depends. If they are not reliable, many people will suffer the consequences.

Historically, many failures in infrastructure have occurred which have caused issues for many people. One well-known example is the 2003 Northeast blackout. According to History.com [13], This blackout was caused by a software issue and affected fifty million people in the United States and Canada. Other failures in infrastructure networks include the nuclear disaster at Three Mile Island in Pennsylvania and the levee failure in Louisiana during Hurricane Katrina. The Three Mile Island disaster was caused by a failure in one of the components in the plant and it caused the rest of the system to shut down. According to the U.S. NRC [44], this component failure caused a partial nuclear meltdown which affected thousands of people in the area surrounding the plant. According to Pruitt [34], the levees in Louisiana were not adequately prepared to handle the water from Hurricane Katrina, thus, they breeched due to the pressure and caused much of New Orleans to flood. These are just a few examples of the many infrastructure failures in the United States. Infrastructure failures are very serious to the people affected as they can cause serious injury or death in some cases. These examples show us how essential it is to ensure infrastructure networks are reliable.

A network in its simplest form is a collection of nodes which are connected by edges. Many of the everyday items we use are connected in a network. Different network types according to Newman [32] are detailed in Table 1.

Network Type	Description	Examples
Technological	The physical infrastructure networks that form the backbone of our society	Internet (physical connections), infrastructure networks (i.e. transportation, telephones, power grid)
Information	Networks of data linked together	World Wide Web (web pages are nodes and links are edges)
Social	A network of people connected through modes such as friendship	Social Media Platforms
Biological	Networks occurring naturally in biology	Food webs and neural networks

Table 1: Types of Networks

Networks can either be directed or undirected. Directed networks have a specified direction of travel. They only allow one-way travel along the links. Undirected networks do not have the one-way travel restriction directed networks have. Examples of undirected networks include social networks and most transportation systems such as highways. Directed networks include the World Wide Web and food webs or a system of food chains. Figures 1 shows an undirected network, so information can travel in any direction along the links. Figure 2 shows the exact same network structure, but the links are directed, so the flow is limited to the direction of the arrows.

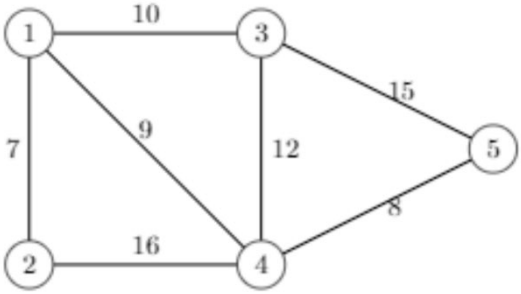


Figure 1: An undirected network

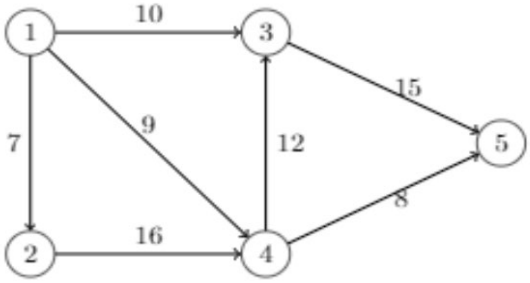


Figure 2: A directed network

Networks are widely used to model infrastructure systems to help make them easier to understand. Milanovic & Zhu [31] use complex network theory to model the cyber-physical network. Goldbeck *et al.* [16] modeled the optimal response to disruptions in infrastructure systems by simulating the network and its assets. Because of the complexity of infrastructure systems, modeling them using networks allows them to be explained more easily. This is especially important when trying to calculate system reliability. The network reliability is the probability a network is performing its intended function at a given point in time. Hui [19] describes two-terminal reliability, the most basic measure of network reliability, as the probability of having at least one operational path between two nodes. In any system where failures occur, studying network reliability is useful and sometimes vital to guarantee safety is the top priority. Ensuring a network will perform its intended function is critically important for most systems especially infrastructure networks which control the everyday functioning of our lives.

Network reliability has been studied in detail for a variety of applications and using many different techniques. Network reliability was initially studied on a small scale using simpler

techniques which did not present much of a challenge computationally. As the networks being studied became more complex, more sophisticated techniques were developed to handle the more computationally intensive networks. Each of the techniques used provide either the exact reliability or an approximation. Some of the most popular techniques used to study network reliability are Monte Carlo Simulation (MCS), complete or partial enumeration of path sets and cut sets, artificial neural networks, and bounding the reliability of the system. These techniques will be discussed in more detail in the literature review.

Calculating the all-terminal reliability of a basic network is relatively simple for a series-parallel network such as the one in Figure 3. However, it becomes much more complicated for other types of networks and when more nodes and links are added. Figure 3 shows a basic network with four nodes and five links. For each of the links, a reliability is shown. To calculate the all-terminal reliability, we calculate the probabilities of the network for all network states where all nodes remain connected. For example, the probability of the system for the state where all edges are working is 0.85*0.8*0.95*0.9*0.75 = 0.43605. Table 2 below has the rest of the calculations. When we sum all the probabilities, we get that the all-terminal network reliability is 0.9414.

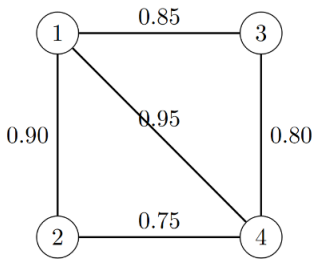


Figure 3: An example of a two-terminal network

Configuration	Probability
All working	0.43605
1-3 failed	0.07695
3-4 failed	0.109013
1-4 failed	0.02295
1-2 failed	0.04845
2-4 failed	0.14535
1-3, 1-4 failed	0.00405
1-3, 1-2 failed	0.00855
1-3, 2-4 failed	0.02565
3-4, 1-4 failed	0.005738
3-4, 1-2 failed	0.012113
3-4, 2-4 failed	0.036338
1-4, 1-2 failed	0.00255
1-4, 2-4 failed	0.00765

Table 2: Reliability Calculations

One of the biggest issues when analyzing network reliability is the vastness of the networks. As the size of networks grows, analyzing them efficiently becomes more difficult. Existing techniques do not possess the capabilities to compute network reliability in a timely manner. Previous research has also examined the difficulty of synthesizing reliable networks. This problem is more complex as it requires repeated evaluation of network reliability.

Motivated by this limitation, machine learning techniques have been developed for the purpose of optimizing network reliability. One machine learning approach which may be able to help solve this problem is reinforcement learning (RL). Reinforcement learning makes decisions based on a rewards system for taking certain actions. The actions we will take aim to improve the reliability of the network over time. This improvement over time is known as reliability growth. Studying the effects of certain actions on the reliability growth of the network will be an important aspect of our research.

Research Motivation

The objective of this research is to apply reinforcement learning to optimize the allocation of resources to improve the reliability of infrastructure networks. Financial constraints are a challenge for every network, but they are especially prevalent when designing infrastructure networks. The limitations these resource constraints present makes the process of designing reliable networks more difficult. In our research, analogous to a reliability growth paradigm, we develop an approach to improve the network reliability of an infrastructure system and evaluate the optimal way to allocate the resources available to maximize the all-terminal reliability of the network. Adding the challenge of optimizing our resource allocation makes an already challenging reliability evaluation problem even more difficult. By undertaking this challenge, our research aims to optimize the available resources to make the networks more reliable in the future.

Literature Review

A literature review was performed to explore previous research in the areas of network reliability, machine learning, and reliability growth. The literature review highlights different approaches for finding network reliability, details the numerous difficulties for solving these problems, and hones in on a specific approach which appropriately fits our research problem.

1. Network Reliability

To establish the type of network we are interested in exploring, we can classify networks into different groups. Ball [4] discusses network reliability models and their varying levels of complexity. The complexity depends heavily on the connectivity of the network. There are three main types of network reliability problems: 2-terminal, k-terminal, and all-terminal. The 2-terminal and all-terminal cases are special cases of the k-terminal problem. The k-terminal reliability problem has k terminal nodes and a root node, s. In general, the reliability of a k-terminal system is the probability that the root node is connected to every other terminal node. For 2-terminal reliability, the system has only two nodes and the reliability of the system is the probability they are connected. All-terminal simply means every node is connected to every other node. The reliability is the probability the system is fully connected. Calculating the reliability of the system becomes computationally challenging the more nodes and links are added to the network, making all-terminal reliability problems the hardest.

Provan & Ball [33] found the all-terminal network reliability calculation to be a #P-complete problem meaning the computational difficulty grows exponentially when more nodes and links are added to the network. To help analyze this problem, numerous methods for calculating network reliability have been used. These methods provide either an exact value or an estimate of the reliability. Examples of these methods are artificial neural networks,

estimating bounds, Monte Carlo simulation, optimization, time exponential and polynomial time algorithms, and state enumeration. The following discussion explains how most of these methods work and explores previous research on these methods to characterize their capabilities and limitations.

In their survey of network reliability methods, Ball *et al.* [5] summarizes exact methods for calculating network reliability such as exponential time exact algorithms for general networks and polynomial time exact algorithms for restricted classes of networks, as well as other methods such as bounds on network reliability, and Monte Carlo simulation. Gaur *et al.* [15] also detailed many different network reliability methods including state enumeration, minimum cut enumeration, and neural networks, and they discussed the limitations involved with each method.

According to Su *et al.* [42], a minimum cut set is a set of system components which, when failed, causes failure of the system. Minimum cut sets do not contain any other subsets of cuts which would cause the system to fail. Minimum cut enumeration methods compute network reliability by enumerating the reliability of all minimum cut sets and using these individual reliabilities to calculate the reliability of the network. Cut enumeration is an exact method for calculating network reliability. It is very useful for small networks, but it reaches its limitations very quickly. According to Gaur *et al.* [15], cut enumeration is most effective for two-terminal reliability problems. However, as the network grows, the number of cut sets grows exponentially. This makes calculating all possible combinations for two-, k-, and all-terminal reliability calculations time consuming.

Monte Carlo simulation (MCS) methods choose a random sample of states to explore and estimate the network reliability as the proportion of sampled states in which the network is functioning properly. Karger [25] utilized MCS to simulate edge failures by checking to see if

the randomly selected edge caused the network to fail. He found one of the flaws of the MCS approach to be that it is very slow when the probability of failure is very low. Cardoso *et al.* [8] studied Monte Carlo simulation in conjunction with neural networks to investigate the structural reliability of different structures. MCS only allows one network structure to be calculated at a time, so it can be very time consuming to calculate the reliability. To combat this issue, they combined neural networks with MCS which allowed them to save computation time and obtain more precise reliability measurements.

Artificial neural networks (ANNs) are based on the biological neural networks within the human body. Just like the brain, the components of ANNs work together in parallel and series to learn based on experiences. This learning occurs using a training set which is a set of inputs with known, target outputs. According to Jain & Mao [21], a few of the main uses of ANNs are pattern recognition, prediction, optimization, associative memory, and control.

Srivaree-ratana *et al.* [41] used an artificial neural network to estimate network reliability. In their study, they trained the ANN using a set of network topologies and link reliabilities. They then used the ANN to estimate the network reliability based on the link reliabilities and the topology to find the optimal network topology. They finally used the chosen topologies and calculated the exact reliability for each of them. They demonstrate that their estimation performs well empirically through comparisons to an exact approach, which is computationally intensive, as well as to an upper bound derived from a polynomial time algorithm.

An alternative approach to exact and approximate reliability calculations is to find bounds for the network reliability. The method of finding bounds is advantageous because it is not as computationally intensive as the other methods, but it is also not as accurate since it only provides bounds rather than a specific reliability. Sebastio *et al.* [37] created an algorithm to find

bounds for a two-terminal network reliability problem. In their algorithm, the user specifies the execution time. In their algorithm, they analyze minpaths and mincuts. A minpath is the path between nodes wherein if one of the links was removed, the network would not be connected anymore. Their algorithm searches the network to find the most important minimum paths and minimum cuts that reduce the reliability between the upper and lower bounds. Satitsatian & Kapur [36] discussed a different use for bounds. They found a lower bound for network reliability to compute the exact reliability and reliability bounds. They created an algorithm to find a subset of lower boundary points to help them obtain the lower reliability bound with less computational effort.

Ramirez-Marquez & Rocco [35] presented a new algorithm for solving all-terminal network reliability allocation problems (RAP). For their problem, their goal was to minimize cost for the network subject to a reliability constraint. They created an algorithm that had three steps which are: generate network configurations, obtain the reliability for each of them using MCS and penalize the networks if they do not meet the reliability constraint, and rank the networks from highest to lowest. They found solutions with their algorithm that were on at worst 7% lower cost and at best 21% lower cost than previous solutions from the literature. Yeh *et al.* [46] also proposed a method for allocating resources using Monte Carlo Simulation. Their method proposed the use of a particle swarm optimization (MCS-PSO). They aimed to minimize the cost of components while meeting the reliability constraints. They found their method to be more efficient and better at approximating reliability than MCS alone.

Others have presented different approaches for synthesizing reliable networks. Mettas [30] studied the reliability allocation problem at the component level for general systems. His work estimated the minimum reliability for multiple components to meet the minimum system

reliability. Both Jan *et al.* [22] and AboElFotoh & Al-Sumait [1] utilized methods to study network topology. They sought to find the optimal topological layout of links which minimized cost and met the minimum network reliability requirement. Jan *et al.* [22] utilized a decomposition method based on a branch and bound method. Their method divides the network into subproblems based on link number which could be solved by their branch and bound algorithm. AboElFotoh & Al-Sumait [1] solved the same problem using an ANN.

2. Reinforcement Learning

Machine learning has numerous different approaches which are classified based on how the algorithm learns. One of these approaches is reinforcement learning. Zhang [47] defines reinforcement learning as a type of machine learning that uses a training data set that is comprised of rewards and punishments which is given as feedback to the machine. The machine uses this feedback to improve the performance of the task. The reinforcement learning algorithm randomly chooses an action, and the value of the action is calculated. The value of the actions stems from an immediate reward as well as the value of ending up in a different state. By repeatedly applying this process, reinforcement learning aims to learn the value of optimal state/action combinations thereby maximizing the total reward gained.

Reinforcement learning (RL) has many different applications. One of the first applications for RL was in video games. In many video games, the player makes decisions for the character. RL treats the decisions they make as the actions. Using RL, researchers were able to explore many different scenarios and allow the algorithm to learn what the best actions are for a given state. Lin *et al.* [27] approached the video game problem for two different games, Flappy Bird and Breakout. They trained both games using reinforcement Q-learning with a neural

network and without a neural network. They found the time it took to completely train the model was much faster when using a neural network than without it.

There are other applications of reinforcement learning that involve networks. Yang *et al.* [45] created a deep reinforcement learning model to study strategies for allocating resources in a computing network. Their goal was to maximize the reliability of the system from end-to-end. They employed a Q-learning algorithm to help the system decide where to allocate resources to prevent the quality of the channels in the system from being below their set standards. Their study found the Q-learning system to be successful after being trained for approximately 100 trials.

Gottesman *et al.* [17] studied the use of Artificial Intelligence including RL in healthcare systems. Decisions regarding when to do certain tasks in a hospital setting are critically important to the well-being of patients. RL studies the results of the decisions which are made and after being trained, it can help healthcare professionals determine the optimal treatment plan for a patient given the initial state of the patient. RL in healthcare has been used to optimize the sequence of care for patients.

3. Reliability Growth

Reliability growth is a technique utilized to improve a systems reliability during its design, development, and operation. The basic approach is to test a system, identify failures, and then make design changes to reduce the likelihood of those failures occurring again in the future, and hence improving the overall reliability of the system. Reliability growth modeling is used to improve in the reliability of the system as a result of a design changes over time. Duane [12] was one of the earliest researchers to study reliability growth. He found electromechanical and mechanical systems to have similar rates of reliability improvement during system development.

His work focused on studying the learning curve for the systems to predict reliability at certain points in time. He found the logarithm of cumulative failure rate to have close to a linear relationship with the logarithm of cumulative operating hours. Crow [10] later studied reliability as it pertains to system age. In his study, he proposed the Army Material Systems Analysis Activity (AMSAA) model. According to Kurtz *et al.* [26], "The AMSAA Reliability Growth Guide summarized the benefits of reliability growth management in finding unforeseen deficiencies, designing improvements, reducing risk, and increasing the probability of meeting objectives." He proposed a nonhomogeneous Poisson process model with a Weibull intensity function to study the age-dependent reliability.

According to Cahoon *et al.* [7], for a general system, there are three ways reliability growth models are used. These are planning the system's reliability improvement, tracking the improvement, and ensuring the project is on-track to achieve its goals. Another practical use for reliability growth modeling is in Department of Defense (DoD) system testing. The DoD uses two types of reliability growth models which are system-level models which use nonhomogeneous Poisson processes (NHPPs) and competing risk models which analyze multiple failure modes operating in series. NHPP models allow for keeping track of the number of failures over time and the time between failures. The competing risk models study the system as a set of components operating in series. This means all components must be working for the system to operate.

There are two main types of reliability growth models: discrete and continuous. Continuous reliability growth models use continuous data such as time. Some examples of continuous growth models are [10, 12, 23, and 43]. Discrete reliability growth models use countable data [e.g. 14, 38, 40].

Reliability growth modeling has been studied in-depth for software development [3, 9, 24] and hardware testing [2, 11, 28]. In software development, reliability growth helps predict and assess product quality, release time, and the cost of testing and debugging. Cinque *et al.* [9] discussed how reliability growth can be used to maximize the efficiency of the improvement in reliability of the system following debugging. Hussin *et al.* [20] provided insight into how reinforcement learning can be used to improve reliability of a distributed system which has both hardware and software components. They developed a RL-based resource management approach to improve network reliability by making design changes. They concluded RL is an effective agent for optimizing reliability growth in large systems.

Infrastructure networks provide numerous opportunities to analyze reliability and make design changes to improve reliability over time. Networks such as telecommunication systems, transit systems, and energy systems are faced with problems which can be studied and improved using machine learning techniques such as RL as well as following the reliability growth paradigm for modeling design improvements to the system over time. Mahmoodzadeh *et al*. [29] examined gas pipelines and the issues they face. Their goal was to minimize cost of the maintenance they had to perform. Rather than performing periodically scheduled maintenance, they utilized RL to analyze when maintenance should actually be performed based on the condition of the pipeline. Serrano [39] discusses the advancement of infrastructure and how all types of AI, specifically RL, can be used to help make infrastructure more intelligent. Intelligent infrastructure is a developing technology which adds sensors, an ability to communicate information, and analysis capabilities to structures such as bridges and buildings. It allows the infrastructure to monitor, protect and repair itself. Serrano [39] used RL to help the sensors in an

intelligent infrastructure network make predictions about the future needs of the system based on the current conditions it is experiencing.

Acevedo *et al.* [2] studied reliability growth in telecommunication systems. They specifically utilized accelerated life testing (ALT) to evaluate critical hardware in these systems to find flaws early in the development cycle. Dempsey & Sheng [11] studied wind turbines and their premature component failures. They also tested component prototypes at a test facility to find flaws. Additionally, they developed a condition-monitoring system to detect damage to components. Similarly, Kurtz *et al.* [26] studied reliability issues at hydrogen stations. They utilized prognostics and diagnostics that have been used in wind turbines to improve station availability and decrease operating and maintenance costs.

Properly allocating available resources over time to improve system reliability is important for the success of networks. As the technology has become more intelligent, the size of networks which are being studied has grown significantly. This growth highlights the importance of having intelligent software which can keep up with the network size. Machine learning, specifically reinforcement learning, has been proven to be an effective means for optimizing reliability growth in large systems. This thesis addresses the need to continuously improve network reliability by utilizing a reliability growth paradigm by allocating resources to make design changes to the network to improve system reliability, emphasizing the impact resource allocation will have on an infrastructure network over a period of time.

Methodology

Reinforcement learning (RL) has been used to solve many different problems in a variety of areas. Its sophistication and ability to quickly solve complex problems makes it a good option to help solve a complex network reliability synthesis problem such as the one we are studying. As stated above, RL has been used to optimize performance in video games, communication systems, and healthcare. However, there appeared to be an opportunity to use RL to improve the reliability of complex networks.

We will explore the use of RL to assist us in determining the appropriate resource allocation strategy in order to achieve a specific reliability improvement goal. Every year, infrastructure systems are allocated a certain number of resources such as their budget and labor. For these resources, decisions must be made to decide how to allocate their resources to help improve the system. For our research, we will modify an existing reinforcement learning model built in Python to accommodate the specific complexities and tradeoffs associated with our problem of interest. The specific details of this model are explained below.

There are two possible ways the network can be made more reliable and they are adding links or dedicating resources to enhance the reliability of existing links. The existing model focused solely on exploring the reliability improvement when adding links.

Reliability improvement through optimal resource allocation is the focus of this research. We will make trade-offs between the different possible actions with the goal of maximizing reliability. Specifically, we will evaluate the network reliability in every state and make decisions based on the available budget. This approach follows the idea of sequential decision-making models where the decisions made in one state may affect decisions and rewards in the

future states. To calculate the reliability of different states, a reliability polynomial was employed. This polynomial is constructed by estimating the coefficients.

For our problem, we will be using a network where the topology changes only if a new edge is added. However, the reliability of the links will increase as resources are allocated to them.

General Model Formulation

Our problem considers an initial network with n nodes and a given set of $n-1$ edges. We consider a sequential decision problem with m time periods, where in each time period $t = 1,2,\dots,m$ we can make limited investments to add potential edges from the set $E = \{\{i,j\}: i = 1,2,\dots,n-1; j = i+1,2,\dots,n\}$ to the network and/or upgrade the reliability of existing edges. An edge $\{i,j\} \in E$ that has been added to the network and improved z_{ij} times is assumed to have reliability $k_{ij} + z_{ij}l_{ij}$, where k_{ij} and l_{ij} are parameters.

Ultimately, we wish to maximize the cumulative discounted reward obtained over time periods $t = 0,1,\dots,m-1$, where the reward obtained in time period t is a function of the network's all-terminal reliability immediately after the time period. A fixed budget B_t is made available at the beginning of each time period $t = 0,1,\dots,m-1$ and can either be used for either immediate actions or carried forward to use in future periods. Parameters c_{ij} and p_{ij} respectively describe the cost to add an edge $\{i,j\} \in E$ and to improve an existing edge $\{i,j\} \in E$.

The state of the network s before any time period is defined by the tuple

$$s = (t, R, \beta),$$

where $t \in \{0,1,\dots,m-1\}$ is the number of time periods already completed, R is an $|E|$-vector specifying the reliability of each edge in the network, and β is the remaining budget. Let the

elements of R be denoted by $r_{ij}, \{i,j\} \in E$, where $r_{ij} = 0$ if edge $\{i,j\}$ has not been added to the network.

In state $s = (t, R, \beta)$, an action is defined by $a = (X, Y)$ where X and Y are $|E|$-vectors. The vector X consists of elements $x_{ij}, \{i,j\} \in E$, where $x_{ij} = 1$ if edge $\{i,j\}$ is added to the network; 0 otherwise. The vector Y consists of elements $y_{ij}, \{i,j\} \in E$, where $y_{ij} = 1$ if an existing edge $\{i,j\}$ is improved; 0 otherwise. The action with $x_{ij} = y_{ij} = 0$ for all $\{i,j\} \in E$ represents deciding to move to the next time period without adding or improving any additional edges. The set of feasible actions in state $s = (t, R, \beta)$, are defined by the equations:

$$\sum_{\{i,j\}\in E}(x_{ij} + y_{ij}) \leq 1 \quad (1)$$

$$\sum_{\{i,j\}\in E}(c_{ij}x_{ij} + p_{ij}y_{ij}) \leq \beta \quad (2)$$

$$x_{ij} = 0, \forall \{i,j\} \in E: r_{ij} > 0 \quad (3)$$

$$y_{ij} = 0, \forall \{i,j\} \in E: r_{ij} = 0 \quad (4)$$

$$r_{ij} + k_{ij}x_{ij} + l_{ij}y_{ij} \leq 1, \ \forall \{i,j\} \in E \quad (5)$$

Equation (1) limits the number of actions we can take during a time period to either zero or one, and Equation (2) requires the actions that are taken to be less than or equal to the remaining budget for the period. Note that any unused budget in time period t may be carried forward for use in future time periods; therefore, it may be desirable to choose to move to the next period even when adequate resources are available to perform one of the other actions. Equation (3) requires $r_{ij} = 0$ for action $x_{ij} = 1$ to be feasible, thus specifying that an edge $\{i,j\} \in E$ cannot be added if it is already in the network. Equation (4) requires $r_{ij} > 0$ for action $y_{ij} = 1$ to be feasible, imposing that an edge $\{i,j\} \in E$ can only be improved if it was previously added to the network. The feasible actions take one of three forms: improving an edge (i.e., $y_{ij} = 1$ for some

$\{i,j\} \in E$), adding an edge (i.e., $x_{ij} = 1$ for some $\{i,j\} \in E$), or choosing to move into the next period (i.e., $x_{ij} = y_{ij} = 0\ \forall \{i,j\} \in E$). Note, taking an action with $x_{ij} = 1$ or $y_{ij} = 1$ for some $\{i,j\} \in E$ does not indicate we will be moving into a new time period, only that the state variables R and β have changed. Equation (5) ensures an edge cannot be upgraded to a reliability value greater than 1.

We now define the state transition function, $(t', R', \beta') = g(s,a)$ given an action $a = (X,Y)$ performed in state $s = (t,R,\beta)$. If $x_{ij} = y_{ij} = 0\ \forall \{i,j\} \in E$, the new state is defined by

$$t' = t + 1, \tag{6}$$

$$R' = R, \tag{7}$$

$$\beta' = \beta + B_{t+1}. \tag{8}$$

Equation (6) states the time period, t', after a state transition is one period later than the previous time period, t. Equation (7) states that the reliability of the network remains the same during a state transition. Equation (8) states the budget in the new state after the transition is a function of the remaining budget from the previous state plus the fixed budget for the new period. If $x_{ij} = 1$ or $y_{ij} = 1$ for some $\{i,j\} \in E$, the new state is defined by

$$t' = t \tag{9}$$

$$r_{ij}' = r_{ij} + k_{ij}x_{ij} + l_{ij}y_{ij}, \forall \{i,j\} \in E, \tag{10}$$

$$\beta' = \beta - \textstyle\sum_{\{i,j\} \in E} \left(c_{ij}x_{ij} + p_{ij}y_{ij}\right). \tag{11}$$

Here, Equation (9) represents the time period of the network remaining the same while actions are being taken. Equation (10) represents the updated reliability, r_{ij}', of each link after an action has been taken. It states the updated link reliability, r_{ij}', is equal to the current link reliability, r_{ij}, plus any additional reliability that is added for an action (x_{ij} or y_{ij}). Equation (11) states the new

budget, β', after an action is taken, is equal to the current budget, β, less the cost of the action taken.

For a state $s = (t, R, \beta)$ and action $a = (X, Y)$ that results in a transition to state $(t', R', \beta') = g(s, a)$, we now define the reward function $f(s, a)$. The rewards we receive are a function of the action we take and the previous state of the network. The new edge reliabilities, R', determine the new all-terminal reliability which determines the rewards we receive. For our problem the reward is received immediately after transitioning into a new state. Taking action $a = (X, Y)$ in state $s = (t, R, \beta)$ and then transitioning into state $g(s, a) = (t', R', \beta')$ results in an immediate reward of $f(R')$, where $f(R')$ is defined as the current state we are in and value of the action we just took. The value of the $f(R')$ is determined by the improvement in the all-terminal reliability with respect to edge reliabilities, R'. These rewards are awarded after every state change rather than in-between periods. Note, there may be multiple state changes that take place within the same time period.

Under a given *policy* for selecting actions in each state, let V_t denote the reward accumulated during time period $t = 0,1, \dots, m-1$. We seek to identify a policy that maximizes the discounted reward function, $G = \sum_{t=0}^{m-1} \gamma^t V_t$, where γ ($0 < \gamma \leq 1$) is a discount factor parameter that specifies how much we care about immediate rewards versus future rewards. A value closer 0 means we value immediate time periods more, while a value of 1 means we value immediate and future time periods equally.

The initial conditions for the network when $t = 0$ are defined by the network consisting of n nodes and $n-1$ edges with equivalent initial reliabilities, R. An episode of the network terminates immediately after leaving a state in which (i) the only feasible action is $x_{ij} = y_{ij} = 0 \; \forall \{i, j\} \in E$ or (ii) the all-terminal reliability is 1.

Initial Model Experimentation and Analysis

In order to train our model, we utilized a standard implementation of the RL problem as defined in [6] and implemented it in Python using the OpenAI stable baselines. The OpenAI package contains a set of reliable implementations of reinforcement learning algorithms. Each implementation contains a pre-trained RL agent who learns from observations, actions, and rewards.

For our research, we used Python version 3.8.15. We ran stable baselines OpenMPI to support all algorithms. Stable baselines required us to specify the number of training episodes for our models. For all models, we used 5000 episodes.

For our experiments, a Maskable Proximal Policy Optimization (M-PPO) algorithm was utilized [18]. This algorithm limits the action space to only feasible actions based on the defined problem constraints. For our specific problem, this means it limits the action space to adding edges that are not within the network, improving edges that are in the network, and ensuring these actions are within budget. The algorithm also uses a mask which is a vector that keeps track of valid actions. This will limit the actions that are included in the X and Y vectors.

Utilizing an iterative approach to developing and analyzing our network, we began by creating an initial model which only allowed one decision per period. This initial model only allowed one kind of action, which was dedicating resources to existing edges in the network to improve their reliabilities. This initial model was created to ensure the model and RL implementation were working properly. We then expanded the model to be more complex to allow us to make multiple decisions in each period and perform different kinds of actions (edge improvement, addition of new edges).

The initial model focused on improving one of the edges in the network. This model takes rewards which are initially arbitrarily assigned to each edge and determines the next best improvement to make. These rewards are expressed as reliability improvements that result from an upgrade to the chosen edge. This upgrade makes future rewards for this edge have a lower value. This initial model has a budget of 1000 units per time period, and each of the edges also has a cost of 1000 units to improve. We utilized the RL algorithm to evaluate each of the possible edges which could be improved, and it chooses the one with the highest immediate reward. The specific network used was the same as Figure 4 for the initial calculation of reliability in the methodology section. Because the model was limited to only choosing one edge to improve, this model runs very quickly. However, as it becomes more complex in future problem instances, we will start seeing the impact it has on the decision-making space and process.

After we analyzed the initial models, we expanded the model to also include the addition of two other action types: adding edges and choosing to move into a new period. We also gave the model a larger budget for each period so more improvements could be made. This larger budget allowed the network to choose more actions to take during a period. These additions made this new model substantially more computationally intensive than the basic model. We explored the decisions this model made in the designed experiments section.

Designed Experiments

In this section, we construct and evaluate a detailed set of experiments based on the insights gained during our preliminary analysis. The goal is to explore how the RL algorithm performs for a broadly defined design space. Table 3 lists the different parameters which were used for the testing. The improvement of edge reliabilities in each set of instances is specified by

two parameters, the *initial reliability* (which specifies the starting reliability of edges present in the initial network) and *new link* reliability (which specifies the common value of k_{ij} for all edges $\{i, j\}$ that are not in the initial network). For all sets of instances, we define $l_{ij} = 0.05$ as the increase to the reliability of edge $\{i, j\}$ as a result of improvement. For each of these sets, tests were performed using a network with 5-, 7-, and 10- nodes. For these tests, the number of periods, m, was set as either 3, 5, or 7, the reward ratio was set as either 2:1, 1:1, or 1:2, and the budget at the beginning of a period was 1000, 2000, or 3000. With these parameters, we ran a total of 243 tests. For all of these tests, we used the same costs to improve and add links as we did in the initial model which were $p_{ij} = 500$ and $c_{ij} = 2000$, respectively. Table 3 below outlines the different parameters we used. The results of the tests can be found in Tables 7-15 in the Appendix.

Table 3: Experimental design parameters

Initial/New Link Reliability	0.7/0.8	0.8/0.85	0.9/0.9
Number of Nodes	5	7	10
Number of Periods	3	5	7
Reward ratio Immediate/future	2:1	1:1	1:2
Budget per Period	1000	2000	3000

To determine which of the parameters may have been the best indicators of what actions the model would choose, we performed sensitivity analysis on the problem instances. We first analyzed the how the model responded to different levels of initial and new link reliabilities. We found that in general, the model trended towards favoring more improvements of existing links

rather than adding new links as the initial and new link reliabilities increased. The problem instances with initial link reliability of 0.7 and new link reliability, 0.8 cared more about adding links while the instances with initial and new link reliability, 0.9 cared more about improving links. For the lowest reliability combination of 0.7/0.8, we found problem instances wanted to add as many links as possible before improving links. However, as the reliability combinations increased to 0.8/0.85 and 0.9/0.9, the model chose to improve more links, and it also improved links earlier. In multiple instances of the highest reliability network (i.e. 7, (2:1), 3000 and 5, (1:1), 2000), improvements were the first action taken. Figure 5 below shows the rise of improvements as the link reliabilities increase.

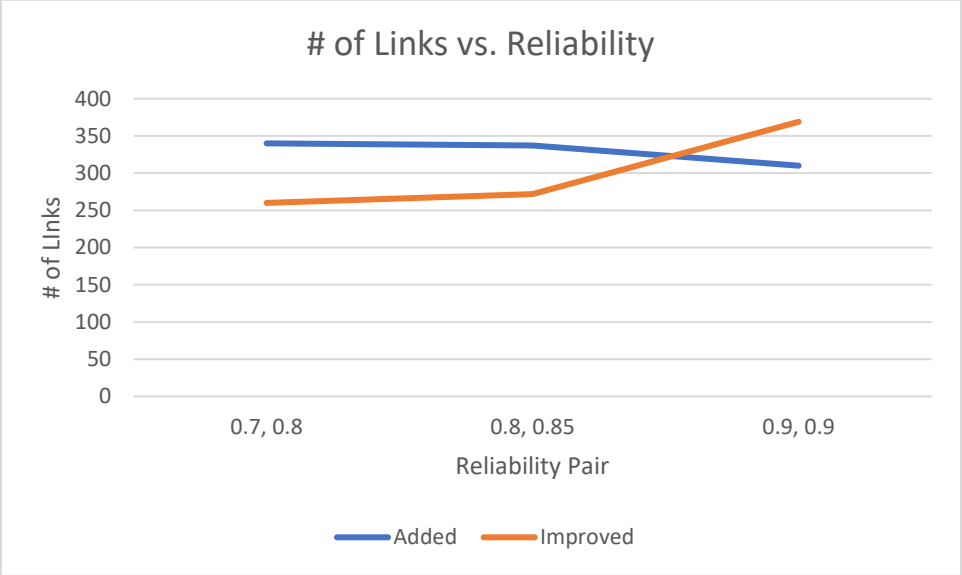


Figure 5: # of Links vs. Reliability Graph

We also analyzed the different reward ratios to see if they had an effect on the decisions the model made. During our analysis, we found that when the ratio was more favorable for immediate rewards, the model chose to improve more links, but when the ratio was more

favorable for long-term rewards, the model chose to add more links. Figure 6 shows the trend in regard to the rewards ratio.

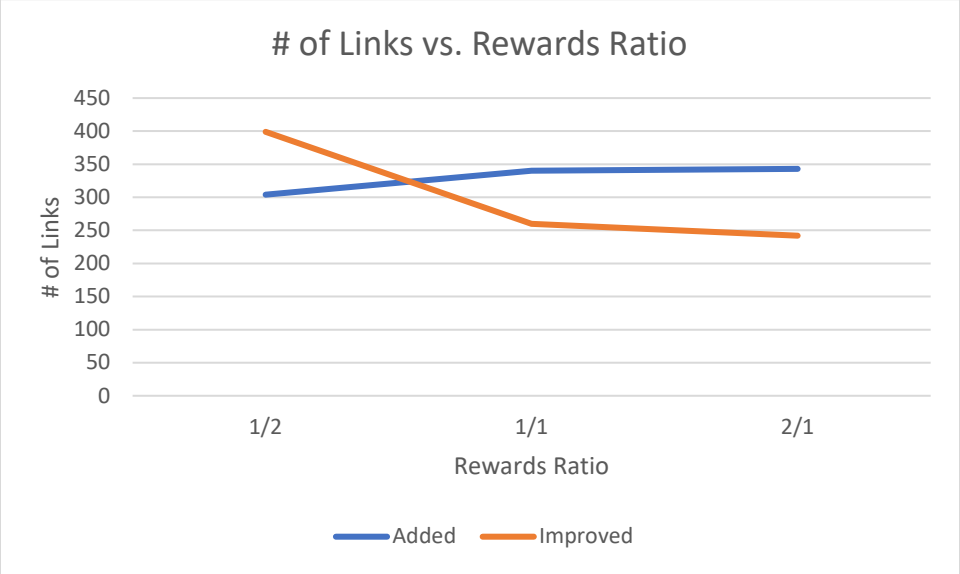


Figure 6: # of Links vs. Rewards Ratio Graph

We also found many instances where the model found better solutions for networks with lower budgets or fewer periods. These results may have been due to the training of the model. If the model had more training episodes, it may have been able to find better solutions for the larger networks. For example, in the five-node network with initial and new link reliabilities of 0.9/0.9, the problem instance with five periods, a 1:2 rewards ratio, and a per period budget of 3000 resulted in a final all-terminal reliability of 1.0 whereas the problem instance with seven periods, a 1:2 rewards ratio, and a per period budget of 3000 resulted in a final all-terminal reliability of 0.9995.

The networks with the highest reliabilities were typically 5-node networks because the total possible links in those networks was much lower than for 7- or 10- node networks. There were four networks with final reliabilities of 1.0, but not all of them were the largest networks with seven periods and a per period budget of 3000. As seen in Table 4, two of these networks

only had five periods. Another interesting find was a model with five periods and a budget of 2000 had a final reliability of 0.9999. This was better than many other models with more periods and larger total budgets. Additionally, we found the final two entries in this table were both seven node networks. Table 4 lists with the networks with the ten highest reliabilities.

Table 4: Highest All-Terminal Reliabilities

Problem Instance (# Periods, Rewards Ratio, Budget)	Final All-Terminal Reliability	Total computation time (min)	Initial Link Reliability	New Link Reliability	Nodes
7, (2:1), 3000	1.0	1.08	0.9	0.9	5
5, (1:1), 3000	1.0	1.1	0.9	0.9	5
7, (1:1), 3000	1.0	1.15	0.9	0.9	5
5, (1:2), 3000	1.0	0.8	0.9	0.9	5
5, (2:1), 3000	0.9999	0.93	0.9	0.9	5
7, (1:2), 3000	0.9999	1.55	0.9	0.9	5
7, (1:2), 2000	0.9999	1.07	0.9	0.9	5
5, (2:1), 2000	0.9999	0.82	0.9	0.9	5
7, (2:1), 3000	0.9999	1.52	0.8	0.85	5
7, (1:1), 3000	0.9997	740.07	0.9	0.9	7
7, (1:2), 3000	0.9997	717.55	0.9	0.9	7

The networks with the lowest reliabilities were all 10-node networks. The first nine networks had the lowest initial and new link reliabilities of 0.7/0.8 while the last one had reliability combination of 0.8/0.85. These results were not as interesting as the highest reliability table as all of the problem instances with the fewest periods, smallest budgets, and lowest

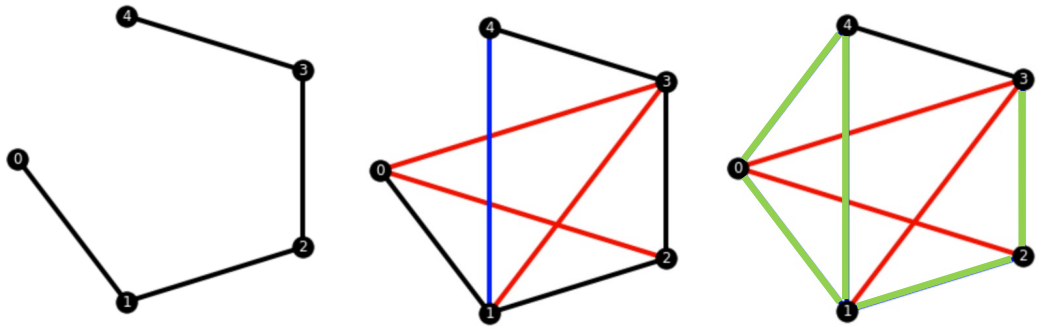
reliability combinations were the ones included in the results. Table 5 below shows the models that had the lowest reliabilities.

Table 5: Lowest All-Terminal Reliabilities

Problem Instance (# Periods, Rewards Ratio, Budget)	Final All-Terminal Reliability	Total computation time (min)	Initial Link Reliability	New Link Reliability	Nodes
3, (2:1), 1000	0.1337	1.22	0.7	0.8	10
3, (1:1), 1000	0.1349	1.2	0.7	0.8	10
3, (1:2), 1000	0.1726	1.13	0.7	0.8	10
5, (2:1), 1000	0.2289	5.6	0.7	0.8	10
5, (1:2), 1000	0.2635	2.08	0.7	0.8	10
5, (1:1), 1000	0.2684	2.07	0.7	0.8	10
3, (1:1), 2000	0.2825	8.33	0.7	0.8	10
3, (2:1), 2000	0.2889	9.6	0.7	0.8	10
7, (1:2), 1000	0.2968	18.73	0.7	0.8	10
3, (1:2), 1000	0.3114	1.07	0.8	0.85	10

For the 5-node networks, we found multiple instances where the final all-terminal reliability was 1.0. The example of the problem instance with initial and new link reliabilities of 0.9, five periods, a 1:1 rewards ratio, and a per period budget of 5000 is shown below. Figure 7 (a) shows the initial network configuration. Figure 7 (b) shows the configuration after four links have been added which is indicated by a red link and one has been improved which is indicated by a blue link. Figure 7 (c) shows the network in its final configuration where five links have

been added and ten have been improved. Even though ten links have been improved, some of them were improved more than once which is indicated by a green link. Three other problem instances also finished with a final all-terminal reliability of 1.0 and they can be found in Table 13 in the Appendix. These problem instances were the only ones in our modeling that did not use the entire budget.

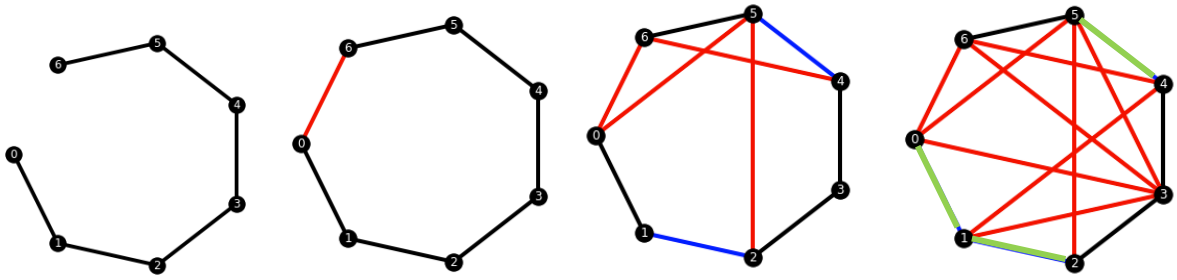


(a) (b) (c)

Figure 7: Iterations of Best 5-node network

A final all-terminal reliability of 1.0 is typically surprising because edges are usually upgraded by a percentage rather than a whole value. Because of this, we wanted to see what would happen if we increased the edge values by 5% for the 5-node networks with a reliability pair of 0.9/0.9. The results of this experimentation are shown in Table 16 in the Appendix. We did find that none of the networks reached a reliability of 1.0, but some of them came very close.

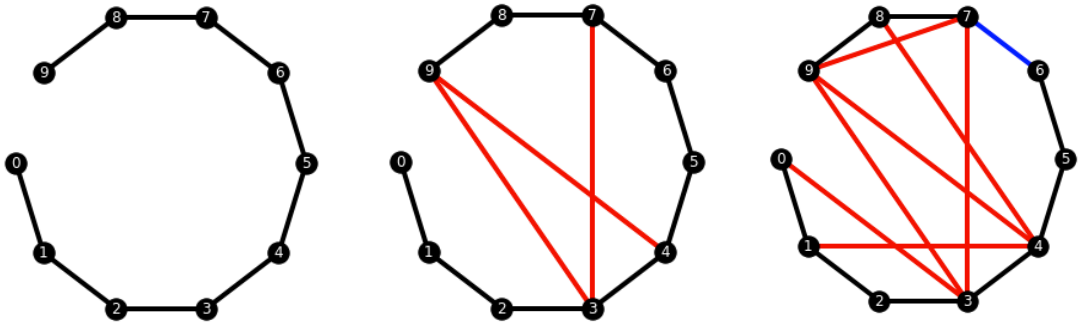
The best 7-node network had an all-terminal reliability of 0.9999. The problem instance which resulted in this reliability had initial and new link reliabilities of 0.9/0.9, seven periods, a 1:2 rewards ratio, and a per period budget of 7000. The model chose to add nine links and improve six. The iterations of different configurations of this network are shown in Figure 8. Green indicates links that have been improved more than once.



(a) (b) (c) (d)

Figure 8: Iterations of Best 7-node network

The best 10-node network had an all-terminal reliability of 0.9885. This problem instance had initial and new link reliabilities of 0.9/0.9. It had five periods, a rewards ratio of 1:1, and a per period budget of 3000. The configuration of the network is shown below. The iterations of this are shown below in Figure 9. The model chose to add seven links and make two improvements to the same edge to reach this reliability.



(a) (b) (c)

Figure 9: Iterations of Best 10-node network

Computation time was one of the biggest issues we faced when running the model. All of the five-node networks were very easy to compute as they all took less than two minutes. As expected, the computation time grew as we added nodes and increased the number of periods and budget. For the seven-node network, twenty-three of the eighty-one problem instances took over sixty minutes to complete. For the ten-node network, this number grew even more as thirty-eight

of eighty-one took more than an hour. These averages were heavily affected by the largest networks with five or seven periods and budgets of 2000 or 3000. For each of the largest 10-node networks with seven periods and a budget of 3000, it took more than twenty-two hours to run the model. Table 6 below shows the ten longest computation times and their associated problem instances. More results showing the issues with computational time are in the appendices.

Table 6: Highest Computation Times

Problem Instance (# Periods, Rewards Ratio, Budget)	Final All-Terminal Reliability	Total computation time (min)	Initial Link Reliability	New Link Reliability	Nodes
7, (1:1), 3000	0.9233	1432.02	0.8	0.85	10
7, (2:1), 3000	0.9257	1422.15	0.8	0.85	10
7, (2:1), 3000	0.9965	1416.03	0.9	0.9	10
7, (1:2), 3000	0.7870	1408.95	0.7	0.8	10
7, (1:1), 3000	0.9973	1403.75	0.9	0.9	10
7, (1:1), 3000	0.7633	1400.05	0.7	0.8	10
7, (1:2), 3000	0.9438	1398.35	0.8	0.85	10
7, (1:2), 3000	0.9982	1390.8	0.9	0.9	10
7, (2:1), 3000	0.7910	1385.2	0.7	0.8	10
5, (2:1), 3000	0.987	1057.08	0.9	0.9	10

Future Work

Our research focused on building 5-, 7-, and 10- node networks which were built using the same costs for adding and improving links. To further explore the model's decisions to add or improve links, different combinations of addition and improvement costs should be studied. Our research also only focused on three network sizes, this proved to be quite computationally expensive as the networks grew, so more research should be performed to create a model is able to make decisions more quickly. Although there are many sources of complexity that may have contributed to the long run-times, one opportunity for improvement is to replace the current reliability-polynomial procedure for evaluating network reliability with an alternative procedure that scales better for larger instances.

Additionally, for our research, the network was allowed to carry resources over to future periods. Further research could be conducted to see study the behavior of the model if the number of resources that could be carried into the next period was limited or eliminated completely. The model we used continuously chose to carry forward resources to add a new link even though the resources available could have been used to improve links in the previous period. Studying the effects of limiting resources would be valuable for this model.

For our research, we did not use the traditional reliability growth approach of test, analyze, fix. Instead, we used a cognate approach which still aimed to optimize resource allocation to improve the network over time. Future work can be done to investigate the feasibility of using reinforcement learning to see what decisions would be made in the traditional reliability growth model. In this model, the network would fail due to links degrading over time as they do in infrastructure networks.

One issue with our model was that it was set up to improve the links by a whole number rather than a percentage of the current reliability. We did some research into how making this change would affect the results, but future work should only focus on percentage changes. Another change we would make to the model is to give the links diminishing returns after they reach a certain reliability threshold. This would make the model want to choose many different links to add or improve rather than focusing on a select few.

We also made many assumptions regarding the initial link reliabilities, new link reliabilities, amount to improve, cost to add and improve, and rewards. These assumptions limited the results we got from our experimentation, so future work should focus on creating more models with less restrictive assumptions.

References

1. AboElFotoh, H. M., & Al-Sumait, L. S. (2001, December). A Neural Approach to Topological Optimization of Communication Networks, with Reliability Constraints. *IEEE Transactions on Reliability, 50*(4), 397-408.

2. Acevedo, P. E., Jackson, D. S., & Kotlowitz, R. W. (2006). Reliability Growth and Forecasting for Critical Hardware through Accelerated Life Testing. *Bell Labs Technical Journal, 11*(3), 121-135.

3. Ahmad, N., Bokhari, M., Quadri, S., & Khan, M. (2008). The exponentiated Weibull software reliability growth model with various testing-efforts and optimal release policy: A performance analysis. *The International Journal of Quality & Reliability Management, 25*(2), 211-235.

4. Ball, M. O. (1986, August). Computational Complexity of Network Reliability Analysis: An Overview. *IEEE Transactions on Reliability, 35*(3), 230-239.

5. Ball, M. O., Colbourn, C. J., & Provan, J. S. (1995). Network Reliability. In *Handbooks in Operations Research and Management Science* (Vol. 7, pp. 673-762).

6. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *arXiv preprint arXiv:1606.01540*.

7. Cahoon, J., Sanborn, K., & Wilson, A. (2021, November). Practical reliability growth modeling. *Quality and Reliability Engineering International Special Issue: Advancing Statistical Methods for Testing and Evaluating Defense Systems, 37*(7), 3108-3124.

8. Cardoso, J. B., de Almeida, J. R., Dias, J. M., & Coelho, P. G. (2008, June). Structural reliability analysis using Monte Carlo simulation and neural networks. *Advances in Engineering Software, 39*(6), 505-513.

9. Cinque, M., Cotroneo, D., Pecchia, A., Pietrantuono, R., & Russo, S. (2017). Debugging-workflow-aware software reliability growth analysis. *Software: Testing Verification, and Reliability, 27*(7).

10. Crow, L. H. (1975). *Reliability Analysis for Complex, Repairable Systems.* Aberdeen Proving Ground, Maryland: Army Material Systems Analysis Activity.

11.Dempsey, P. J., & Sheng, S. (2013, May). Investigation of data fusion applied to health monitoring of wind turbine drivetrain components. *Wind Energy, 16*(4), 479-489.

12. Duane, J. T. (1964, April). Learning Curve Approach to Reliability Monitoring. *IEEE Transactions on Aerospace, 2*(2), 563-566.

13. Editors, H. (2009, November 24). *Blackout hits Northeast United States*. Retrieved from History: https://www.history.com/this-day-in-history/blackout-hits-northeast-united-states

14. Fries, A. (1993, June). Discrete Reliability-Growth Models Based on a Learning-Curve Property. *IEEE Transactions on Reliability, 42*(2), 303-306.

15. Gaur, V., Yadav, O. P., Soni, G., & Rathore, A. P. (2021). A literature review on network reliability analysis and its engineering applications. *Journal of Risk and Reliability, 235*(2), 167-181.

16. Goldbeck, N., Angeloudis, P., & Ochieng, W. Y. (2019, August). Resilience assessment for interdependent urban infrastructure systems using dynamic network flow models. *Reliability Engineering & System Safety, 188*, 62-79.

17. Gottesman, O., Johansson, F., Komorowski, M., Faisal, A., & Sontag, D. (2019, Jan). Guidelines for reinforcement learning in healthcare. *Nature Medicine, 25*(1), 16-18.

18. Huang, S., & Ontañón, S. (2020). A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. *arXiv preprint arXiv:2006.14171*.

19. Hui, K.-P. (2005). *Network Reliability Estimation.* Doctoral Dissertation.

20. Hussin, M., Hamid, N. A., & Kasmiran, K. A. (2015, January). Improving reliability in resource management through adaptive reinforcement learning for distributed systems. *Journal of Parallel and Distributed Computing, 75*, 93-100.

21. Jain, A. K., & Mao, J. (1996). Artificial Neural Networks: A Tutorial. *Computer*, 31-44.

22. Jan, R.-H., Hwang, F.-J., & Cheng, S.-T. (1993, March). Topological Optimization of a Communication Network Subject to a. *IEEE Transactions on Reliability, 42*(1), 63-70.

23. Kanoun, K., Kaaniche, M., Beounes, C., Laprie, J.-C., & Arlat, J. (1993, June). Reliability Growth of Fault-Tolerant Software. *IEEE Transactions on Reliability, 42*(2), 205-219.

24. Kapur, P., & Garg, R. (1992, July). A Software-Reliability Growth-Model for an Error-Removal Phenomenon. *Software Engineering Jornal, 7*(4), 291-294.

25. Karger, D. R. (1999). A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Journal on Computing, 29*(2), 492-514.

26. Kurtz, J., Sprik, S., & Bradley, T. H. (2019). Review of transportation hydrogen infrastructure performance and reliability. *International Journal of Hydrogen Energy*, 12010-12023.

27. Lin, C.-J., Jhang, J.-Y., Lin, H.-Y., Lee, C.-L., & Young, K.-Y. (2019). Using a Reinforcement Q-Learning-Based Deep Neural Network for Playing Video Games. *Electronics*.

28. Lloyd, D. K. (1986). Forecasting Reliability Growth. *Quality and Reliability Engineering International, 2*, 19-23.

29. Mahmoodzadeh, Z., Wu, K.-Y., Droguett, E. L., & Mosleh, A. (2020). Condition-Based Maintenance with Reinforcement Learning for Dry Gas Pipeline Subject to Internal Corrosion. *Sensors, 20*(19).

30. Mettas, A. (2000). Reliability Allocation and Optimization for Complex Systems. *Annual Reliability and Maintainability Symposium. 2000 Proceedings. International Symposium on Product Quality and Integrity*, 216-221.

31. Milanovic, J. V., & Zhu, W. (2018, September). Modeling of Interconnected Critical Infrastructure Systems Using Complex Network Theory. *IEEE Transactions on Smart Grid, 9*(5), 4637-4648.

32. Newman, M. (2010). *Networks: An introduction.* Oxford, United Kingdom: Oxford University Press.

33. Provan, J. S., & Ball, M. O. (1983, November). The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected. *SIAM Journal on Computing, 12*(4), 777-788.

34. Pruitt, S. (2020, August 27). *How Levee Failures Made Hurricane Katrina a Bigger Disaster*. Retrieved from History: https://www.history.com/news/hurricane-katrina-levee-failures

35. Ramirez-Marquez, J. E., & Rocco, C. M. (2008, November). All-terminal network reliability optimization via probabilistic solution discovery. *Reliability Engineering & System Safety, 93*(11), 1689-1697.

36. Satitsatian, S., & Kapur, K. C. (2006, June). An Algorithm for Lower Reliability Bounds of Multistate Two-Terminal Networks. *IEEE Transactions on Reliability, 55*(2), 199-206.

37. Sebastio, S., Trivedi, K. S., Wang, D., & Yin, X. (2014, November 1). Fast computation of bounds for two-terminal network reliability. *European Journal of Operational Research, 238*(3), 810-823.

38. Sen, A., & Fries, A. (1998). Model Misspecification Effects Within A Family of Alternative Discrete Reliability-Growth Models. *Lifetime Data Analysis*, 65-81.

39. Serrano, W. (2019). Deep Reinforcement Learning Algorithms in Intelligent Infrastructure. *Infrastructures, 4*(3).

40. Shao, X., & Fang, X. (2016). Estimation in discrete reliability growth, a growth model with coefficient condition. *Journal of Systems Science and Complexity*, 1112-1122.

41. Srivaree-ratana, C., Konak, A., & Smith, A. E. (2002, June). Estimation of all-terminal network reliability using an artificial neural network. *Computers & Operations Research, 29*(7), 849-868.

42. Su, Y.-C., Mays, L. W., Duan, N., & Lansey, K. E. (1987, December). Reliability-Based Optimization Model for Water Distribution Systems. *Journal of Hydraulic Engineering, 113*(12), 1539-1556.

43. Talafuse, T. P., & Pohl, E. A. (2017). Small sample reliability growth modeling using a grey systems model. *Quality Engineering, 29*(3), 455-467.

44. U.S. NRC. (2018, June 21). *Backgrounder on the Three Mile Island Accident*. Retrieved from United States Nuclear Regulatory Commission: https://www.nrc.gov/reading-rm/doc-collections/fact-sheets/3mile-isle.html

45. Yang, T., Hu, Y., Gursoy, M. C., Schmeink, A., & Mathar, R. (2018). Deep Reinforcement Learning based Resource Allocation in Low Latency Edge Computing Networks. *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, 1-5.

46. Yeh, W.-C., Lin, Y.-C., Chung, Y. Y., & Chih, M. (2010, March). A Particle Swarm Optimization Approach Based on Monte Carlo Simulation for Solving the Complex Network Reliability Problem. *IEEE Transactions on Reliability, 59*(1), 212-221.

47. Zhang, X.-D. (2020). Machine Learning. In *A Matrix Algebra Approach to Artificial Intelligence* (pp. 223-225). Singapore: Springer.

Appendices

Table 7: Problem Instances with Initial Link Reliability 0.7, New Link 0.8, Nodes 5

Problem Instance (# Periods, Rewards Ratio, Budget)	# of edges added	# of edges improved	Total Budget Used	Final All-Terminal Reliability	Total Computation Time (min)
3, (2:1), 1000	1	2	3000	0.6194	0.65
5, (2:1), 1000	2	2	5000	0.7244	0.683
7, (2:1), 1000	3	2	7000	0.8442	0.783
3, (1:1), 1000	1	2	3000	0.6194	0.667
5, (1:1), 1000	2	2	5000	0.7260	0.7
7, (1:1), 1000	3	2	7000	0.8371	0.767
3, (1:2), 1000	1	2	3000	0.6115	0.667
5, (1:2), 1000	2	2	5000	0.7244	0.667
7, (1:2), 1000	3	2	7000	0.8545	0.767
3, (2:1), 2000	2	4	6000	0.7605	0.717
5, (2:1), 2000	4	4	10000	0.9440	0.983
7, (2:1), 2000	5	8	14000	0.9782	1.23
3, (1:1), 2000	3	0	6000	0.8274	0.75
5, (1:1), 2000	5	0	10000	0.9611	1.02
7, (1:1), 2000	6	4	14000	0.9922	1.25
3, (1:2), 2000	3	0	6000	0.8520	0.767

5, (1:2), 2000	5	0	10000	0.9562	1.02
7, (1:2), 2000	6	4	14000	0.9922	1.32
3, (2:1), 3000	4	2	9000	0.9270	0.95
5, (2:1), 3000	6	6	15000	0.9908	1.43
7, (2:1), 3000	6	18	21000	0.9982	1.43
3, (1:1), 3000	4	2	9000	0.8818	0.967
5, (1:1), 3000	6	6	15000	0.9930	1.32
7, (1:1), 3000	6	18	21000	0.9978	1.45
3, (1:2), 3000	4	2	9000	0.9306	0.933
5, (1:2), 3000	6	6	15000	0.9925	1.28
7, (1:2), 3000	6	18	21000	0.9971	1.43

Table 8: Problem Instances with Initial Link Reliability 0.7, New Link 0.8, Nodes 7

Problem Instance (# Periods, Rewards Ratio, Budget)	# of edges added	# of edges improved	Total Budget Used	Final All-Terminal Reliability	Total Computation Time (min)
3, (2:1), 1000	1	2	3000	0.3445	0.817
5, (2:1), 1000	2	2	5000	0.5540	1.12
7, (2:1), 1000	3	2	7000	0.6382	1.92
3, (1:1), 1000	1	2	3000	0.3445	0.833
5, (1:1), 1000	2	2	5000	0.5066	1.1
7, (1:1), 1000	3	2	7000	0.7010	1.85
3, (1:2), 1000	1	2	3000	0.3345	0.867
5, (1:2), 1000	2	2	5000	0.5299	1.13
7, (1:2), 1000	3	2	7000	0.6465	1.82
3, (2:1), 2000	3	0	6000	0.6629	1.58
5, (2:1), 2000	5	0	10000	0.8194	9.72
7, (2:1), 2000	7	0	14000	0.9243	70.98
3, (1:1), 2000	3	0	6000	0.5595	1.45
5, (1:1), 2000	5	0	10000	0.7833	9.33
7, (1:1), 2000	7	0	14000	0.9270	71.32
3, (1:2), 2000	3	0	6000	0.6384	1.62
5, (1:2), 2000	5	0	10000	0.8659	7.88

7, (1:2), 2000	7	0	14000	0.9375	63.7
3, (2:1), 3000	4	2	9000	0.7246	4.97
5, (2:1), 3000	6	6	15000	0.9150	101.03
7, (2:1), 3000	5	22	21000	0.8930	706.8
3, (1:1), 3000	4	2	9000	0.7832	5.05
5, (1:1), 3000	7	2	15000	0.8650	98.77
7, (1:1), 3000	10	2	21000	0.9754	713.02
3, (1:2), 3000	4	2	9000	0.7953	4.58
5, (1:2), 3000	7	2	15000	0.9439	101.48
7, (1:2), 3000	9	6	21000	0.9615	714.65

Table 9: Problem Instances with Initial Link Reliability 0.7, New Link 0.8, Nodes 10

Problem Instance (# Periods, Rewards Ratio, Budget)	# of edges added	# of edges improved	Total Budget Used	Final All-Terminal Reliability	Total Computation Time (min)
3, (2:1), 1000	1	2	3000	0.1337	1.22
5, (2:1), 1000	2	2	5000	0.2289	5.6
7, (2:1), 1000	3	2	7000	0.3388	23.47
3, (1:1), 1000	1	2	3000	0.1349	1.2
5, (1:1), 1000	2	2	5000	0.2684	2.07
7, (1:1), 1000	3	2	7000	0.3948	20.05
3, (1:2), 1000	1	2	3000	0.1726	1.13
5, (1:2), 1000	2	2	5000	0.2635	2.08
7, (1:2), 1000	3	2	7000	0.2968	18.73
3, (2:1), 2000	3	0	6000	0.2889	9.6
5, (2:1), 2000	4	4	10000	0.5394	96.83
7, (2:1), 2000	7	0	14000	0.7347	580.42
3, (1:1), 2000	3	0	6000	0.2825	8.33
5, (1:1), 2000	4	4	10000	0.4769	93.2
7, (1:1), 2000	7	0	14000	0.7313	598.33
3, (1:2), 2000	3	0	6000	0.3158	8.3
5, (1:2), 2000	5	0	10000	0.5515	86.72

7, (1:2), 2000	7	0	14000	0.7439	548.15
3, (2:1), 3000	4	2	9000	0.4238	61.87
5, (2:1), 3000	5	10	15000	0.6780	976.65
7, (2:1), 3000	7	14	21000	0.7910	1385.2
3, (1:1), 3000	4	2	9000	0.4667	60.90
5, (1:1), 3000	6	6	15000	0.6820	843.4
7, (1:1), 3000	9	6	21000	0.7633	1400.05
3, (1:2), 3000	4	2	9000	0.4187	59.47
5, (1:2), 3000	6	6	15000	0.6531	978.05
7, (1:2), 3000	10	2	21000	0.7870	1408.95

Table 10: Problem Instances with Initial Link Reliability 0.8, New Link 0.85, Nodes 5

Problem Instance (# Periods, Rewards Ratio, Budget)	# of edges added	# of edges improved	Total Budget Used	Final All-Terminal Reliability	Total Computation Time (min)
3, (2:1), 1000	1	2	3000	0.8026	0.85
5, (2:1), 1000	2	2	5000	0.8711	0.83
7, (2:1), 1000	3	2	7000	0.9436	0.9
3, (1:1), 1000	1	2	3000	0.8026	0.85
5, (1:1), 1000	2	2	5000	0.8699	0.92
7, (1:1), 1000	3	2	7000	0.9412	0.9
3, (1:2), 1000	1	2	3000	0.8026	0.82
5, (1:2), 1000	2	2	5000	0.8915	0.97
7, (1:2), 1000	3	2	7000	0.9109	0.87
3, (2:1), 2000	2	4	6000	0.8874	0.68
5, (2:1), 2000	4	4	10000	0.9827	0.95
7, (2:1), 2000	6	4	14000	0.9968	1.17
3, (1:1), 2000	3	0	6000	0.9321	0.73
5, (1:1), 2000	5	0	10000	0.9861	1.00
7, (1:1), 2000	6	4	14000	0.9980	1.25
3, (1:2), 2000	3	0	6000	0.9321	0.72
5, (1:2), 2000	5	0	10000	0.9861	0.95

7, (1:2), 2000	6	4	14000	0.9980	1.25
3, (2:1), 3000	3	6	10000	0.9860	0.92
5, (2:1), 3000	6	6	15000	0.9972	1.28
7, (2:1), 3000	4	26	21000	0.9999	1.52
3, (1:1), 3000	4	2	9000	0.9767	0.93
5, (1:1), 3000	6	6	15000	0.9977	1.47
7, (1:1), 3000	6	18	21000	0.9993	1.48
3, (1:2), 3000	4	2	9000	0.9807	0.92
5, (1:2), 3000	6	6	15000	0.9985	1.32
7, (1:2), 3000	7	14	21000	0.9999	1.55

Table 11: Problem Instances with Initial Link Reliability 0.8, New Link 0.85, Nodes 7

Problem Instance (# Periods, Rewards Ratio, Budget)	# of edges added	# of edges improved	Total Budget Used	Final All-Terminal Reliability	Total Computation Time (min)
3, (2:1), 1000	1	2	3000	0.5734	0.87
5, (2:1), 1000	2	2	5000	0.7110	1.3
7, (2:1), 1000	3	2	7000	0.8610	1.9
3, (1:1), 1000	1	2	3000	0.6357	0.87
5, (1:1), 1000	2	2	5000	0.7426	1.2
7, (1:1), 1000	3	2	7000	0.8521	1.85
3, (1:2), 1000	1	2	3000	0.6396	0.97
5, (1:2), 1000	2	2	5000	0.7591	1.13
7, (1:2), 1000	3	2	7000	0.8237	1.77
3, (2:1), 2000	3	0	6000	0.8496	1.53
5, (2:1), 2000	5	0	10000	0.9080	7.88
7, (2:1), 2000	6	4	14000	0.9740	55.92
3, (1:1), 2000	3	0	6000	0.8372	1.83
5, (1:1), 2000	5	0	10000	0.9274	8.36
7, (1:1), 2000	7	0	14000	0.9735	62.63
3, (1:2), 2000	3	0	6000	0.8259	1.68
5, (1:2), 2000	4	4	10000	0.9069	8.02

7, (1:2), 2000	7	0	14000	0.9442	60.17
3, (2:1), 3000	4	2	9000	0.9216	4.72
5, (2:1), 3000	6	6	15000	0.9707	91.17
7, (2:1), 3000	8	10	21000	0.9975	755.25
3, (1:1), 3000	4	2	9000	0.9195	4.62
5, (1:1), 3000	7	2	15000	0.9515	86.13
7, (1:1), 3000	10	2	21000	0.9962	787.82
3, (1:2), 3000	4	2	9000	0.8936	4.58
5, (1:2), 3000	7	2	15000	0.9696	88.43
7, (1:2), 3000	10	2	21000	0.9963	728.95

Table 12: Problem Instances with Initial Link Reliability 0.8, New Link 0.85, Nodes 10

Problem Instance (# Periods, Rewards Ratio, Budget)	# of edges added	# of edges improved	Total Budget Used	Final All-Terminal Reliability	Total Computation Time (min)
3, (2:1), 1000	1	2	3000	0.3574	1.02
5, (2:1), 1000	2	2	5000	0.5293	1.87
7, (2:1), 1000	2	6	7000	0.5940	8.03
3, (1:1), 1000	1	2	3000	0.3435	1.08
5, (1:1), 1000	1	6	5000	0.4483	2.28
7, (1:1), 1000	2	6	7000	0.5822	7.27
3, (1:2), 1000	1	2	3000	0.3114	1.07
5, (1:2), 1000	2	2	5000	0.4041	2.38
7, (1:2), 1000	3	2	7000	0.5394	7.33
3, (2:1), 2000	3	0	6000	0.6571	5.15
5, (2:1), 2000	5	0	10000	0.8089	90.22
7, (2:1), 2000	6	4	14000	0.8504	579.60
3, (1:1), 2000	2	4	6000	0.5461	6.07
5, (1:1), 2000	4	4	10000	0.6698	90.8
7, (1:1), 2000	7	0	14000	0.8555	574.28
3, (1:2), 2000	2	4	6000	0.5411	5.32
5, (1:2), 2000	4	4	10000	0.7164	97.05

7, (1:2), 2000	7	0	14000	0.9042	619.7
3, (2:1), 3000	3	6	9000	0.6775	55.35
5, (2:1), 3000	6	6	15000	0.8643	981.1
7, (2:1), 3000	8	10	21000	0.9257	1422.15
3, (1:1), 3000	4	2	9000	0.7044	57.82
5, (1:1), 3000	7	2	15000	0.8891	997.4
7, (1:1), 3000	9	6	21000	0.9233	1432.02
3, (1:2), 3000	4	2	9000	0.6499	59.55
5, (1:2), 3000	6	6	15000	0.8764	948.7
7, (1:2), 3000	10	2	21000	0.9438	1398.35

Table 13: Problem Instances with Initial Link Reliability 0.9, New Link 0.9, Nodes 5

Problem Instance (# Periods, Rewards Ratio, Budget)	# of edges added	# of edges improved	Total Budget Used	Final All-Terminal Reliability	Total Computation Time (min)
3, (2:1), 1000	1	2	3000	0.9477	0.85
5, (2:1), 1000	1	6	5000	0.9900	0.83
7, (2:1), 1000	2	6	7000	0.9801	0.8
3, (1:1), 1000	1	2	3000	0.9477	0.75
5, (1:1), 1000	2	2	5000	0.9696	0.75
7, (1:1), 1000	3	2	7000	0.9871	0.83
3, (1:2), 1000	1	2	3000	0.8748	0.7
5, (1:2), 1000	2	2	5000	0.8690	0.57
7, (1:2), 1000	3	2	7000	0.9368	0.67
3, (2:1), 2000	2	4	6000	0.9882	0.73
5, (2:1), 2000	3	8	10000	0.9999	0.82
7, (2:1), 2000	6	4	14000	0.9966	1.1
3, (1:1), 2000	3	0	6000	0.9295	0.62
5, (1:1), 2000	3	8	10000	0.9990	0.9
7, (1:1), 2000	5	8	14000	0.9988	1.13
3, (1:2), 2000	3	0	6000	0.9281	0.62
5, (1:2), 2000	5	0	10000	0.9872	0.83

7, (1:2), 2000	2	20	14000	0.9999	1.07
3, (2:1), 3000	4	2	9000	0.9813	0.77
5, (2:1), 3000	4	14	15000	0.9999	0.93
7, (2:1), 3000	3	25	18500	1.0000	1.08
3, (1:1), 3000	4	2	9000	0.9792	0.83
5, (1:1), 3000	5	10	15000	1.0000	1.1
7, (1:1), 3000	5	22	21000	1.0000	1.15
3, (1:2), 3000	4	2	9000	0.9806	0.78
5, (1:2), 3000	3	12	12000	1.0000	0.8
7, (1:2), 3000	5	22	21000	0.9995	1.17

Table 14: Problem Instances with Initial Link Reliability 0.9, New Link 0.9, Nodes 7

Problem Instance (# Periods, Rewards Ratio, Budget)	# of edges added	# of edges improved	Total Budget Used	Final All-Terminal Reliability	Total Computation Time (min)
3, (2:1), 1000	1	2	3000	0.8267	0.8
5, (2:1), 1000	2	2	5000	0.9324	1.07
7, (2:1), 1000	3	2	7000	0.9536	1.88
3, (1:1), 1000	1	2	3000	0.8267	0.85
5, (1:1), 1000	2	2	5000	0.9123	1.12
7, (1:1), 1000	3	2	7000	0.9719	1.72
3, (1:2), 1000	1	2	3000	0.8857	0.83
5, (1:2), 1000	2	2	5000	0.9330	1.05
7, (1:2), 1000	3	2	7000	0.9654	1.77
3, (2:1), 2000	2	4	6000	0.9441	1.47
5, (2:1), 2000	5	0	10000	0.9681	6.67
7, (2:1), 2000	5	8	14000	0.9988	51.28
3, (1:1), 2000	2	4	6000	0.9179	1.62
5, (1:1), 2000	5	0	10000	0.9844	7.2
7, (1:1), 2000	7	0	14000	0.9965	53.57
3, (1:2), 2000	3	0	6000	0.9396	1.58
5, (1:2), 2000	5	0	10000	0.9681	7.78

7, (1:2), 2000	7	0	14000	0.9966	56.87
3, (2:1), 3000	4	2	9000	0.9664	4.72
5, (2:1), 3000	6	6	15000	0.9978	84.37
7, (2:1), 3000	8	10	21000	0.9990	743.15
3, (1:1), 3000	4	2	9000	0.9743	4.48
5, (1:1), 3000	6	6	15000	0.9988	81.6
7, (1:1), 3000	9	6	21000	0.9997	740.07
3, (1:2), 3000	4	2	9000	0.9839	4.43
5, (1:2), 3000	7	2	15000	0.9977	90.95
7, (1:2), 3000	9	6	21000	0.9999	717.55

Table 15: Problem Instances with Initial Link Reliability 0.9, New Link 0.9, Nodes 10

Problem Instance (# Periods, Rewards Ratio, Budget)	# of edges added	# of edges improved	Total Budget Used	Final All-Terminal Reliability	Total Computation Time (min)
3, (2:1), 1000	1	2	3000	0.7705	1.12
5, (2:1), 1000	2	2	5000	0.8519	2.17
7, (2:1), 1000	3	2	7000	0.8960	6.08
3, (1:1), 1000	1	2	3000	0.6457	1.08
5, (1:1), 1000	2	2	5000	0.8519	2.27
7, (1:1), 1000	3	2	7000	0.9074	6.47
3, (1:2), 1000	1	2	3000	0.6887	1.05
5, (1:2), 1000	2	2	5000	0.8063	2.23
7, (1:2), 1000	3	2	7000	0.8454	6.18
3, (2:1), 2000	2	4	6000	0.8839	4.68
5, (2:1), 2000	4	4	10000	0.9021	89.92
7, (2:1), 2000	6	4	14000	0.9591	650.92
3, (1:1), 2000	3	0	6000	0.8996	4.6
5, (1:1), 2000	4	4	10000	0.9167	107.15
7, (1:1), 2000	7	0	14000	0.9585	618
3, (1:2), 2000	3	0	6000	0.8585	8.68
5, (1:2), 2000	5	0	10000	0.9264	86.43

7, (1:2), 2000	7	0	14000	0.9712	598.85
3, (2:1), 3000	3	6	9000	0.8942	54.27
5, (2:1), 3000	5	10	15000	0.9870	1057.08
7, (2:1), 3000	6	18	21000	0.9865	1416.03
3, (1:1), 3000	4	2	9000	0.9108	53.78
5, (1:1), 3000	7	2	15000	0.9885	1035.55
7, (1:1), 3000	7	14	21000	0.9773	1403.75
3, (1:2), 3000	4	2	9000	0.9272	57.58
5, (1:2), 3000	6	6	15000	0.9601	1037.98
7, (1:2), 3000	8	10	21000	0.9772	1390.8

Table 16: Problem Instances with Initial Link Reliability 0.9, New Link 0.9, Nodes 5 using 5% increase

Problem Instance (# Periods, Rewards Ratio, Budget)	# of edges added	# of edges improved	Total Budget Used	Final All-Terminal Reliability	Total Computation Time (min)
3, (2:1), 1000	1	2	3000	0.9454	0.72
5, (2:1), 1000	2	2	5000	0.9687	0.75
7, (2:1), 1000	3	2	7000	0.9950	0.8
3, (1:1), 1000	1	2	3000	0.9454	0.72
5, (1:1), 1000	2	2	5000	0.9687	0.7
7, (1:1), 1000	3	2	7000	0.9852	0.8
3, (1:2), 1000	1	2	3000	0.9454	0.73
5, (1:2), 1000	2	2	5000	0.9693	0.75
7, (1:2), 1000	3	2	7000	0.9852	0.82
3, (2:1), 2000	2	4	6000	0.9711	0.8
5, (2:1), 2000	4	4	10000	0.9982	1.08
7, (2:1), 2000	4	12	14000	0.9989	1.17
3, (1:1), 2000	3	0	6000	0.9842	0.77
5, (1:1), 2000	4	4	10000	0.9985	0.97
7, (1:1), 2000	4	12	14000	0.9990	1.13
3, (1:2), 2000	3	0	6000	0.9842	0.78

5, (1:2), 2000	5	0	10000	0.9977	1.05
7, (1:2), 2000	6	4	14000	0.9995	1.23
3, (2:1), 3000	4	2	9000	0.9977	0.97
5, (2:1), 3000	4	14	15000	0.9998	1.25
7, (2:1), 3000	5	22	21000	0.9999	1.42
3, (1:1), 3000	4	2	9000	0.9966	0.92
5, (1:1), 3000	4	14	15000	0.9999	1.27
7, (1:1), 3000	5	22	21000	0.9999	1.38
3, (1:2), 3000	4	2	9000	0.9977	0.97
5, (1:2), 3000	6	6	15000	0.9998	1.22
7, (1:2), 3000	6	18	21000	0.9999	1.35