Technical University of Crete, Greece
Department of Electronic and Computer Engineering

Learning Strategies for Network Fault Detection and Remediation

Aggelos Aggelidakis

Thesis Committee
Assistant Professor Michail G. Lagoudakis (ECE)
Assistant Professor Georgios Chalkiadakis (ECE)
Assistant Professor Polychronis Koutsakis (ECE)

Chania, February 2013

Πολυτεχνειο Κρητης

Τμημα Ηλεκτρονικων Μηχανικων και Μηχανικων Υπολογιστων

Μάθηση Στρατηγικών για Εντοπισμό και Αποκατάσταση Βλαβών Δικτύου

Άγγελος Αγγελιδάκης

Εξεταστική Επιτροπή

Επίκουρος Καθηγητής Μιχαήλ Γ. Λαγουδάκης (ΗΜΜΥ)

Επίκουρος Καθηγητής Γεώργιος Χαλκιαδάκης (ΗΜΜΥ)

Επίκουρος Καθηγητής Πολυχρόνης Κουτσάκης (ΗΜΜΥ)

Χανιά, Φεβρουάριος 2013

Abstract

Network repair is a domain of growing significance. Viewing the network repair problem as a sequential decision problem, offers the opportunity to design autonomous agents able to implement minimum-cost fault detection and remediation policies through the execution of appropriate test and repair actions on the network. A repair policy consists of a sequence of test actions that help diagnose the faulty node of the network, followed by repair actions to restore it to proper operation with minimum cost. This thesis extends the Cost-Sensitive Fault Remediation (CSFR) model developed for such problems by introducing a minimax criterion for planning and learning and applies the extended CSFR model and algorithms to the problem of network repair. The minimax criterion, in contrast to the minimum expected cost criterion of the original CSFR planning algorithm, aims to deliver safer policies which keep the maximum (worst) cost low. Planning with the minimax criterion is complemented with three heuristics for resolving possible repair action ordering issues. The original planning algorithm and the three variations of the proposed planning algorithm were applied on fault detection and remediation problems in simulated networks of various sizes. To account for the computational limitations of the planning agent, another agent was developed using instance-based reinforcement learning. While the learning agent cannot guarantee optimality, it can handle much larger problems by exploiting data collected through interaction with the environment. Our results with both planning and learning indicate that the CSFR-based approach is able to handle network repair problems effectively. In several cases, the proposed minimax criterion yields a maximum actual repair cost which is lower compared to the maximum actual cost of the minimum expected cost criterion.

Περίληψη

Η αποκατάσταση δικτύων είναι ένας τομέας αυξανόμενης σημασίας. Βλέποντας το πρόβλημα αποκατάστασης δικτύου ως ένα πρόβλημα διαδοχικών αποφάσεων, δίνεται η ευκαιρία για το σχεδιασμό ενός αυτόνομου πράκτορα, ο οποίος είναι ικανός να υλοποιήσει πολιτικές ελαχίστου κόστους, οι οποίες στοχεύουν στην ανίχνευση της βλάβης και στην αποκατάσταση, μέσω της εκτέλεσης κατάλληλων ενεργειών διάγνωσης και επιδιόρθωσης στο δίκτυο. Μια πολιτική αποκατάστασης αποτελείται από μια σειρά ενεργειών διάγνωσης, που βοηθούν στον εντοπισμό του εσφαλμένου κόμβου του δικτύου, και από ενέργειες επιδιόρθωσης για να αποκατασταθεί η σωστή λειτουργία με το ελάχιστο δυνατό κόστος. Η παρούσα διπλωματική εργασία επεκτείνει το μοντέλο Cost-Sensitive Fault Remediation (CSFR), το οποίο αναπτύχθηκε για τέτοιου είδους προβλήματα, με την προσθήκη ενός minimax κριτηρίου για σχεδιασμό και μάθηση και εφαρμόζει το νέο CSFR μοντέλο και τους αντίστοιχους αλγορίθμους στο πρόβλημα της αποκατάστασης δικτύου. Το minimax κριτήριο, σε αντίθεση με το κριτήριο ελαχίστου αναμενόμενου κόστους του αρχικού CSFR αλγορίθμου σχεδιασμού, έχει ως στόχο να προσφέρει μία ασφαλέστερη πολιτική που διατηρεί το μέγιστο (χειρότερο) κόστος σε χαμηλά επίπεδα. Ο σχεδιασμός με το minimax κριτήριο συμπληρώνεται με τρεις ευρετικές μεθόδους για την επίλυση πιθανών προβλημάτων κατά τη διάταξη των ενεργειών επιδιόρθωσης. Ο αρχικός αλγόριθμος σχεδιασμού και οι τρεις παραλλαγές του προτεινόμενου αλγορίθμου εφαρμόστηκαν σε προβλήματα εντοπισμού και αποκατάστασης βλαβών σε προσομοιωμένα δίκτυα διαφορετικών μεγεθών. Λαμβάνοντας υπόψη τους υπολογιστικούς περιορισμούς του πράκτορα σχεδιασμού, αναπτύχθηκε ένας επιπλέον πράκτορας με χρήση ενισχυτικής μάθησης στηριζόμενη σε στιγμιότυπα. Ο πράκτορας μάθησης δεν μπορεί να εγγυηθεί βελτιστότητα, όμως μπορεί να χειριστεί πολύ μεγαλύτερα προβλήματα αξιοποιώντας τα δεδομένα που συλλέγει κατά την αλληλεπίδραση με το περιβάλλον. Τα αποτελέσματά μας τόσο με το σχεδιασμό όσο και με τη μάθηση καταδεικνύουν ότι η προσέγγιση που επιλέχθηκε στηριζόμενη στο CSFR μοντέλο είναι σε θέση να χειριστεί αποτελεσματικά τα προβλήματα αποκατάστασης δικτύου. Σε αρκετές περιπτώσεις, ο προτεινόμενος αλγόριθμος σχεδιασμού με το minimax κριτήριο αποδίδει μέγιστο πραγματικό κόστος αποκατάστασης, το οποίο είναι χαμηλότερο σε σχέση με το μέγιστο πραγματικό κόστος του αρχικού CSFR αλγορίθμου σχεδιασμού που έχει ως κριτήριο το ελάχιστο αναμενόμενο κόστος.

Acknowledgements

I would like to take this opportunity to sincerely thank my thesis supervisor, Prof. Michail G. Lagoudakis for his support and guidance towards the completion of my diploma. I really appreciate his valuable help and understanding, and I am deeply thankful due to the fact that he has always spared his time to discuss and solve problems that presented during my implementation. Special thanks is also extended to Prof. Polychronis Koutsakis and Prof. Georgios Chalkiadakis for their valuable advice, suggestions and support to solve the problems arising during implementation. Furthermore I would like to express my gratitude to my parents for their continuous support, endless encouragement and confidence especially during my studies. Many thanks to my friends, who have been very helpful in giving me suggestions and moral support towards my studies and implementation of my diploma.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Μιχαήλ Γ. Λαγουδάκη για την υποστήριξη και την καθοδήγηση του κατά τη διάρκεια της διπλωματικής μου εργασίας. Εκτιμώ πραγματικά τη πολύτιμη βοήθεια και κατανόηση του και είμαι πραγματικά ευγνώμων γιατί πάντα διέθετε χρόνο για να συζητήσουμε και να λύσουμε τυχόν προβλήματα που παρουσιάζονταν κατά τη διάρκεια της υλοποίησης. Ένα μεγάλο ευχαριστώ στους καθηγητές Πολυχρόνη Κουτσάκη και Γεώργιο Χαλκιαδάκη για τις πολύτιμες συμβουλές, τις προτάσεις τους και την υποστήριξη τους στα προβλήματα που εμφανίζονταν κατά τη διάρκεια της υλοποίησης. Επιπλέον θα ήθελα να εκφράσω την ευγνωμοσύνη μου στους γονείς μου για την αδιάλειπτη υποστήριξη, την ενθάρρυνση και την εμπιστοσύνη που μου έδειξαν όλα αυτά τα χρόνια κατά τη διάρκεια των σπουδών μου. Ένα μεγάλο ευχαριστώ στους φίλους μου για τη βοήθεια τους, τις προτάσεις τους και την ηθική τους υποστήριξη κατά τη διάρκεια των σπουδών μου και την εκπόνηση της διπλωματικής μου.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Organizations and individuals are becoming more dependent on computer networks to accomplish their daily tasks. This fact implies that faults and time delays induce a significant cost. New technologies, applications, and service providers need a high level of services to maintain proper operation. As the number of users constantly increases, the stability and proper service of a network becomes a challenging task. Fault management is an important functional area in the Open Systems Interconnection (OSI) model of network management. The purpose of fault management is to detect and resolve faults occurring at nodes of the network. Fault detection is a domain of critical importance due to the fact that correct detection and diagnosis of a fault narrows the range of actions required in order to resolve the problem.

This thesis focuses on network fault detection and remediation. What exactly is a network and what does it consist of? A network can be described as a collection of workstations (computers) and other hardware interconnected by communication channels (links) that allow sharing of services, data, and information. The basic operation of each workstation is sending and receiving data to and from another device. The type of network considered in this thesis includes workstations, servers, hubs, and switches. A *workstation* is a high-end single-user computer designed to run technical or scientific applications. A *server* is a physical computer dedicated to run one or more services (as a host) to serve the needs of multiple users of other computers on the network. A *switch* is a networking device that links several workstations with each other and allows connections to other parts of a network through hubs. A *hub* is a networking device for connecting multiple Ethernet devices (servers, other hubs) together and making them

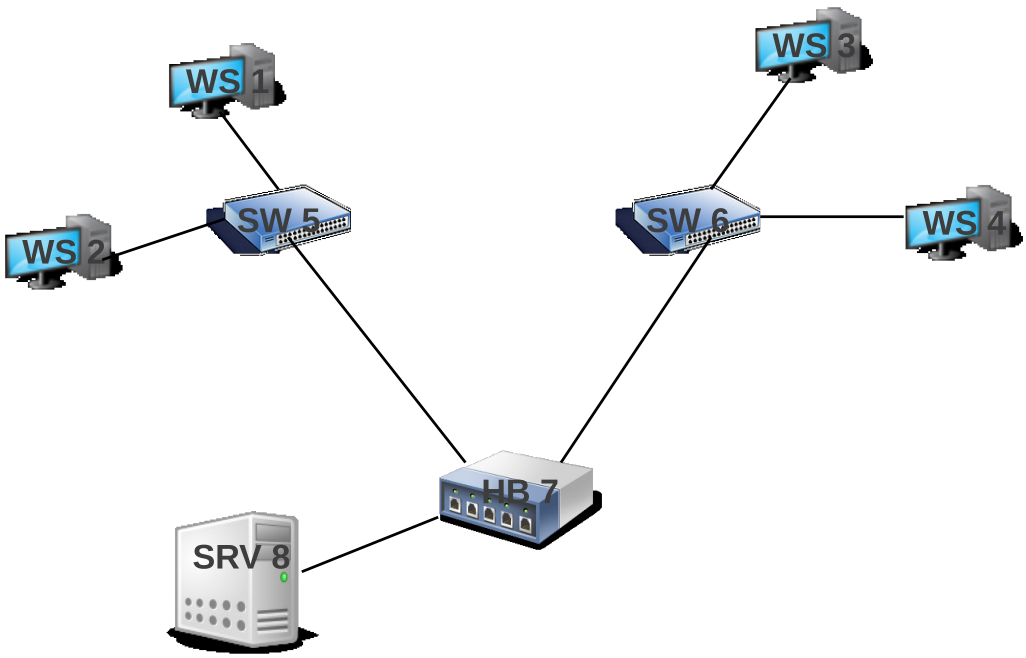


Figure 1.1: Network example

act as a single segment in the network. A schematic that gives an explicit view of the network type used in this thesis is shown in Figure 1.1.

The fault manager proposed in this research is an autonomous agent. An autonomous agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors [1]. The structure of an agent includes:

(a) **Percepts**: the information received by the sensors at each time.

(b) **Percept Sequence**: the complete history of everything the agent has ever perceived.

(c) **Actions**: actuation of the effectors (decisions) at each time.

(d) **Agent Function**: mapping from percept sequences to actions

Figure 1.2 shows how an agent interacts with the environment and receives feedback.

The benefits of viewing a fault manager as an autonomous agent are studied in this thesis in order to provide a method for efficient and dynamic network fault detection and remediation. Consider a network that includes switches, hubs, servers, and workstations. In such a network, workstations can communicate, through hubs and switches, with

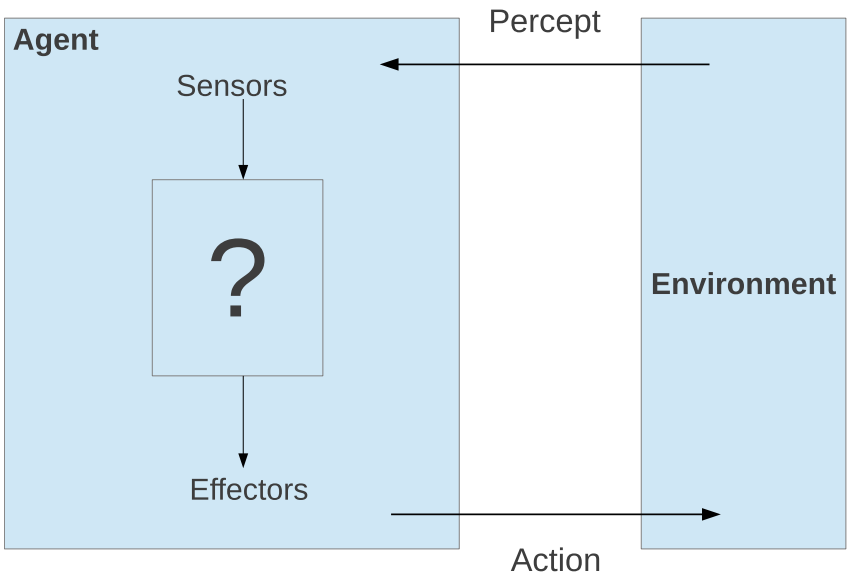


Figure 1.2: Autonomous Agent

servers or with each other. Occasionally, problems occur and nodes break down. A faulty node cannot participate in any communication activity in the network, nor can it relay any messages to the network. Detecting and identifying a fault is naturally a difficult task due to unordered arrival of information, topologies of networks, varying link sizes, and package congestion [2]. Intelligent systems are lately developed using techniques from Artificial Intelligence and Machine Learning towards dynamic, robust, and optimized network fault management. This research, in particular, uses planning under uncertainty [3] and reinforcement learning [4] techniques to cope effectively with the challenges of network fault detection and remediation.

1.1 Thesis Contribution

The purpose of this research is to provide an intelligent agent for detecting and repairing network faults. This agent overviews a network and has access to each node of the network. If the occurrence of a fault is signaled, the agent tries to determine the fault using a sequence of test actions. Finally, the agent remedies the fault by taking one or more repair actions. The architecture of the agent is based on the Cost-Sensitive Fault Remediation (CSFR) model. The original CSFR model has its own limitations, as it

relies on the criterion of expected cost minimization. For certain domains, where safe repair policies are preferable, this criterion may not be appropriate; in such domains, the agent could rely on the criterion of maximum cost minimization. The following objectives summarize the contribution of this research:

(a) Extend the CSFR planning algorithm to optimize a minimax criterion

(b) Enhance the minimax CSFR planning algorithm with repair action ordering heuristics

(c) Implement a simulated network environment, a flexible platform for experimentation

(d) Implement a CSFR planning algorithm in order to determine an optimal decision policy for diagnosis of network faults

(e) Implement an instance-based learning algorithm to learn a decision policy for diagnosis of network faults that improves with experience

(f) Evaluate the applicability and efficiency of the planning and learning algorithms

1.2 Thesis Layout

This thesis is organized into the following chapters:

Chapter 2 presents the original Cost-Sensitive Fault Remediation (CSFR) model, as well as the original planning and learning algorithms based on the expected cost criterion for total repair cost.

Chapter 3 presents our minimax version of CSFR and, more specifically, the minimax CSFR planning and learning algorithms based on the minimax criterion for total repair cost along with the repair action ordering heuristics.

Chapter 4 presents the problem of network fault repair and the details of our implementation for the network simulation and the platform used to test the CSFR-based approaches. The formation of states is described, as well as, the implementation of test and repair actions.

Chapter 5 shows how the CSFR model applies to the network fault detection and remediation problem. In particular, the full CSFR model (fault states, test and repair actions, cost function, observation model) for this domain is defined. Subsequently, the original and our extended CSFR planning algorithms are applied to find decision policies.

The entire procedure of fault detection and remediation is illustrated through detailed examples.

Chapter 6 describes a reinforcement learning approach to decision making within the CSFR model for network fault detection and remediation. According to this instance-based approach, a CSFR model is learned from experience. The difference between the true and the learned CSFR models lies mainly in the set of faulty states; in the learning approach the set of faulty states is formed dynamically through experience by storing past instances of fault detection and remediation.

Chapter 7 presents our experimental results highlighting the strengths and weaknesses of the proposed network fault detection and remediation approach. The chapter describes the experimental setup and presents experiments with both planning and learning under both criteria (expected and minimax cost) to make comparisons and draw conclusions.

Chapter 8 summarizes the results of this research, gives an evaluation of the achievement of our objectives, and describes the advantages and disadvantages of our method. Finally, it proposes directions for future work in the field of network repair from the perspective of planning and learning.

Chapter 2

Cost-Sensitive Fault Remediation

The Cost-Sensitive Fault Remediation (CSFR) model [3, 5, 6] is a modeling framework for partially-observable environments capturing diagnosis-and-repair situations. Under the CSFR model, an agent is able to detect the monitored system has entered a failure mode, execute a sequence of test actions to diagnose the fault, attempt to repair the fault by executing repair actions, and finally detect if the monitored system has been restored to normal operation. A CSFR model is defined by a set of fault states, a set of test and a set of repair actions, a cost function, and an observation model. The CSFR planning algorithm for optimal decision policies is implemented via dynamic programming and calculates a minimum-cost repair policy to detect and remedy the fault. On the other hand, the reinforcement learning approach to decision making within the CSFR model forms the set of faulty states dynamically through experience by storing past instances of fault detection and remediation.

2.1 CSFR Fault Manager

Fault detection and remediation within the CSFR approach can be more comprehensible in terms of some basic components, as shown in Figure 2.1. The environment gives information to the CSFR-based fault manager and the manager eventually gives feedback to the environment by commanding actions, one at a time. In fact, the fault manager is an agent that operates on behalf of the user and aims to identify and repair faults of the system by deciding which test or repair action to execute. By assumption, according to the CSFR approach, only one fault is present at each time; whenever a fault occurs,

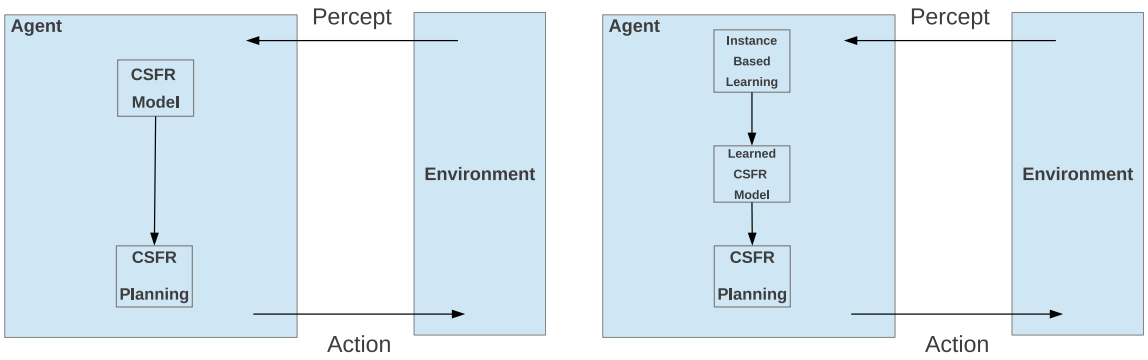


Figure 2.1: Planning fault manager (left) and Learning fault manager (right)

the manager is signaled to initiate a repair procedure without knowing where the fault is. After the system is restored to normal operation, another fault may occur at a later time signaling the manager and so on.

The fault manager has two alternative modes of operation: planning and learning. Planning uses the CSFR planning algorithm on the true CSFR model to calculate an optimal-cost repair policy in order to diagnose and repair the fault. On the other hand, learning uses an instance-based learning approach to form a learned CSFR model. In fact, the manager runs episodes without initially knowing the kinds of faults that occur in the environment. Each one of the collected episodes is stored in the learned model, so that it can be used in the future as a representative instance of some kind of fault for which the stored repair action can be re-used, if that instance is identified again. In that sense, the manager based on learning improves its effectiveness over time, while avoiding to consider the entire range of all possible faults as done by the manager based on planning. This difference will become clear in the rest of this chapter.

2.2 CSFR Model

The full definition of a CSFR model is given by the following items:

· S**, the set of fault states,** representing all possible faults that may occur

· $Pr(s)$**, a prior probability for each fault state** $s \in S$**.** These probability values are provided by the user and can be extracted by observation of the environment.

· A_T, **the set of test actions,** which are used to gain information in order to detect and diagnose the current fault in the environment.

· A_R, **the set of repair actions,** which are used to resolve specific faults in the environment, if present.

· **a cost function** $c(s, a)$ **over actions** $a \in A_T \cup A_R$ **and states** $s \in S$**.** The numeric cost of an action depends on the state it is taken in. This cost is typically related to time, however this is not restricted as the model could adjust to a large range of problems.

· **a deterministic observation model** $o(s, a)$ **over actions** $a \in A_T \cup A_R$ **and states** $s \in S$**.** The output of the observation model is binary, `0` or `1`. For repair actions, `0` is interpreted to mean that the repair is unsuccessful in state s and the fault remains, while `1` means that the repair action resolves the problem in state s and the fault is eliminated. For test actions, the outcome, `0` or `1`, depends on the current state s, is defined by the user according to the nature of the problem, and can be used to identify the actual current state.

It is clear from the model, that a cost is incurred for any action performed, either test or repair action. Apparently, the total cost during a detection and repair case will be the main decision parameter. The goal is to detect and eliminate the fault and restore normal operation, while paying the lowest possible cost. To this end, test and/or repair actions must be chosen carefully. The fault manager acts as a decision-maker facing a sequential decision making problem. Initially, a sequence of test actions will help pinpoint the fault followed by some repair action to restore operation. These actions will be chosen by a minimum-cost policy. The original CSFR model assumes that the test actions are indicative enough to uniquely identify the fault.

2.3 CSFR Planning

Since the true fault state is unknown the agent has to reason over sets of possible fault states, known as beliefs. If S is the set of possible fault states, the beliefs over fault states is the power set of S, 2^S. Let $Q(b, a)$ be the expected total cost until the fault is repaired, when the current belief is $b \in 2^S$ and the action taken at the current step is $a \in A_T \cup A_R$.

An optimal repair policy in a CSFR model can be extracted by taking at each step those actions that minimize the expected total cost for the current belief. The expected total cost function Q can be computed via dynamic programming using the following recursive equation [5]:

$$Q(b,a) = \begin{cases} \dfrac{Pr(b_0)}{Pr(b)}\Big(c(b_0,a) + V(b_0)\Big) + \dfrac{Pr(b_1)}{Pr(b)}\Big(c(b_1,a) + V(b_1)\Big), & \text{if } Pr(b_0) > 0, Pr(b_1) > 0 \\ & \text{or } Pr(b_1) > 0, a \in A_R \\ \infty, & \text{otherwise} \end{cases}$$

where $b_i = \{s \in b \mid o(s,a) = i\}$ is the belief state resulting from taking action a in belief state b and obtaining outcome $i \in \{\texttt{0}, \texttt{1}\}$. $V(b)$ is the minimum expected total cost until the fault is repaired, when the current belief is $b \in 2^S$. If $a \in A_R$, for $i = \texttt{1}$ we define $V(b_1) = 0$, as there is no additional cost incurred once a repair action is successful. Essentially, belief b_1 becomes an empty set in this case. In all other cases, the value of a belief state is the minimum action value taken over all available action choices:

$$V(b) = \min_{a \in A_T \cup A_R} Q(b,a)$$

$Pr(b)$ and $c(b,a)$ are the prior probability and the cost function extended to belief states. More specifically, $Pr(b) = \sum_{s \in b} Pr(s)$, therefore the prior probability of a belief b is the sum of the prior probabilities of the states it contains. Also, $c(b,a) = \sum_{s \in b} Pr(s)\ c(s,a)$, therefore the cost of taking some action a in some belief b is computed by the costs of taking action a in each of the states contained in b weighted by the corresponding prior probabilities of those states. The condition

$$\Big(Pr(b_0) > 0 \text{ and } Pr(b_1) > 0\Big) \text{ or } \Big(Pr(b_1) > 0 \text{ and } a \in A_R\Big)$$

in the recursion ensures that the recursion eventually ends, since no quantity is defined in terms of itself. This restriction rules out only clearly suboptimal policies that include actions that do not alter the belief state and guarantees that the algorithm can be implemented without considering cyclic dependencies. In other words, the condition is satisfied

Algorithm 1 CSFR Value Computation Algorithm

function GETVALUE(environment env, belief b)
  Returns: best value V_{best}, best action a_{best}
  if $b = \emptyset$ **then** ▷ empty belief
    $V_{\text{best}} \leftarrow 0$
  else
    $V_{\text{best}} \leftarrow \infty$
    for all $a \in A_T \cup A_R$ **do** ▷ check all actions
      $b_0 \leftarrow \{s \in b \mid o(s,a) = \texttt{0}\}$
      $b_1 \leftarrow \{s \in b \mid o(s,a) = \texttt{1}\}$
      $Pr(b_0) \leftarrow \sum_{s \in b_0} Pr(s)$
      $Pr(b_1) \leftarrow \sum_{s \in b_1} Pr(s)$
      if $\Big((Pr(b_0) > 0)\ \&\ (Pr(b_1) > 0)\Big) \ \Big|\ \Big((Pr(b_1) > 0)\ \&\ (a \in A_R)\Big)$ **then**
        $Pr(b) \leftarrow \sum_{s \in b} Pr(s)$
        $c(b_0, a) \leftarrow \sum_{s \in b_0} Pr(s)\ c(s,a)$
        $c(b_1, a) \leftarrow \sum_{s \in b_1} Pr(s)\ c(s,a)$
        $V(b_0) \leftarrow$ GETVALUE(env, b_0)
        if $a \in A_R$ **then**
          $b_1 \leftarrow \{\}$ ▷ the repair action fixes these fault states
        end if
        $V(b_1) \leftarrow$ GETVALUE(env, b_1)
        $Q(b,a) \leftarrow \frac{Pr(b_0)}{Pr(b)}\Big(c(b_0,a) + V(b_0)\Big) + \frac{Pr(b_1)}{Pr(b)}\Big(c(b_1,a) + V(b_1)\Big)$
      else
        $Q(b,a) \leftarrow \infty$
      end if
      if $Q(b,a) < V_{\text{best}}$ **then**
        $a_{\text{best}} \leftarrow a$ ▷ action that minimizes the expected cost
        $V_{\text{best}} \leftarrow Q(b,a)$ ▷ minimum expected cost
      end if
    end for
  end if
  return $V_{\text{best}}, a_{\text{best}}$
end function

only when an action splits the current belief into two beliefs with non-zero probability or a repair action yields a belief b_1 with a non-zero probability, therefore it fixes at least one state in the current belief. Due to the nature of this condition, there is an implicit assumption in CSFR that the prior probabilities of fault states cannot be zero, since a

Algorithm 2 CSFR Planning Agent

```
1: function CSFRPLANNING(environment env)
2:     Returns: total repair cost totalCost
3:
4:     totalCost ← 0
5:     belief ← S
6:     while belief ≠ ∅ do
7:         {V_best, a_best} ← GETVALUE(env, belief)
8:         [outcome, cost] ← EXECUTEACTION(env, a_best)
9:         totalCost ← totalCost + cost
10:        if (a_best ∈ A_R) & (outcome = 1) then        ▷ fault was successfully removed
11:            belief ← ∅
12:        else if a_best ∈ A_T then
13:            belief ← {s ∈ belief | o(s, a_best) = outcome}
14:        else if a_best ∈ A_R then
15:            belief ← {s ∈ belief | o(s, a_best) = 0}
16:        end if
17:    end while
18:    return totalCost
19: end function
```

fault that never occurs is not a fault! However, if zero prior probabilities are given for any reason, the condition above may fail to get satisfied. Algorithm 1 shows the main CSFR function that computes expected costs (V values) recursively for any belief state.

Initially, the belief state includes all possible fault states. The agent chooses the actions that minimize the expected total cost V at each step. These are typically test actions, which help reduce the size of the belief state to locate the fault, however repair actions may be executed in early steps, if they are likely to succeed. The reduced belief state at each step will be determined by the outcome of most recent action. As the belief state of the agent shrinks gradually, eventually the correct repair action will be executed restoring the system to normal operation. The correct repair action may be executed after a series of actions have brought the belief state down to a singleton set and the fault state has been precisely identified, but this is not always the case. Early repair is possible, if the correct repair action is picked because it minimizes the expected cost in some non-singleton belief. If the agent ever reaches a non-singleton belief state where no splits are possible by further test actions and therefore repair actions must be

applied, the ordering of these repair actions will be determined by a balancing of cost and likelihood (prior probabilities) in the sense of expect cost. Algorithm 2 summarizes the internal functionality of the CSFR planning agent. It should noted that the best values and actions can either be computed once and stored explicitly in memory or be computed again and again using function GETVALUE on the current belief state at each step of an episode.

2.4 CSFR Learning

Apart from planning, CSFR offers an instance-based reinforcement learning approach which creates and manipulates a CSFR model based on unstructured interaction with the environment. More specifically, the set of fault states is constructed gradually from examples of complete repair episodes. Each complete repair episode contains a number of test and repair actions taken during the episode along with their outcomes and costs and ends with a successful repair action that fixed the fault. Due to the nature of the CSFR model, the exact ordering of actions tried during the episode has no significant meaning. Likewise, due to the deterministic nature of costs and observations, there is no repetition of actions. In a sense, each such episode represents a particular type of fault (a fault state), which can remedied by executing the stored successful repair action from that episode. In practice, in CSFR learning, the set of faulty states S is replaced by a set of episodes E.

Initially, before any repair attempt, the set of episodes E may be empty, but new episodes may be added at any time, especially when facing a type of fault which cannot be matched to any of the stored episodes. If the presence of fault is signaled, the learning fault manager will try to fix the fault by running CSFR planning on the existing set of episodes. If the current fault is matched against some stored episode (fault state) and the fault is fixed, then the set of episodes E remains unchanged. If, however, CSFR planning ends with an empty belief and the fault has not been fixed, a new episode has to be added. This new episode contains already the actions tried during the failed CSFR planning attempt and will be extended by additional random action selection (over the actions that have not been tried already) until the fault is fixed. Actions should not be repeated during an episode, since observations and costs are deterministic. In the worst case, all actions will be tried, but it is guaranteed that the fault will eventually be fixed.

Algorithm 3 CSFR Learning Agent

function CSFRLEARNING(environment env, set of episodes E)
    Returns: total repair cost $totalCost$, updated set of episodes E

    $fixed \leftarrow$ `false`
    $totalCost \leftarrow 0$
    $e \leftarrow \emptyset$
    $\forall a \in A_T \cup A_R : \ c(e,a) \leftarrow 0$
    $\forall a \in A_T \cup A_R : \ o(e,a) \leftarrow$ `?`
    $belief \leftarrow E$
    while $fixed =$ `false` **do**
        if $belief \ \neq \ \emptyset$ **then**
            $\{V_{\text{best}}, a_{\text{best}}\} \leftarrow$ GETVALUE$(env, belief)$
        else
            $a_{\text{best}} \leftarrow$ choose randomly from $\{a \in A_T \cup A_R : o(e,a) =$ `?`$\}$
        end if
        $[outcome, cost] \leftarrow$ EXECUTEACTION(env, a_{best})
        $totalCost \leftarrow totalCost + cost$
        $e \leftarrow e \cup \{a_{\text{best}}\}$
        $c(e, a_{\text{best}}) \leftarrow cost$
        $o(e, a_{\text{best}}) \leftarrow outcome$
        if $(a_{\text{best}} \in A_R)$ & $(outcome =$ `1`$)$ **then** ▷ fault was successfully fixed
            if $belief \ \neq \ \emptyset$ **then** ▷ fixed by existing episode
                $\forall e_s \in belief : \ count(e_s) \leftarrow count(e_s) + 1$
            else ▷ fixed by the new episode
                $count(e) \leftarrow 1$
                $E \leftarrow E \cup e$ ▷ add the new episode to E
            end if
            $fixed \leftarrow$ `true`
        else if $a_{\text{best}} \ \in A_T$ **then**
            $belief \leftarrow \{e_s \in belief \mid o(e_s, a_{\text{best}}) = outcome$ or `?`$\}$
        else if $a_{\text{best}} \ \in A_R$ **then**
            $belief \leftarrow \{e_s \in belief \mid o(e_s, a_{\text{best}}) =$ `0` or `?`$\}$
        end if
    end while
    $\forall e \in E : \ Pr(e) = \dfrac{count(e)}{\sum_{e_s \in E} count(e_s)}$
    return $totalCost, E$
end function

The observation model and the cost function for each new episode are defined implicitly by the outcomes and the costs observed and recorded during the course of the episode. Note that since not all actions may be tried during an episode, the gaps in the observation model and the cost function are filled with the wildcard `?` (matching both observations `0` and `1`) and a numeric value of 0 respectively. Finally, the prior probabilities over the stored episodes are formed by the frequency they are matched and lead to successful repair over multiple cycles of fault occurrence and repair. Algorithm 3 shows the details of the learning agent.

Given the above learned (estimated) CSFR model, decision making boils down to CSFR planning on the estimated model and the learning fault manager proceeds similarly to the planning one. The belief states are now formed by subsets of instances (episodes). While optimal decision policies cannot be guaranteed in this case, it is expected that the actual episodes encountered and stored during the course of learning will be much smaller compared to the set of all possible faulty states in the corresponding planning problem. Thus, the lack of optimality is counterbalanced by significant savings in execution time.

Chapter 3

MiniMax CSFR

The planning algorithm for the CSFR model presented in Chapter 2 focuses on the *minimum expected* cost optimality criterion, which aims to find an optimal policy for the *average* case. In practice, this means that the performance of the agent will be optimal, if viewed as the average repair cost over many fault remediation episodes. This expected cost criterion, however, suffers from two facts. First, it depends highly on the accuracy of the prior probabilities; if the true probabilities are significantly different, then the agent will optimize its choices for a non-existent average case. Second, if the fault remediation episodes are scarce, it may be preferable to follow a safe policy regardless of what the true or estimated average case is. Such safe, yet conservative, policies may be obtained by the well-established *minimax* criterion, which focuses on the *worst* case. In practice, this means that the agent will try to keep the worst-possible total repair cost low, even during a single remediation episode.

In this chapter we describe a minimax version version of CSFR planning and learning, which utilizes the same CSFR model described already in Section 2.2, but alters the optimality criterion for decision making. Since the minimax version of CSFR uses the same CSFR model definition, we focus only on decision making through planning or learning. In addition to the minimax formulation, we offer three heuristics for ordering repair actions in situations where the minimax criterion gives no preference. The minimax criterion for fault remediation may be more appropriate in certain applications, where it is critical to guarantee that the total repair cost will remain below certain levels.

3.1 MiniMax CSFR Planning

Once again, since the true fault state is unknown, the agent has to reason over sets of possible fault states or beliefs. Let $\widehat{Q}(b,a)$ be the minimax total cost until the fault is repaired, when the current belief is $b \in 2^S$ and the action taken at the current step is $a \in A_T \cup A_R$. An optimal repair policy in a minimax CSFR model can be extracted by taking at each step those actions that minimize the maximum (worst) total cost for the current belief. The maximum total cost function $\widehat{Q}$ can be computed via dynamic programming using the following recursive equation:

$$\widehat{Q}(b,a) = \begin{cases} \max\Big\{\big(\widehat{c}(b_0,a) + \widehat{V}(b_0)\big), \big(\widehat{c}(b_1,a) + \widehat{V}(b_1)\big)\Big\}, & \text{if } |b_0| > 0, |b_1| > 0 \\ & \text{or } |b_1| > 0, a \in A_R \\ \infty, & \text{otherwise} \end{cases}$$

where $b_i = \{s \in b \mid o(s,a) = i\}$ is the belief state resulting from taking action a in belief state b and obtaining outcome $i \in \{\mathtt{0},\mathtt{1}\}$. $\widehat{V}(b)$ is the minimax total cost until the fault is repaired, when the current belief is $b \in 2^S$. If $a \in A_R$, for $i = \mathtt{1}$ we define $\widehat{V}(b_1) = 0$, as there is no additional cost incurred once a repair action is successful. Essentially, belief b_1 becomes an empty set in this case. In all other cases, the value of a belief state is the minimum action value taken over all available action choices:

$$\widehat{V}(b) = \min_{a \in A_T \cup A_R} \widehat{Q}(b,a)$$

As before, $\widehat{c}(b,a)$ is the cost function extended to belief states, however this time in the minimax sense. More specifically, $\widehat{c}(b,a) = \max_{s \in b} c(s,a)$, therefore the cost of taking some action a in some belief b is computed by the maximum cost of taking action a over all states contained in b. The condition

$$\Big(|b_0| > 0 \text{ and } |b_1| > 0\Big) \text{ or } \Big(|b_1| > 0 \text{ and } a \in A_R\Big)$$

in the recursion ensures that the recursion eventually comes to an end, since no quantity is defined in terms of itself. In other words, the condition is satisfied only when an action splits the current belief into two non-empty (smaller) beliefs or a repair action fixes at

Algorithm 4 MiniMax CSFR Value Computation Algorithm

function GETMINIMAXVALUE(environment env, belief b)
  Returns: best value $\widehat{V}_{\text{best}}$, best action a_{best}
  if $b = \emptyset$ **then** ▷ empty belief
    $\widehat{V}_{\text{best}} \leftarrow 0$
  else
    $\widehat{V}_{\text{best}} \leftarrow \infty$
    for all $a \in A_T \cup A_R$ **do** ▷ check all actions
      $b_0 \leftarrow \{s \in b \mid o(s,a) = \texttt{0}\}$
      $b_1 \leftarrow \{s \in b \mid o(s,a) = \texttt{1}\}$
      $Pr(b_0) \leftarrow \sum_{s \in b_0} Pr(s)$
      $Pr(b_1) \leftarrow \sum_{s \in b_1} Pr(s)$
      if $\Big(\big((|b_0| > 0)\ \&\ (|b_1| > 0)\big) \ \Big|\ \big((|b_1| > 0)\ \&\ (a \in A_R)\big)\Big)$ **then**
        $\widehat{c}(b_0, a) \leftarrow \max_{s \in b_0} c(s,a)$
        $\widehat{c}(b_1, a) \leftarrow \max_{s \in b_1} c(s,a)$
        $\widehat{V}(b_0) \leftarrow$ GETMINIMAXVALUE(env, b_0)
        if $a \in A_R$ **then**
          $b_1 \leftarrow \{\}$ ▷ the repair action fixes these fault states
        end if
        $\widehat{V}(b_1) \leftarrow$ GETMINIMAXVALUE(env, b_1)
        $\widehat{Q}(b,a) \leftarrow \max\Big\{\big(\widehat{c}(b_0,a) + \widehat{V}(b_0)\big), \big(\widehat{c}(b_1,a) + \widehat{V}(b_1)\big)\Big\}$
      else
        $\widehat{Q}(b,a) \leftarrow \infty$
      end if
      if $\widehat{Q}(b,a) < \widehat{V}_{\text{best}}$ **then**
        $a_{\text{best}} \leftarrow a$ ▷ action that minimizes the expected cost
        $\widehat{V}_{\text{best}} \leftarrow \widehat{Q}(b,a)$ ▷ minimum expected cost
      end if
    end for
  end if
  return $\widehat{V}_{\text{best}}, a_{\text{best}}$
end function

least one state in the current belief and therefore yields a smaller belief in the next step. Note the slightly different formulation of this condition compared to the original CSFR algorithm. We check the sizes of the resulting beliefs as opposed to their probabilities; this choice was preferred to avoid singularities occurring when a belief b_i is non-empty, yet it has a zero probability $Pr(b_i)$ due to a possible wrong set of prior probabilities that

assigns zero probability values to some fault states. Algorithm 4 shows the main minimax CSFR function that computes minimax costs ($\widehat{V}$ values) recursively for any belief state.

The minimax CSFR fault manager proceeds exactly as the original one presented in Algorithm 2; the only difference is the call to function GETMINIMAXVALUE instead of GETVALUE in Line 7. Initially, the belief state includes all possible fault states, the entire set S. The agent chooses the actions that minimize the maximum total cost $\widehat{V}$ at each step. These are typically actions, which help reduce the size of the belief state to locate the fault in ways that guarantee that the worst cost will remain small. The reduced belief state at each step is determined by the outcome of the most recent action. As the belief state of the agent shrinks gradually, eventually the correct repair action will be executed restoring the system to normal operation. The correct repair action may be executed after a series of actions have brought the belief state down to a singleton set and the fault state has been precisely identified, but this is not always the case. Early repair is possible, if the correct repair action is picked earlier, because it minimizes the maximum cost in some non-singleton belief. As in the original CSFR, it should noted that the function $\widehat{V}$ can either be computed once and stored explicitly in memory or be computed again and again recursively for the current belief state during an episode.

3.2 MiniMax CSFR Repair Action Ordering

If the agent during a remediation episode ever reaches a non-singleton belief state where no further belief splits are possible by test actions and therefore repair actions must be applied, the ordering of these repair actions is indifferent under the minimax criterion. Any ordering of the appropriate repair actions yields the same maximum cost. We call such beliefs *repair-only beliefs*. While any ordering of repair actions in a repair-only belief guarantees the same maximum cost, in practice it makes sense to impose some heuristic ordering which is likely to reduce the actual cost. Repair-only beliefs can be easily identified, since all test actions in such beliefs induce an infinite cost as they cannot offer a split. Algorithm 5 shows the variation of minimax CSFR (see lines 27–29) that incorporates the detection of repair-only beliefs and calls another function that determines the ordering of repair actions to figure out which repair action will be executed first in the current belief.

Algorithm 5 MiniMax CSFR Value Computation Algorithm with Repair Ordering

function GETMINIMAXVALUE(environment env, belief b)
  Returns: best value $\widehat{V}_{\text{best}}$, best action a_{best}
  if $b = \emptyset$ **then** ▷ empty belief
    $\widehat{V}_{\text{best}} \leftarrow 0$
  else
    $\widehat{V}_{\text{best}} \leftarrow \infty$
    for all $a \in A_T \cup A_R$ **do** ▷ check all actions
      $b_0 \leftarrow \{s \in b \mid o(s,a) = \texttt{0}\}$
      $b_1 \leftarrow \{s \in b \mid o(s,a) = \texttt{1}\}$
      if $\Big((|b_0| > 0)\ \&\ (|b_1| > 0)\Big) \ \Big|\ \Big((|b_1| > 0)\ \&\ (a \in A_R)\Big)$ **then**
        $\widehat{c}(b_0, a) \leftarrow \max_{s \in b_0} c(s,a)$
        $\widehat{c}(b_1, a) \leftarrow \max_{s \in b_1} c(s,a)$
        $\widehat{V}(b_0) \leftarrow$ GETMINIMAXVALUE(env, b_0)
        if $a \in A_R$ **then**
          $b_1 \leftarrow \{\}$ ▷ the repair action fixes these fault states
        end if
        $\widehat{V}(b_1) \leftarrow$ GETMINIMAXVALUE(env, b_1)
        $\widehat{Q}(b,a) \leftarrow \max\Big\{\big(\widehat{c}(b_0,a) + \widehat{V}(b_0)\big), \big(\widehat{c}(b_1,a) + \widehat{V}(b_1)\big)\Big\}$
      else
        $\widehat{Q}(b,a) \leftarrow \infty$
      end if
      if $\widehat{Q}(b,a) < \widehat{V}_{\text{best}}$ **then**
        $a_{\text{best}} \leftarrow a$ ▷ action that minimizes the expected cost
        $\widehat{V}_{\text{best}} \leftarrow \widehat{Q}(b,a)$ ▷ minimum expected cost
      end if
    end for
    if $\forall a \in A_T :\ \widehat{Q}(b,a) = \infty$ **then** ▷ a repair-only belief has been reached
      $a_{\text{best}} =$ CHOOSEREPAIRACTION(env, b)
    end if
  end if
  return $\widehat{V}_{\text{best}}, a_{\text{best}}$
end function

Two heuristics that can be applied in repair belief states to impose some good ordering are the *cheapest-first* heuristic (the repair action with the minimum cost is applied first) and the *likeliest-first* heuristic (the repair action fixing the fault state with the highest prior probability goes first). These two heuristic focus exclusively on cost or probability.

Algorithm 6 Heuristics for Repair Action Selection

```
1: function chooseRepairAction(environment env, belief b)
2:     Returns: chosen action a_first
3:
4:     /* Cheapest-First Heuristic */
5:     ĉ_min ← ∞
6:     for all a ∈ A_R do                                   ▷ check all repair actions
7:         b_1 ← {s ∈ b | o(s, a) = 1}
8:         if |b_1| > 0 then                  ▷ consider only those that fix some fault(s)
9:             ĉ(b_1, a) ← max_{s∈b_1} c(s, a)
10:            if ĉ(b_1, a) < ĉ_min then
11:                a_first ← a                    ▷ choose an action that minimizes cost
12:                ĉ_min ← ĉ(b_1, a)
13:            end if
14:        end if
15:    end for
16:
17:    /* Likeliest-First Heuristic */
18:    P̂r_max ← 0
19:    for all a ∈ A_R do                                   ▷ check all repair actions
20:        b_1 ← {s ∈ b | o(s, a) = 1}
21:        if |b_1| > 0 then                  ▷ consider only those that fix some fault(s)
22:            Pr(b_1) ← Σ_{s∈b_1} Pr(s)
23:            if Pr(b_1) > P̂r_max then
24:                a_first ← a            ▷ choose an action that maximizes likelihood
25:                P̂r_max ← Pr(b_1)
26:            end if
27:        end if
28:    end for
29:
30:    /* Least-Expected-Cost-First Heuristic */
31:    (V, a_first) ← getValue(env, b)        ▷ choose action using the original CSFR
32:
33:    return a_first
34: end function
```

Yet another choice that balances these two parameters is to resort on the expected cost criterion of the original CSFR which balances cost and probability. According to this *least-expected-cost-first* heuristic the action that minimizes the expected (repair) cost

goes first. All three heuristics are shown in Algorithm 6 within the same function for completeness; in practice only one of them should be used.

3.3 MiniMax CSFR Learning

Since learning in the context of CSFR is model-based, the minimax version of the CSFR learning agent will be identical the original CSFR learning agent shown in Algorithm 3 the only difference being the call to function GETMINIMAXVALUE instead of GETVALUE in Line 12.

Chapter 4

Network Repair

This chapter introduces the domain of network fault repair (detection and remediation) and describes our network simulation which supports the definition of fault states and the execution of test and repair actions.

4.1 Network Repair

A computer network is a collection of computers and other hardware devices interconnected by communication channels (links) that allow sharing of services, data, and information. The basic operation of each workstation is sending and receiving data to and from another device. The type of network considered in this thesis includes the following types of devices:

workstation (WS) a high-end computer designed to run technical or scientific applications typically for a single user

server (SRV) a physical computer dedicated to run one or more services (as a host) to serve the needs of multiple users of other computers on the network

switch (SW) a networking device that links several workstations with each other and allows connections to other parts of a network through hubs

hub (HB) a networking device for connecting multiple Ethernet devices (servers, other hubs) together and making them act as a single segment in the network

Consider a network that includes switches, hubs, servers, and workstations such as the one shown in Figure 4.1. In such a network, workstations can communicate, through hubs and switches, with servers or with each other. Occasionally, problems occur and nodes break down. A faulty node cannot participate in any communication activity in the network, nor can it relay any messages to the network. Such faults may cause communication failures, especially if they appear on critical nodes that break unique communication paths in the network. A system administrator typically repairs a faulty node by initiating a node `reboot` which restores the node to normal operation paying a time cost proportional to the time of initiating and completing the boot operation. Detecting and identifying a faulty node is a difficult task and may require the execution of several tests over the network to determine which parts of the network are functional and which parts fail. The most common test for network connectivity is the execution of a `ping` command between two computer nodes (workstation or server). The `ping` initiates from one node and gets a reply back from the other node, if a functional communication path between the two nodes exists and its (time) cost in this case is proportional to the distance of the two nodes in the network in terms of network hops. If the communication path between the two nodes is broken, the `ping` initiates from one node, but fails to get a reply from the other node and its (time) cost is partly proportional to the distance of the two nodes in the network and partly proportional to a timeout for no reply which may depend on the the type of the broken node. In our implementation, we assume that this timeout is equal to the boot time of the broken device.

Our goal in this thesis is to automate the procedure of network fault detection and remediation through the use of an intelligent agent able to execute carefully-selected test and repair actions aiming to restore the network to normal operation while minimizing the total cost for repair. To this end we are using the CSFR framework to cope effectively with the challenges of sequential decision making for network fault detection and remediation.

4.2 Network Simulation

Any virtual network specified by the user of the type described above can be simulated for the needs of this thesis. Our simulation framework was developed using the `C++` programming language. Nodes of the network are represented using structures of different

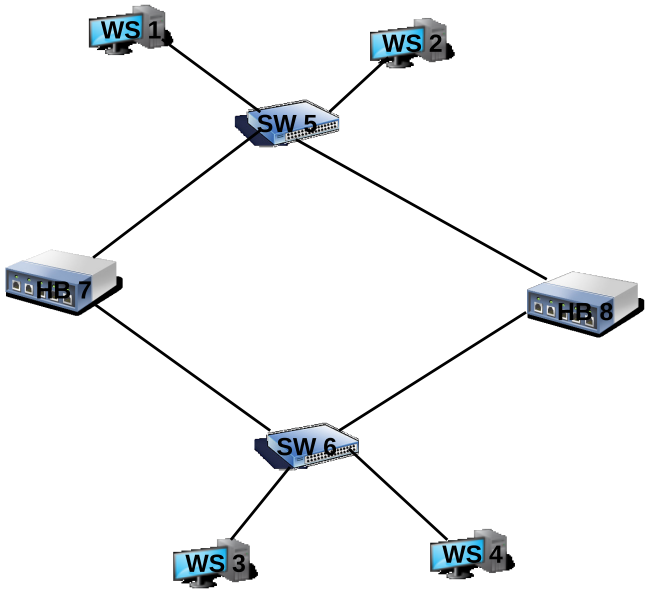


Figure 4.1: An example of the networks considered in this thesis

type for servers, workstations, switches, and hubs. These structures include several features: a unique ID to identify the node, a vector indicating the links of that node to other nodes, and a boolean variable indicating the operational status of the node (`false`:node broken down, `true`:node working properly). These structures are critically useful, as they are the main components of the network and can be used to create any desired network as an undirected graph. Users can create a new network or modify an existing one just by providing or modifying the text file that specifies the network.

The agent implementing the fault manager overviews the network and can be informed about the presence (or the elimination) of a fault in the network, as well as initiate any test or repair action on the network. It is assumed that faults occur only on nodes, not links. Signaling the presence of a fault contains no identifying information about the faulty node. One way to automatically detect that there is a fault in the network is to test if all `ping` commands between pairs of workstations/servers are successful. While this is sufficient in acyclic networks (all nodes along any path must operate normally), it fails in networks with cycles. In the example shown in Figure 4.1, if HB 8 is faulty, the `ping` between WS1 and WS3 will still be successful due to the alternative path through HB7 and the presence of a fault may not be detected. Therefore, in networks containing

Algorithm 7 Ping/Test Action - Modified Breadth First Search

```
1: for all u ∈ V − {s} do                    ▷ initialize all vertices in V[G] except source
2:     color[u] ← WHITE
3:     depth[u] ← ∞
4:     path[u] ← NULL
5: end for
6: color[s] ← GRAY                           ▷ initialize the source vertex
7: depth[s] ← 0
8: path[s] ← NULL
9: Q ← {}                                    ▷ Clear queue Q
10: Enqueue(Q, s)                            ▷ begin with the source
11: while Q is non-empty do
12:     u ← Dequeue(Q)
13:     for all v adjacent to u do
14:         if color[v] = WHITE then
15:             color[v] ← GRAY
16:             depth[v] ← depth[u] + 1
17:             path[v] ← u
18:             if operational status of v is true then
19:                 Enqueue(Q, v)
20:             end if
21:         end if
22:     end for
23:     Dequeue(Q)
24:     color[u] ← BLACK
25: end while
```

cycles, the presence of a fault must be signaled by another "omni-present" mechanism which essentially computes the `AND` of all node status variables and informs the fault manager if the result is `false`.

Faults are easily simulated in our network; the operational status of a node simply changes from `true` to `false`. Our simulator supports the repetitive (sequential) introduction of faults according to any given distribution. Similarly, the execution of `reboot` commands boils down to changing the operational status of a node from `false` to `true`. The simulation of `ping` commands is a little mode involved and is implemented using a modified Breadth First Search (BFS) algorithm. The BFS search begins at the node initiating the ping and proceeds towards all available directions in a breadth-first manner

using the time cost per hop as cost function and making sure that no cycles are explored. The modification introduced relates to fault node. Faulty nodes are not enqueued in the priority queue and therefore paths stop there. If there exists a path between the initiating and the target node of the ping, the search algorithm will return the shortest (in terms of ping time cost) of these paths. If no such path exists (due to some fault), the search algorithm returns only a partial shortest path which ends at the faulty node. The details of the modified BFS algorithm are shown in Algorithm 7.

Chapter 5

CSFR Planning for Network Repair

In this chapter, we discuss how CSFR planning is applied to the problem of network fault detection and remediation. A definition of the CSFR model for this problem is important. Additionally,an example along with a description of the solution given unravels the way CSFR is applied to our problem. Assume a network that contains hubs, switches, servers, and workstations. Each of workstations and servers can communicate through hubs and switches, with each other. The communication is achieved by the network operation "ping". Occasionally, a node of the network breaks down. As a result of that, the node cannot communicate with other nodes or even worse, other nodes cannot communicate with each other through the fault node. The agent, designed for such circumstances, based on Cost-Sensitive Fault Remediation model. The agent declares each fault node as distinguished fault state and uses pings as test actions and reboots of nodes as repair actions in order to diagnose and remedy the fault node of the network respectively.

5.1 CSFR Model for Network Repair

The full definition of a CSFR model for the network fault detection and remediation problem takes the following form:

· S**, the set of fault states.** Each node in the network can possibly break down. Hence, the total number of fault states is the total number of nodes in the network:

$$|S| = |workstations| + |servers| + |switches| + |hubs|$$

· $Pr(s)$**, a prior probability for each fault state** $s \in S$**.** This is the probability of a network node to fail based on prior knowledge or statistics about the domain. The distribution over nodes may be uniform or arbitrary.

· A_T**, the set of test actions.** The test actions in this research examine if there is an active communication between any pair of nodes. The test is based on the success or failure of a `ping` command between two nodes. Only workstations and servers are involved in these `ping` test actions, therefore, their total number is

$$|A_T| = \binom{|workstations| + |servers|}{2}$$

· A_R**, the set of repair actions.** Each repair action restores a node by rebooting the device at the node. Therefore, the total number of repair actions is the total number of nodes in the network:

$$|A_R| = |workstations| + |servers| + |switches| + |hubs|$$

· **a cost function** $c(s, a)$ **over actions** $a \in A_T \cup A_R$ **and states** $s \in S$**.** The cost of actions in the network domain relates to time. Each action, whether test or repair, incurs a cost equal to the number of seconds it takes to complete in the current state. More specifically, the cost of a test action (`ping`) depends on the number of hops and the integrity of the path(s) between the corresponding nodes. The cost of repair actions differs depending on the type of node and increases along the following ordering: hub reboot, switch reboot, workstation reboot, server reboot.

· **a deterministic observation model** $o(s, a)$ **over actions** $a \in A_T \cup A_R$ **and states** $s \in S$**.** For each test action a, the observation model indicates whether the `ping` command between the corresponding nodes was successful in the current state s. For each repair action a, the observation model indicates whether the `reboot` command on the corresponding node was successful in repairing the fault in the current state s.

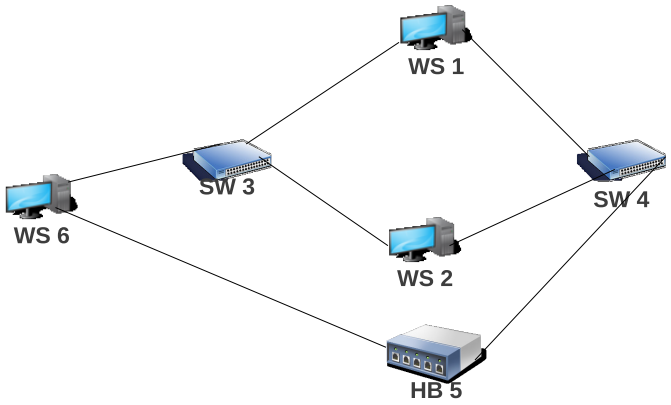


Figure 5.1: Network example for planning simulator

Fault states						
s_1	s_2	s_3	s_4	s_5	s_6	
0.167	0.167	0.167	0.167	0.167	0.167	Prior Probabilities

Table 5.1: Prior probabilities of states in example shown in Figure 5.1

5.2 CSFR Planning Solution for Network Repair

The most critical decision parameter is the total cost of detecting and repairing a faulty node. The expected value of this cost is computed dynamically by the recursive equation presented in Chapter 2. The less this value is, the better the decision policy followed. In order to give the opportunity to the reader to fully understand the generation of the optimal CSFR policy and the safe MiniMax CSFR policy, an example follows. Figure 5.1 shows a simple network with 3 workstations (WS), 2 switches (SW), and 1 hub (HB); each node also carries a unique numeric ID. The complete CSFR model for this example is the following:

· **Fault states** $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$, $|S| = 6$

· **Prior probabilities** $Pr(s)$ The prior probabilities are based on a uniform distribution, therefore, as a result, the prior probability of each node to break down is 1/6 (see Table 5.1).

· **Test actions** A_T The number of test actions is $\binom{3+0}{2} = 3$:

- `ping` from node WS1 to node WS2

- `ping` from node WS1 to node WS6
- `ping` from node WS2 to node WS6

· **Repair actions** A_R The number of repair actions is 6:

- `reboot`/repair of node WS1
- `reboot`/repair of node WS2
- `reboot`/repair of node SW3
- `reboot`/repair of node SW4
- `reboot`/repair of node HB5
- `reboot`/repair of node WS6

· **Cost function** $c(s, a)$ Table 5.2 shows the complete cost function. The columns of the table refer to fault states and the rows of the table refer to (test and repair) actions. When the communication path is not interrupted by the fault node, the cost of a test action is the number of hops between the participating nodes times a constant time cost (default=100ms) for the `ping` along a single hop. When the communication path is interrupted by the fault node, the cost of a test action is the number of hops from the originating node up to the fault (interrupting) node times the constant time cost for the `ping` along a single hop augmented by the repair cost of the faulty node. For example, the cost of the first test action (ping between nodes WS1 and WS2) is $100 \times 2 = 200$, when the fault state is s_3, whereas the cost of the same action is $100 \times 2 + 1800 = 2000$, when the fault state is s_2. Observe that the last six actions (repair actions) have the same value across columns, due to the fact that the cost of repairing a fault node is constant regardless of the current fault state.

· **Observation model** $o(s, a)$ Table 5.3 shows the complete observation model. Recall that 1 indicates success, whereas 0 indicates failure of the corresponding action in each fault state.

Table 5.4 presents the actual costs of the original CSFR algorithm and the Mini-Max CSFR algorithm with the addendum of three different heuristics. Furthermore, the average and maximum costs are presented for all four algorithms. Let us have a look

Fault states						
s_1	s_2	s_3	s_4	s_5	s_6	**Actions**
1800	2000	200	200	200	200	`ping` WS1 to WS2
1800	200	1500	200	200	2000	`ping` WS1 to WS6
200	1800	1500	200	200	2000	`ping` WS2 to WS6
1800	1800	1800	1800	1800	1800	`reboot` WS1
1800	1800	1800	1800	1800	1800	`reboot` WS2
1400	1400	1400	1400	1400	1400	`reboot` SW3
1400	1400	1400	1400	1400	1400	`reboot` SW4
1000	1000	1000	1000	1000	1000	`reboot` HB5
1800	1800	1800	1800	1800	1800	`reboot` WS6

Table 5.2: Cost function for the example shown in Figure 5.1

Fault states						
s_1	s_2	s_3	s_4	s_5	s_6	**Actions**
0	0	1	1	1	1	`ping` WS1 to WS2
0	1	0	1	1	0	`ping` WS1 to WS6
1	0	0	1	1	0	`ping` WS2 to WS6
1	0	0	0	0	0	`reboot` WS1
0	1	0	0	0	0	`reboot` WS2
0	0	1	0	0	0	`reboot` SW3
0	0	0	1	0	0	`reboot` SW4
0	0	0	0	1	0	`reboot` HB5
0	0	0	0	0	1	`reboot` WS6

Table 5.3: Observation model for the example shown in Figure 5.1

s_1	s_2	s_3	s_4	s_5	s_6	**Average**	**Max**	
3600	5600	3100	2800	1400	5400	3650	5600	**original CSFR**
5400	4000	4900	4600	3200	2000	4016	5400	**minimax CSFR Least Expected**
5400	4000	4900	4600	3200	2000	4016	5400	**minimax CSFR Cheapest**
5400	4000	4900	3600	4600	2000	4083	5400	**minimax CSFR Likeliest**

Table 5.4: Actual costs for the example shown in Figure 5.1

at states s_1 and s_6. The original CSFR algorithm gives an actual cost which is worse than the actual cost of any MiniMax CSFR algorithm at state s_6 and better at state s_1. Figures 5.2 and 5.3 present the management of beliefs by the original CSFR and the MiniMax CSFR algorithms respectively. On one hand, both models conclude successfully

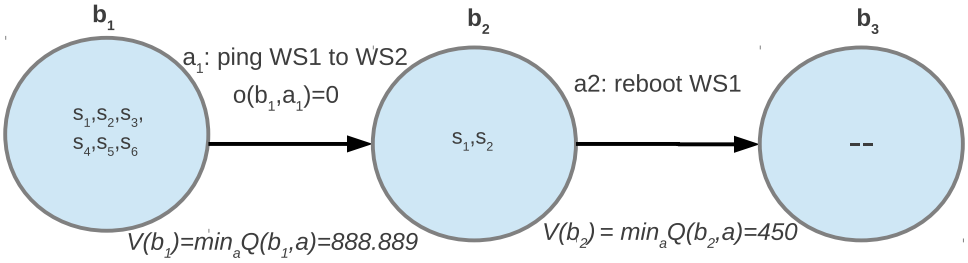


Figure 5.2: Original CSFR planning for the example in Figure 5.1 when WS1 is faulty

with the detection of the faulty state s_1, but on the other hand the final actual cost is different due to the different policies they follow.

Suppose that node WS1 breaks down. If WS1 tries to communicate with WS2, the `ping` command will be unsuccessful, since the starts from WS1: (**WS1**, SW3, WS2). On the other hand, if WS2 makes an attempt to communicate with WS6, the communication will be successful, since the path is: (WS2, SW3, WS6); in this case, the fault node is not included in the path.Figure 5.2 shows a schematic that illustrates the policy of CSFR planning in this case. The belief state is initialized to S (it contains all possible faults) and reduces gradually. The policy allows the agent to find action a_1 that minimizes the expected value in belief b_1. Action a_1 is chosen to be a `ping` from WS1 to WS2 and the resulting observation is `0`, which means that the ping was unsuccessful. As a result, states s_1 and s_2 are remained in the belief, due to the fact that only their outcomes concur with the observed one. Then, the agent decides to take a repair action . The expected cost weighted by the probabilities of the two remaining states are exactly the same, because both the prior probabilities of state s_1 and s_2 to be fault (see definition of Prior Probabilities at the begin of the section 5.2) and the repair cost of each state are the same respectively. The agent decided to repair the states in turn. At this point, the fault manager repairs $s1$ which leads to repair the actual faulty state. The remediation finished successfully and the manager clears its belief.

Figure 5.3 shows a schematic that illustrates the policy of MiniMax CSFR planning in the same case, using least expected cost heuristic. The belief state is initialized to S (it contains all possible faults) and reduces gradually. The policy allows the agent to find action a_1 that minimizes the expected value in belief b_1. Action a_1 is chosen to be a `ping` from WS1 to WS2 and the resulting observation is `0`, which means that the ping was unsuccessful. As a result, states s_1 and s_2 are remained in the belief, due to the fact

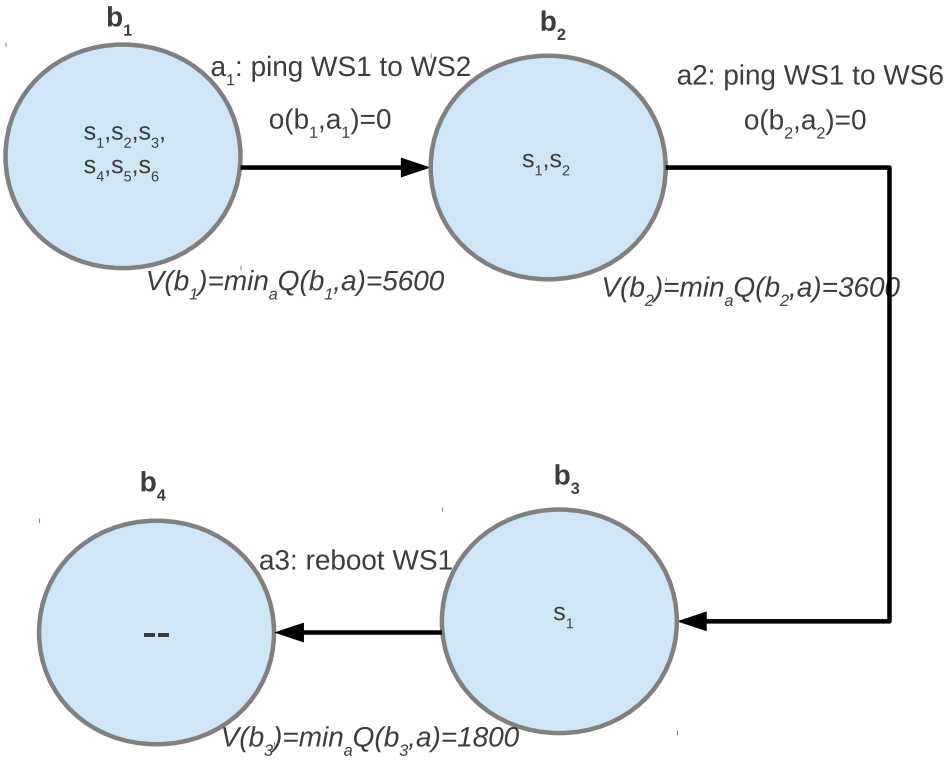


Figure 5.3: MiniMax CSFR planning for the example in Figure 5.1 when WS1 is faulty

that only their outcomes concur with the observed one. The next action is chosen to be a `ping`from WS1 to WS6, instead of a repair action that is chosen from the original CSFR model at the corresponding situation. The outcome is `0`, outcome of s_2 does not concur with the observed one, therefore, state s_2 is removed from the belief. At this point, the fault manager repairs $s1$ with the straightforward action `reboot` WS1.

Figures 5.4 and 5.5 present the different policies which the original CSFR and the MiniMax CSFR follow for state s_6. Figure 5.4 shows a schematic that illustrates the policy of CSFR planning in the case of faulty state s_6. The belief state is initialized to S (it contains all possible faults) and reduces gradually. The policy allows the agent to find action a_1 that minimizes the expected value in belief b_1. Action a_1 is chosen to be a `ping` from WS3 to WS4 and the resulting observation is `1`, which means that the ping was successful. States s_1 and s_2 are removed from the belief, due to the fact that their outcomes do not concur with the observed one. The next action a_2 is chosen to be a `ping` from WS1 to WS6 and the resulting observation is `0`, which means that the ping

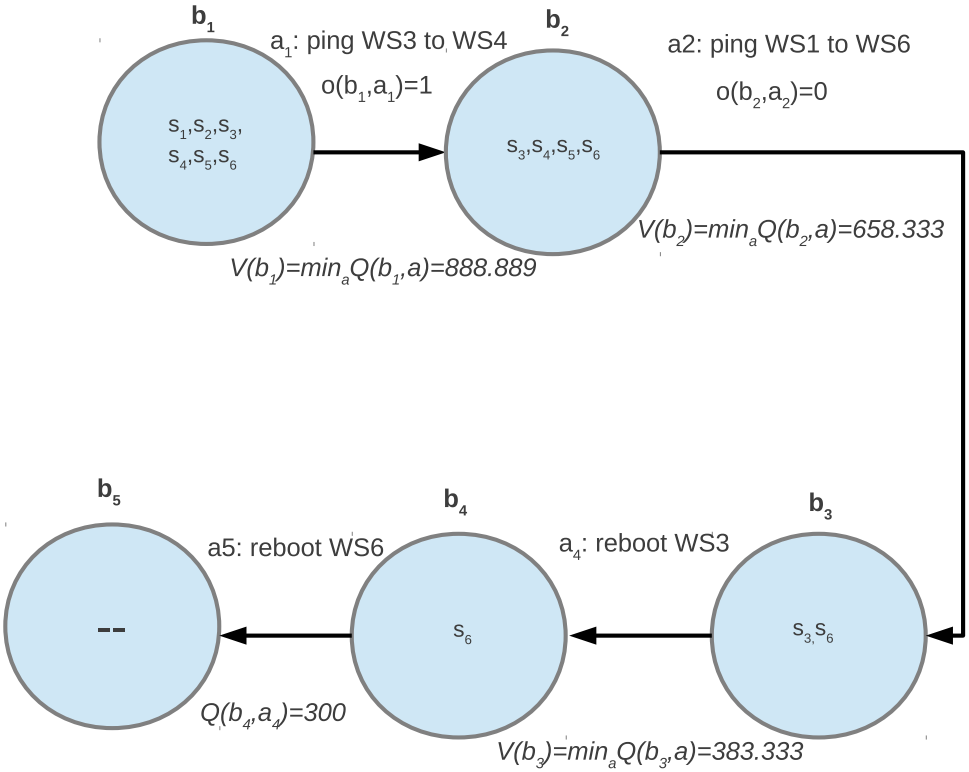


Figure 5.4: Original CSFR planning for the example in Figure 5.1 when WS6 is faulty

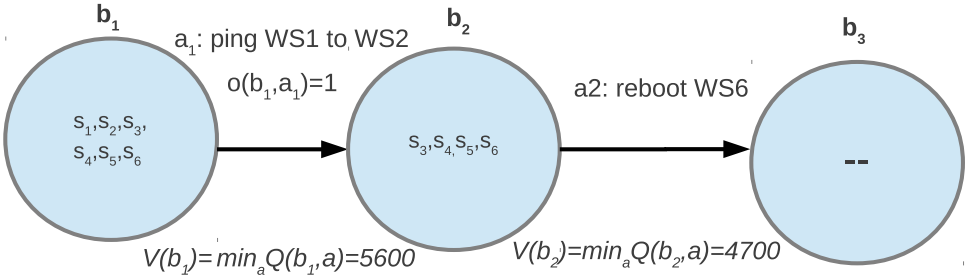


Figure 5.5: MiniMax CSFR planning for the example in Figure 5.1 when WS6 is faulty

was unsuccessful. States s_4 and s_5 are removed from the belief, due to the fact that their outcomes do not concur with the observed one. The following action a_3 is chosen to be the repair action `reboot` WS3. The fault remains but the state s_3 is removed from the belief. Eventually, it is chosen the straightforward action `reboot` WS6 for the singleton belief.

Figure 5.4 shows a schematic that illustrates the policy of CSFR planning in "faulty state s_6" case. The belief state is initialized to S (it contains all possible faults) and reduces gradually. The policy allows the agent to find action a_1 that minimizes the expected value in belief b_1. Action a_1 is chosen to be a `ping` from WS1 to WS2 and the resulting observation is `1`, which means that the ping was successful. As a result, states s_1 and s_2 are removed from the belief, due to the fact that their outcomes do not concur with the observed one. No further test actions can reduce the belief. As a result, the agent has to take a repair action. The policy which is followed by the agent force the agent to minimize the maximum (worst) total cost for the current belief b_2. The repair action is chosen to be `reboot` WS6 and the faulty state is remedied. It is worth noted that even if another heuristic function (Cheapest first / Likeliest first) is chosen, any ordering of the appropriate repair actions yields the same maximum cost (see Table 5.4).

Chapter 6

CSFR Learning for Network Repair

The previous section described how to create optimal repair policies via planning analysis. Apart from planning analysis, CSFR offers a learning methodology that can create and manipulate models based on unstructured interaction with the environment. CSFR learning applies to the planning algorithm from the previous chapter 5 by replacing set of state S, with the set of episodes E. Episodes are defined by the user, who undertakes the responsibility to choose a number of episodes that covers a range of faults. The number of possible history sequences is on the order of $(|A_T| + |A_R|)$ since an episode consists of an ordering of test actions and their outcomes, along with unsuccessful repair actions. Also, there is no need to repeat an action twice, because action outcomes are deterministic. The state space for planning is that of belief states formed by subsets of instances instead of individual instances [5]. This provides a more direct approach to the problem of taking actions to gain information.

6.1 Learned CSFR Model for Network Repair

In an attempt to adjust the solution of the problem in CSFR-learning solution, a formally definition is made by the quantities:

· **E, set of episodes** , user predefines a number of episodes to run off-line or and export results, that will be saved and used during on-line simulation. If a fault state is not covered by the predefined episodes or user predefines zero episodes to run off-line, its values are calculated on the fly and are stored.

· **Pr(e), a prior probability for episode state** , $e \in E$. It is based on the uniform distribution, as a result of that, the prior probability of each episode is: 1/(E).

· A_T**, set of test actions**

· A_R **set of repair actions**

· **c(e,a) cost function over actions** $a \in A_t \cup A_R$ **and states** $e \in E$, The cost of actions that are not applied on an episode state are defined as 0.

· **o(e,a), an outcome or observation model over actions** , $a \in A_t \cup A_R$ and episodes $e \in E$. Each test action has an outcome that is 0, 1, or ? (not yet executed), at most one repair action can be successful, and the others have outcomes of 0 or ?.

6.2 CSFR Learning Solution for Network Repair

Planning gives us satisfied results to a redundant number of fault states. However planning has its own weaknesses, the time which is consumed at a larger state space is discouraged. An alternative approach that can be used to develop an intelligent system is Learning. Planning analysis would be unable to detect and remedy the fault in a short time as learning, at most times planning consumes a forbidden time to solve the problem, depending on the network. However, cost-sensitive fault remediation model is amenable to implementation as a reinforcement-learning system. We are describing an instance-based state representation, through an example. The definition above shows the critical difference between planning analysis and learning in the framework of the CSFR model. The basic difference is that states S replaced by episodes E. Each episode emanates from the range of hypothesized fault states. In order to give the opportunity to the reader to fully understand the usage of reinforcement learning, we give an example. Assume a network 6.1 with four workstations, three switches, two hubs, and a server. Six episodes are chosen to run off-line (for the framework of the specific example, none episodes could be chosen as well). The selected fault nodes for the chosen episodes are: SW3, SW4, HB7, HB8, HB9, SRV10. Each episode gives a feedback that is stored on the tables 6.2 and 6.3. We suppose that SW4 breaks down. If WS1 tries to communicate with WS5, the procedure of ping will be unsuccessful, since the path to achieve the communication passes through SW4: WS1, SW3, HB8, **SW4**, WS5. On the other hand if WS1 makes

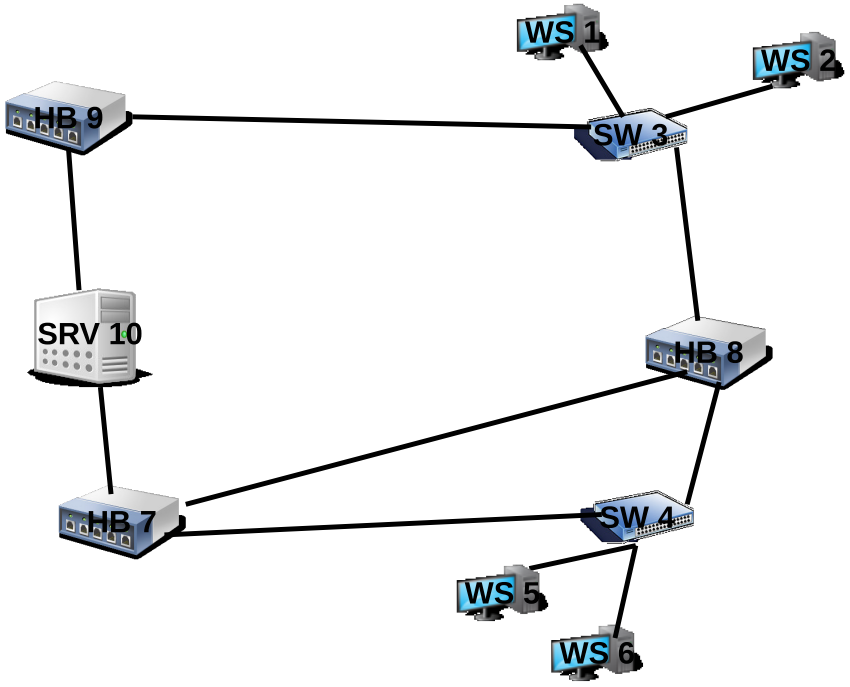


Figure 6.1: Network example for rl simulator

an attempt to communicate with WS2, the communication will be successful, since the path is: WS1, SW3, WS2. As we can see the fault node is not included. Let us match each defined term of previous section, to current example:

· **E, set of episode states** $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, $|E| = 6$

· **Pr(e), prior probability** , probability is based on the uniform distribution, as a result of that the prior probability of each node to be damaged is: 1/6 6.6

Episode - states						
e_1	e_2	e_3	e_4	e_5	e_6	
0.167	0.167	0.167	0.167	0.167	0.167	Prior Probabilities

Table 6.1: Prior Probabilities

· **Test actions** A_T The number of test actions is $\binom{4+1}{2} = 10$:

- `ping` from node WS1 to node WS2

- `ping` from node WS1 to node WS5
- ...
- `ping` from node WS5 to node WS10
- `ping` from node WS6 to node WS10

· **Repair actions** A_R The number of repair actions is 10:

- `reboot`/repair of node WS1
- `reboot`/repair of node WS2
- ...
- `reboot`/repair of node SW9
- `reboot`/repair of node SW10

· **c(e,a) cost function over actions** $a \in A_t \cup A_R$ **and states** $e \in E$, the table 6.2 represents the cost function.

Episode states						
e_1	e_2	e_3	e_4	e_5	e_6	**Actions**
1500	?	?	200	?	200	`ping` WS1 to WS2
1500	1700	?	?	?	?	`ping` WS1 to WS5
1500	1700	400	600	?	?	`ping` WS1 to WS6
?	300	?	300	400	2500	`ping` WS1 to SRV10
1500	1700	?	?	?	?	`ping` WS2 to WS5
1500	1700	400	600	?	?	`ping` WS2 to WS6
1500	300	300	300	?	?	`ping` WS2 to SRV10
200	?	200	?	?	?	`ping` WS5 to WS6
300	1500	?	?	?	?	`ping` WS5 to SRV10
300	1500	?	300	?	?	`ping` WS6 to SRV10
1800	1800	?	1800	?	?	`reboot` WS1
?	?	?	?	?	?	`reboot` WS2
1400	1400	?	1400	?	?	`reboot` SW3
1400	1400	?	?	?	1400	`reboot` SW4
?	1800	?	1800	?	?	`reboot` WS5
1800	1800	?	1800	1800	?	`reboot` WS6
1000	1000	1000	?	?	?	`reboot` HB7
?	1000	?	1000	?	?	`reboot` HB8
?	1000	1000	?	1000	?	`reboot` HB9
2200	2200	2200	2200	?	2200	`reboot` SRV10

Table 6.2: Cost Function RL

Episode states						
e_1	e_2	e_3	e_4	e_5	e_6	**Actions**
0	?	?	1	?	1	`ping` WS1 to WS2
0	0	?	?	?	?	`ping` WS1 to WS5
0	0	1	1	?	?	`ping` WS1 to WS6
?	1	?	1	1	0	`ping` WS1 to SRV10
0	0	?	?	?	?	`ping` WS2 to WS5
0	0	1	1	?	?	`ping` WS2 to WS6
0	1	1	1	?	?	`ping` WS2 to SRV10
1	?	1	?	?	?	`ping` WS5 to WS6
1	0	?	?	?	?	`ping` WS5 to SRV10
1	0	?	1	?	?	`ping` WS6 to SRV10
0	0	?	0	?	?	`reboot` WS1
?	?	?	?	?	?	`reboot` WS2
1	0	?	0	?	?	`reboot` SW3
0	1	?	?	?	0	`reboot` SW4
?	0	?	0	?	?	`reboot` WS5
0	0	?	0	0	?	`reboot` WS6
0	0	1	?	?	?	`reboot` HB7
?	0	?	1	?	?	`reboot` HB8
?	0	0	?	1	?	`reboot` HB9
0	0	0	0	?	1	`reboot` SRV10

Table 6.3: Observe Function RL

· **o(e,a) observation function over actions** $a \in A_t \cup A_R$ **and states** $e \in E$, the table 6.3 represents the observation model. Columns of table 6.2 refer to episode states, which are simulated off-line. The node which is fault can not be detected through the table 6.2 which represents cost function, it can be detected through the table that represents observation model 6.3. Take a look at the table 6.3, it is obvious that the fault node at the first episode is SW3, as the repair action `reboot` SW3 concurs with the outcome `1`. The rest fault nodes of the remaining five episodes are similarly in turn: SW4, HB7, HB8, HB9, and SRV10.

At this point the schematic 6.2 will be presented in order to explain the strategy. The first belief includes all the episodes. The system acts with a `ping` from WS5 to WS10 and the observation result is 0. Take a look at the test action `ping` WS5 to WS10 of the observation model. Candidate episodes of the new belief are: e_2 with outcome 0 and e_3, e_4, e_5, e_6 with outcome `?`. The next action is chosen to be ping from WS6 to WS10. The observation result is 0. The episodes of the current belief concurs with that outcome or outcome `?` are episodes: e_2, e_3, e_5, e_6. As a result, episode e_4 is removed from the belief. The next action is chosen to be repair action `reboot` HB7, as result, e_3 is removed as its outcome for the specific repair

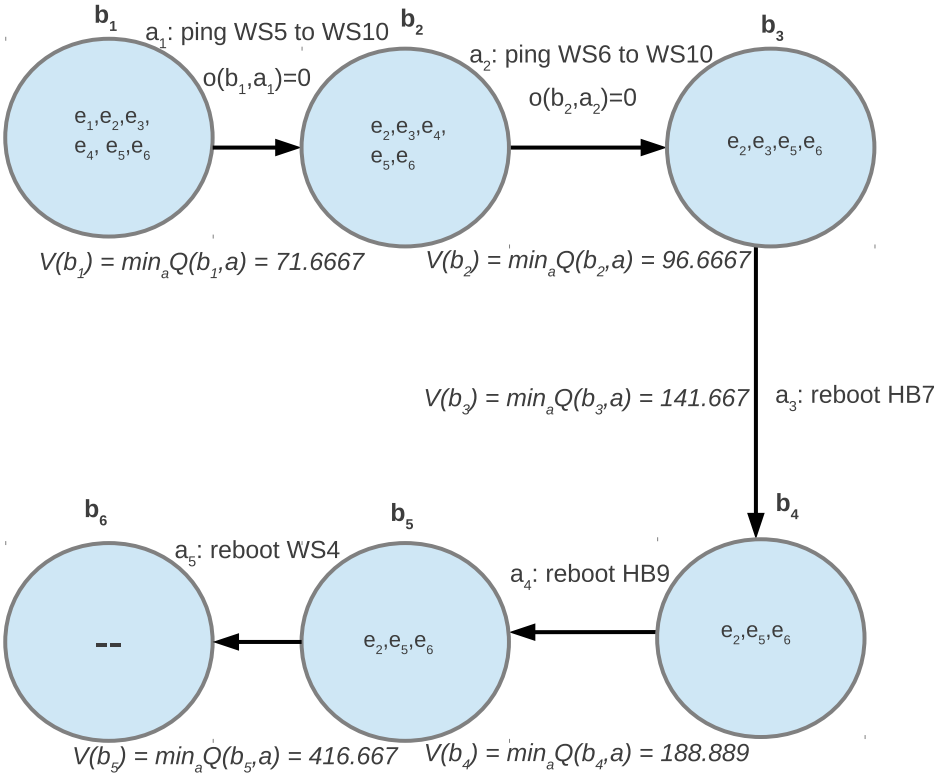


Figure 6.2: Belief Schematic of CSFR Learning for example 6.1

action is 1. Finally, the next action is `reboot` WS4, the outcome of the action is `1` for the episode e_2, and the faulty node WS4 is remedied. Assume that another fault appears on the network, the current faulty node is **WS6**. The fault manager conclude to remove all the episode states, following the previous policy. However none of them covers the current fault. Eventually, the agent adds an other episode on fly, by applying CSFR planning to the current instance of the faulty network (see Figure 6.3). The current forms of cost and observation tables include an additional column, the column with the new episode state e_7.

Let us use the same network in order to describe the policy of MiniMax CSFR Learning. Four episodes are chosen to run off-line. Episode e_1 corresponds to fault node WS1, e_2 corresponds to fault node SW4, e_3 corresponds to fault node HB8, e_4 corresponds to fault node SRV10. The explicit form of the terms of MiniMax CSFR Learning is given comprehensively.

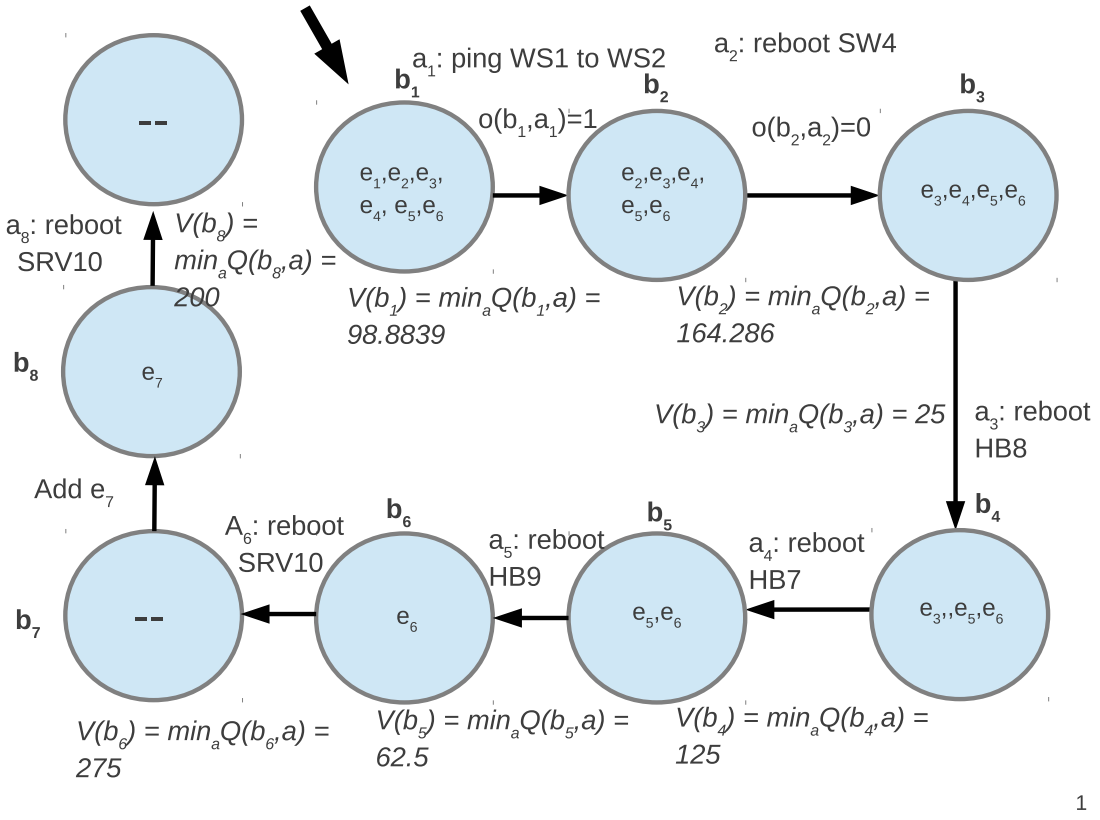


Figure 6.3: Belief Schematic of CSFR Learning – adding episode e for example 6.1

· **E, set of episode states** $E = \{e_1, e_2, e_3, e_4\}$, $|E| = 4$

· **Pr(e), prior probability** , probability is based on the uniform distribution, as a result of that the prior probability of each node to be damaged is: 1/4 6.6

· **Test actions** A_T The number of test actions is $\binom{4+1}{2} = 10$:

- `ping` from node WS1 to node WS2
- `ping` from node WS1 to node WS5
- ...
- `ping` from node WS5 to node WS10
- `ping` from node WS6 to node WS10

Episode states							
e_1	e_2	e_3	e_4	e_5	e_6	e_7	**Actions**
1500	?	?	200	?	200	?	`ping` WS1 to WS2
1500	1700	?	?	?	?	400	`ping` WS1 to WS5
1500	1700	400	600	?	?	?	`ping` WS1 to WS6
?	300	?	300	400	2500	?	`ping` WS1 to SRV10
1500	1700	?	?	?	?	400	`ping` WS2 to WS5
1500	1700	400	600	?	?	2200	`ping` WS2 to WS6
1500	300	300	300	?	?	300	`ping` WS2 to SRV10
200	?	200	?	?	?	2000	`ping` WS5 to WS6
300	1500	?	?	?	?	300	`ping` WS5 to SRV10
300	1500	?	300	?	?	1800	`ping` WS6 to SRV10
1800	1800	?	1800	?	?	1800	`reboot` WS1
?	?	?	?	?	?	1800	`reboot` WS2
1400	1400	?	1400	?	?	?	`reboot` SW3
1400	1400	?	?	?	1400	1400	`reboot` SW4
?	1800	?	1800	?	?	1800	`reboot` WS5
1800	1800	?	1800	1800	?	1800	`reboot` WS6
1000	1000	1000	?	?	?	?	`reboot` HB7
?	1000	?	1000	?	?	1000	`reboot` HB8
?	1000	1000	?	1000	?	1000	`reboot` HB9
2200	2200	2200	2200	?	2200	?	`reboot` SRV10

Table 6.4: Cost Function RL with Additional Episode

Episode states							
e_1	e_2	e_3	e_4	e_5	e_6	e_7	**Actions**
0	?	?	1	?	1	?	`ping` WS1 to WS2
0	0	?	?	?	?	1	`ping` WS1 to WS5
0	0	1	1	?	?	?	`ping` WS1 to WS6
?	1	?	1	1	0	?	`ping` WS1 to SRV10
0	0	?	?	?	?	1	`ping` WS2 to WS5
0	0	1	1	?	?	0	`ping` WS2 to WS6
0	1	1	1	?	?	1	`ping` WS2 to SRV10
1	?	1	?	?	?	0	`ping` WS5 to WS6
1	0	?	?	?	?	1	`ping` WS5 to SRV10
1	0	?	1	?	?	0	`ping` WS6 to SRV10
0	0	?	0	?	?	0	`reboot` WS1
?	?	?	?	?	?	0	`reboot` WS2
1	0	?	0	?	?	?	`reboot` SW3
0	1	?	?	?	0	0	`reboot` SW4
?	0	?	0	?	?	0	`reboot` WS5
0	0	?	0	0	?	1	`reboot` WS6
0	0	1	?	?	?	?	`reboot` HB7
?	0	?	1	?	?	0	`reboot` HB8
?	0	0	?	1	?	0	`reboot` HB9
0	0	0	0	?	1	?	`reboot` SRV10

Table 6.5: Observe Function RL with Additional Episode

· **Repair actions** A_R The number of repair actions is 10:

Episode - states				
e_1	e_2	e_3	e_4	
0.25	0.25	0.25	0.25	Prior Probabilities

Table 6.6: Prior Probabilities

- `reboot`/repair of node WS1
- `reboot`/repair of node WS2
- ...
- `reboot`/repair of node SW9
- `reboot`/repair of node SW10

· **c(e,a) cost function over actions** $a \in A_t \cup A_R$ **and states** $e \in E$, the table 6.7 represents the cost function.

Episode states				
e_1	e_2	e_3	e_4	**Actions**
1800	?	200	200	`ping` WS1 to WS2
1800	?	600	?	`ping` WS1 to WS5
1800	?	600	400	`ping` WS1 to WS6
1800	?	300	2500	`ping` WS1 to SRV10
400	?	600	400	`ping` WS2 to WS5
400	?	600	400	`ping` WS2 to WS6
300	?	?	2500	`ping` WS2 to SRV10
200	?	200	200	`ping` WS5 to WS6
?	1500	300	?	`ping` WS5 to SRV10
300	?	300	2500	`ping` WS6 to SRV10
1800	1800	1800	1800	`reboot` WS1
1800	?	1800	1800	`reboot` WS2
1400	?	1400	1400	`reboot` SW3
1400	1400	1400	1400	`reboot` SW4
1800	?	1800	1800	`reboot` WS5
1800	1800	1800	?	`reboot` WS6
1000	1000	1000	1000	`reboot` HB7
1000	?	1000	1000	`reboot` HB8
1000	1000	?	1000	`reboot` HB9
2200	?	2200	2200	`reboot` SRV10

Table 6.7: Cost Function RL

· **o(e,a) observation function over actions** $a \in A_t \cup A_R$ **and states** $e \in E$, the table 6.8 represents the observation model.

Episode states				
e_1	e_2	e_3	e_4	**Actions**
0	?	1	1	`ping` WS1 to WS2
0	?	1	?	`ping` WS1 to WS5
0	?	1	1	`ping` WS1 to WS6
0	?	1	0	`ping` WS1 to SRV10
1	?	1	1	`ping` WS2 to WS5
1	?	1	1	`ping` WS2 to WS6
1	?	?	0	`ping` WS2 to SRV10
1	?	1	1	`ping` WS5 to WS6
?	0	1	?	`ping` WS5 to SRV10
1	?	1	0	`ping` WS6 to SRV10
1	0	0	0	`reboot` WS1
0	?	0	0	`reboot` WS2
0	?	0	0	`reboot` SW3
0	1	0	0	`reboot` SW4
0	?	0	0	`reboot` WS5
0	0	0	?	`reboot` WS6
0	0	0	0	`reboot` HB7
0	?	1	0	`reboot` HB8
0	0	?	0	`reboot` HB9
0	?	0	1	`reboot` SRV10

Table 6.8: Observe Function RL

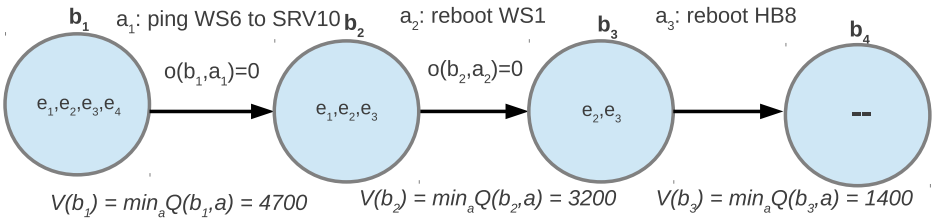


Figure 6.4: Belief Schematic of MiniMax CSFR Learning for example 6.1

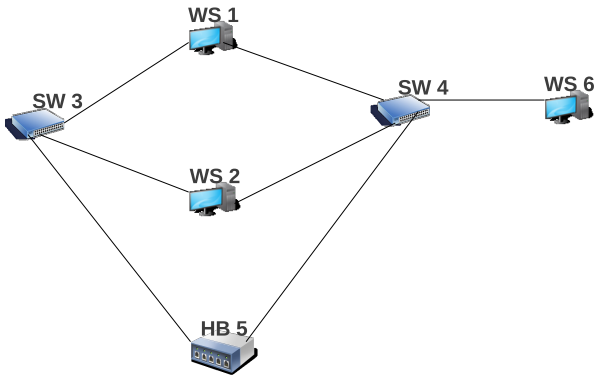
At this point, Figure 6.4 will be used in order to explain the strategy. Assume that the faulty node is HB8. The first belief includes all the episodes. The system acts with a `ping` from WS6 to SRV10 and the observation result is 1. Take a look at the test action `ping` WS6 to SRV10 of the observation model. Candidate episodes of the new belief are: e_2 with outcome `?` and e_1, e_3 with outcome `1`. No further test actions can reduce the belief. As a result, the agent has to take a repair action. The policy followed by the agent force the agent to minimize the maximum (worst) total cost for the current belief b_2. The next action is chosen to be `reboot` WS1. Episode e_1 is removed from the belief b_3, as its outcome for the previous repair action is `0`. Finally, the system is remedied by the repair action `reboot` HB8.

Chapter 7

Results

In this chapter we will demonstrate a variety of experiments in order to present original Cost Fault Sensitive Remediation Model and MiniMax Cost Fault Sensitive Remediation Model with the addendum of the three heuristics: "least expected cost", "cheapest first", and "likeliest first", applied to two different networks in order to succeed detection and remediation of faulty nodes. The experiments are built scenarios of seven sets. The simulation has run independently without any interference in all circumstances and at each conclusion we collect the data we need such as: the average actual cost, the 95% confidence interval, and max actual cost.

The first set of experiments include, original CSFR planning , and three heurestics of MiniMax CSFR planning applied to the cyclic network 7.1. The networks consists of three workstations, two switches, and one hub. The number of test actions is $|A_T| = 3$ and the number of repair actions is $|A_R| = 6$. The first set of experiments are executed twice. At first execution, faults of the network are being uniformly distributed to the nodes with probability $\frac{1}{6}$ and prior probabilities of the original CSFR model $Pr(s_i) = \frac{1}{6}$, $i \in (WS1, W2, SW3, SW4, HB5, WS6)$. At second execution, faults of the network are being nonuniformly distributed to the nodes, and the probabilities of each node to be faulty are in turn: 0.05, 0.15, 0.3, 0.3, 0.15, 0.05. The term of the prior probabilities $Pr(s)$ of the original CSFR model follows the same distribution $Pr(WS1) = 0.05$, $Pr(WS2) = 0.15$, $Pr(SW3) = 0.3$, $Pr(SW4) = 0.3$, $Pr(HB5) = 0.15$, $Pr(WS6) = 0.05$, therefore the best average of actual costs is expected to be the average of the original CSFR model. However, maximum actual cost of MiniMax CSFR is expected to be less than the corresponding value of the original CSFR model. The second set of experiments include,



Information of topology	
Element	#No.
total	6
workstations	3
switches	2
hubs	1
servers	0
workstations' link	2
switches' link	4
hubs' link	2
servers' link	0
maximum number of links	4
number of test actions	3
number of repair actions	6
number of total actions	9

Figure 7.1: Network example for Planning experiments

original CSFR learning, and three heurestics of MiniMax CSFR learning applied to the same cyclic network 7.1. We are expected to take respective results at that case too.

AVG	**95% C.I.**	**MAX**	
3581.2	90.1541	5600	**original CSFR**
4053.4	69.7095	5400	**MiniMax CSFR least expected**
4053.4	69.7095	5400	**MiniMax CSFR cheapest**
4112.2	67.0518	5400	**MiniMax CSFR likeliest**

Table 7.1: Planning with correct priors and uniform dist for network 7.1

AVG	**95% C.I.**	**MAX**	
2929.9	74.7966	7000	**original CSFR**
4189.7	46.8434	5400	**MiniMax CSFR least expected**
4281.9	49.4719	5400	**MiniMax CSFR cheapest**
4189.7	46.8434	5400	**MiniMax CSFR likeliest**

Table 7.2: Planning with correct priors and nonuniform dist for network 7.1

The third set of experiments include, original CSFR planning, and the three heurestics of MiniMax CSFR planning applied to the same cyclic network 7.1, using wrong prior probabilities for fault states/nodes as input of the original CSFR model. Third set of experiments are executed twice, for uniform and non uniform distribution of fault to be occurred. When faults of the network are being uniformly distributed to the nodes with probability $\frac{1}{6}$, the prior probabilities of the original CSFR model follows a nonuniform distribution, due to the the nature of sampling these observed statistics

AVG	95% C.I.	MAX	
4532.9	114.195	7800	**original CSFR**
5150.6	93.4443	7800	**MiniMax CSFR least expected**
5258.8	103.637	8400	**MiniMax CSFR cheapest**
5309.8	99.7211	8400	**MiniMax CSFR likeliest**

Table 7.3: Learning with uniform dist for network 7.1

AVG	95% C.I.	MAX	
4136.5	123.873	8600	**original CSFR**
5807.6	148.944	9200	**MiniMax CSFR least expected**
5657.4	140.11	9200	**MiniMax CSFR cheapest**
5716.8	143.302	9200	**MiniMax CSFR likeliest**

Table 7.4: Learning with nonuniform dist for network 7.1

are wrong, $Pr(WS1) = 0.05$, $Pr(WS2) = 0.15$, $Pr(SW3) = 0.3$, $Pr(SW4) = 0.3$, $Pr(HB5) = 0.15$, $Pr(WS6) = 0.05$. At the next execution, the two distributions are reversed. Our expectations are better or, at worst case, the same for the maximum actual cost of MiniMax CSFR model, we cannot guarantee the performance of the average. Following example presents better maximum actual cost for the MiniMax CSFR model, but a worse average.

AVG	95% C.I.	MAX	
3869.2	106.949	7000	**original CSFR**
4073.6	68.5294	5400	**MiniMax CSFR least expected**
4002	70.9756	5400	**MiniMax CSFR cheapest**
4073.6	68.5294	5400	**MiniMax CSFR likeliest**

Table 7.5: Planning with wrong priors and uniform dist for network 7.1

The fourth set of experiments include, original CSFR planning , and three heurestics of MiniMax CSFR planning applied to the cyclic network 7.2. Network 7.2 offers the opportunity to present some different cases which affects the average and maximum actual cost. The networks consists of four workstations, two switches, and two hubs. The number of test actions is $|A_T| = 6$ and the number of repair actions is $|A_R| = 8$. The first set of experiments are executed twice. At first execution, faults of the network are being uniformly distributed to the nodes with probability $\frac{1}{8}$ and prior probabilities of the

AVG	95% C.I.	MAX	
3297.2	80.7423	5600	**original CSFR**
4261.2	52.5597	5400	**MiniMax CSFR least expected**
4261.2	52.5597	5400	**MiniMax CSFR cheapest**
4180.8	50.586	5400	**MiniMax CSFR likeliest**

Table 7.6: Planning with wrong priors and nonuniform dist for network 7.1

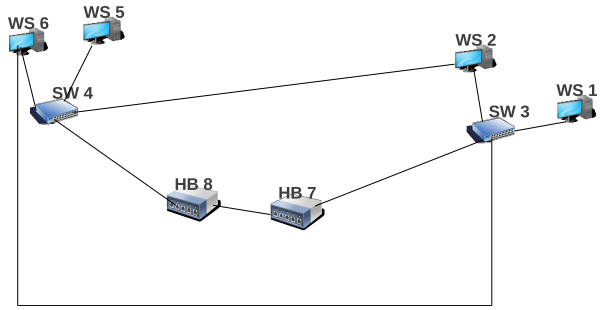


Figure 7.2: Network second example for Planning experiments

Information of topology	
Element	#No.
total	8
workstations	4
switches	2
hubs	2
servers	0
workstations' link	2
switches' link	4
hubs' link	2
servers' link	0
maximum number of links	4
number of test actions	6
number of repair actions	8
number of total actions	14

original CSFR model $Pr(s_i) = \frac{1}{8}$, $i \in (WS1, W2, SW3, SW4, WS5, WS6, HB7, HB8)$. At second execution, faults of the network are being nonuniformly distributed to the nodes, and the probabilities of each node to be faulty are in turn: 0.05, 0.1, 0.15, 0.2, 0.2, 0.15, 0.1, 0.05. The term of the prior probabilities $Pr(s)$ of the original CSFR model follows the same distribution $Pr(WS1) = 0.05$, $Pr(WS2) = 0.1$, $Pr(SW3) = 0.15$, $Pr(SW4) = 0.2$, $Pr(WS5) = 0.2$, $Pr(WS6) = 0.15$, $Pr(HB7) = 0.1$, $Pr(HB8) = 0.05$, therefore the best average of actual costs is expected to be the average of the original CSFR model. However, maximum actual cost of MiniMax CSFR is expected to be less than the corresponding value of the original CSFR model. The second set of experiments include, original CSFR learning, and three heurestics of MiniMax CSFR learning applied to the same cyclic network 7.2. We are expected to take respective results at that case too.

AVG	95% C.I.	MAX	
3887.5	84.6744	5600	**original CSFR**
4090.9	69.1644	5600	**MiniMax CSFR least expected**
4083.4	68.5474	5600	**MiniMax CSFR cheapest**
4141.3	64.0875	5600	**MiniMax CSFR likeliest**

Table 7.7: Planning with correct priors and uniform dist for network 7.2 for 1000 faults

AVG	95% C.I.	MAX	
3577.74	76.8742	5600	**original CSFR**
3871.12	74.7681	5600	**MiniMax CSFR least expected**
3912.94	74.1027	5600	**MiniMax CSFR cheapest**
3871.12	74.7681	5600	**MiniMax CSFR likeliest**

Table 7.8: Planning with correct priors and nonuniform dist for network 7.2 for 1000 faults

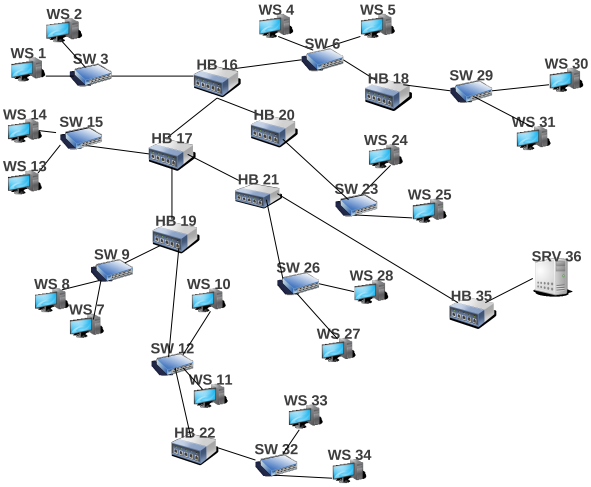
AVG	95% C.I.	MAX	
6300	199.066	12400	**original CSFR**
6620.4	192.314	11100	**MiniMax CSFR least expected**
5293.6	142.615	12000	**MiniMax CSFR cheapest**
5821.4	171.364	9800	**MiniMax CSFR likeliest**

Table 7.9: Learning with uniform dist for network 7.2 for 1000 faults

AVG	95% C.I.	MAX	
6589.79	181.564	10400	**original CSFR**
6950.93	229.335	12000	**MiniMax CSFR least expected**
4547.2	87.3459	8800	**MiniMax CSFR cheapest**
7117.18	241.885	12000	**MiniMax CSFR likeliest**

Table 7.10: Learning with nonuniform dist for network 7.2 for 1000 faults

The sixth set of experiments include, original CSFR planning, and the three heurestics of MiniMax CSFR planning applied to the same cyclic network 7.2, using wrong prior probabilities for fault states/nodes as input of the original CSFR model. Third set of experiments are executed twice, for uniform and non uniform distribution of fault to be occurred. When faults of the network are being uniformly distributed to the nodes with probability $\frac{1}{8}$, the prior probabilities of the original CSFR model follows a nonuniform distribution, due to the the nature of sampling these observed statistics are wrong, $Pr(WS1) = 0.05$, $Pr(WS2) = 0.1$, $Pr(SW3) = 0.15$, $Pr(SW4) = 0.2$, $Pr(WS5) = 0.2$, $Pr(WS6) = 0.15$, $Pr(HB7) = 0.1$, $Pr(HB8) = 0.05$. At the next execution, the two distributions are reversed. Our expectations are better or, at worst case, the same for the maximum actual cost of MiniMax CSFR model, the MiniMax CSFR model cannot guarantee anything about the average, however following examples



Information of topology	
Element	#No.
total	36
workstations	18
switches	9
hubs	8
servers	1
workstations' link	1
switches' link	4
hubs' link	4
servers' link	1
maximum number of links	4
number of test actions	171
number of repair actions	36
number of total actions	207

Figure 7.3: Network example for Learning experiments

presents some cases in which the average of the MiniMax CSFR model is better than the average of the original CSFR model, but the maximum actual cost of the two models is equal.

AVG	95% C.I.	MAX	
4088.4	75.0541	5600	**original CSFR**
4021.7	70.3242	5600	**MiniMax CSFR least expected**
4141.3	67.8437	5600	**MiniMax CSFR cheapest**
4021.7	70.3242	5600	**MiniMax CSFR likeliest**

Table 7.11: Planning with wrong priors and uniform dist for network 7.2 for 1000 faults

AVG	95% C.I.	MAX	
3957.27	82.3303	5600	**original CSFR**
3962.32	73.8436	5600	**MiniMax CSFR least expected**
3905.69	74.3051	5600	**MiniMax CSFR cheapest**
4152.59	71.6135	5600	**MiniMax CSFR likeliest**

Table 7.12: Planning with wrong priors and nonuniform dist for network 7.2 for 1000 faults

The seventh set of experiments include, original CSFR learning, and three heurestics of MiniMax CSFR learning applied to the larger non cyclic network 7.3. The networks consists of eighteen workstations, nine switches, eight hubs, and 1 server. The test actions

are $|A_T| = 171$ in total and repair actions are $|A_R| = 36$. Fourth set is executed twice, for uniform and non uniform distribution of fault to be occurred. Given a sampling of the observed faulty nodes, some nodes break down with zero probability and $SW15$, $HB20$, $WS25$, $WS30$, $HB35$ with probabilities $Pr(SW15) = 0.1$, $Pr(HB20) = 0.2$, $Pr(WS25) = 0.4$, $Pr(WS30) = 0.2$, $Pr(HB35) = 0.1$. The next execution is occurred with uniformly distributed probabilities.

AVG	**95% C.I.**	**MAX**	
4639.19	80.1208	6100	**original CSFR**
4222.02	52.3757	5700	**MiniMax CSFR least expected**
4222.02	52.3757	5700	**MiniMax CSFR cheapest**
4222.02	52.3757	5700	**MiniMax CSFR likeliest**

Table 7.13: Learning with uniform dist for network 7.3

AVG	**95% C.I.**	**MAX**	
4135.94	197.385	7500	**original CSFR**
3723.35	155.68	5800	**MiniMax CSFR least expected**
3723.35	155.68	5800	**MiniMax CSFR cheapest**
3723.35	155.68	5800	**MiniMax CSFR likeliest**

Table 7.14: Learning with nonuniform dist for network 7.3

Chapter 8

Conclusion

This chapter demonstrates our basic findings of this research and its contribution. Furthermore, limitations of the system are highlighted and a proposal for future work is provided. The limitations of the model as well as its assumptions could be revisited and eliminated, so a safe as well as an optimal decision making system could be presented in the near future. Using planning with the MiniMax criterion complemented by the three heuristics, presented in Chapter 3, better decisions could be taken in the field of autonomic repair systems. The intelligent system created by this research puts the sequential decisions for diagnosis and repairing on a firmer and safer footing. The basic conclusions of this research summarize as follows:

(a) The Cost-sensitive Fault Remediation (CSFR) technique can make sequential decisions suitable for diagnosis and repair systems. The CSFR model monitors an environment and takes actions based on the current state. These actions are both test and repair. Test actions aim to detect the fault state of the environment and repair actions aim to restore the environment to its proper functionality. The policy that CSFR follows is based on the cost of each action, either test or repair. The CSFR algorithm accomplishes its goal by discovering a repair policy that achieves an optimal total cost.

(b) The planning algorithm, proposed in this research, optimizes a minimax criterion, which is in most cases safer than the original CSFR criterion. The original CSFR criterion yields an upper bound value for the actual costs which in several cases is

higher than the corresponding bound of the MiniMax CSFR criterion. The worst case for the MiniMax CSFR criterion is the two values of the bounds to be equal.

(c) The experimental results indicate that the system acts in safe on behalf of the user by following the minimax criterion. Moreover, the fault manager handles weaknesses of the MiniMax CSFR model successfully, while it uses the heuristics ("least expected cost", "cheapest first", "likeliest first") described in Chapter 3.

8.1 Future Work

A possible future research direction is to apply the CSFR model with the minimax criterion in another environment apart from network repair and evaluate the efficiency of the extended model. Furthermore, the application of the model on a real network would reveal advantages and disadvantages in practice. An adjustment to a real network would set the stages for a real contribution of learning domain to the field of network repair. Last, but not least, the investigation of other criteria, would be an important contribution.

References

[1] Franklin, S., Graesser, A.: Is it an agent, or just a program?: A taxonomy for autonomous agents. In: Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages. ECAI '96, London, UK, UK, Springer-Verlag (1997) 21–35 2

[2] Oates, T.: Fault identification in computer network: A review and a new approach. Technical report, University of Massachusetts, Amherst, MA, USA (1995) 3

[3] Littman, M.L., Nguyen, T., Hirsh, H.: Cost-sensitive fault remediation for autonomic computing. In: In Proc. of IJCAI Workshop on AI and Autonomic Computing. (2003) 3, 7

[4] Sutton, R.S., Barto, A.G.: Introduction to reinforcement learning (1998) 3

[5] Littman, M.L., Ravi, N.: An instance-based state representation for network repair. In: in Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI). (2004) 287–292 7, 10, 41

[6] Littman, M.L., Ravi, N., Fenson, E., Howard, R.: Reinforcement learning for autonomic network repair. In: ICAC, IEEE Computer Society (2004) 284–285 7