Data-Driven Policies for Manufacturing Systems and Cyber Vulnerability Maintenance

DISSERTATION

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Sayak Roychowdhury

Graduate Program in Industrial and Systems Engineering

The Ohio State University

2017

Dissertation Committee:

Prof. Theodore T. Allen (Advisor)

Prof. Cathy H. Xia

Prof. Gagan Agrawal

Abstract

This research explores deterministic and stochastic policies to help organizations make data-driven optimal decisions. The two major application areas identified in this research are manufacturing and cyber security. In a recent report published by McKinsey Analytics, the manufacturing industry uses only 20%-30% of the potential of data analytics. This suggests that there are still plenty of opportunities to use analytics in manufacturing processes. In the first part of my research, I formulate an Integer Programming model for the "stamping" process in automotive manufacturing. I develop a production scheduling method for automotive stamping to maintain optimal inventory positions.

In stamping, different types of parts are scheduled for processing in the press, which requires different die-sets to be mounted on the press. This has all the elements of conventional scheduling problems with tardiness objectives and setup costs. Yet, it also has capacity constraints and part production constraints. We show that these constraints make solution with branch and bound difficult for problem sizes of interest. In this research, I use the structure of the scheduling problem and implemented heuristic methods like Genetic Algorithm alongside Earliest Due-date (EDD) rules to prioritize

production of parts with low inventory as well as minimize the number of die-set changeovers. I call this new method Genetic Algorithm with Generalized Earliest Due-date (GAGEDD). I illustrate the computational advantages compared with alternatives and show its benefits using data from a real life automotive stamping press scheduling problem to build a decision support tool for the schedulers.

The second part of this research is motivated towards improving cyber vulnerability maintenance policies under uncertainty. A conservative estimate by McAfee in 2014 puts annual cost of cybercrime at US$375B. This is an important contemporary issue where role of data analytics and optimization have a lot to offer. Here I implement stochastic optimization procedures for cybersecurity applications, where learning is incorporated to account for future rewards. First, I formulate a Partially Observable Markov Decision Process (POMDP) model to derive policies for cases when the state of compromise of a host is uncertain. This method assumes there is no parametric uncertainty. Next, I implement Bayes Adaptive Markov Decision Process model (BAMDP) on a dataset obtained from the cyber logs of an organization using finite numbers of model scenarios. Earlier BAMDP formulations use infinite model scenarios. I also describe the benefits of using finite scenarios including the ability to solve the problem optimally as a POMDP. The resulting BAMDP formulation accounts for the parametric uncertainty caused by the lack of data for certain events. I use a point based value iteration method known as PERSEUS to solve both of these problems to generate α-vectors, that can be used to design optimal policies based on the belief-state of the system.

Another benefit of using finite numbers of model scenarios relates to decision making for multiple identical systems, e.g., a "fleet" of identical Linux computer hosts. The issue of identical systems in machine learning has apparently received little attention despite the widespread relevance in data analytics. I propose a method for solving multiple identical system policy problems. The proposed method is based on a relatively large POMDP formulation with methods to compute the relevant transition, expected reward, and observation methods being provided.

Then, I explore additional advantages of finite model scenario BAMDPs relating to the ability to incorporate reward-based or other learning in intuitive ways. Also, the speed of learning and the concept of "fast learning" and average learning time are proposed and explored computationally. In concluding, I offer suggestions about how this research can be extended to build more powerful models with faster learning capabilities to help decision makers.

To Baba, Ma and Dada

Acknowledgments

I would like to thank my parents Mr. Samar Kumar Roychowdhury and Mrs. Namita Roychowdhury and my brother Mr. Somak Roychowdhury for their constant love and support. They are the ones who made me the person that I am today. My brother Somak has always inspired me to pursue Mathematics, which has played a key role in my decision to get involved in the field of Operations Research. I would also like to express my sincere gratitude to my academic advisor Dr. Ted Allen for his guidance, support and patience. Dr. Allen is a coauthor for all of this research. I have learned so many things from him, about scholarship, and life in general. I want to thank my dissertation committee members Dr. Cathy Xia and Dr. Gagan Agrawal for their support. I want to thank Prof. Natasha Bowen for agreeing to attend my doctoral defense as a graduate school representative. I want to thank my MS research project supervisor Dr. Jerald Brevick for his guidance and support. He has helped me to become a better engineer and has shown me new ways of problem solving, that would otherwise remain unknown to me. I would like to thank my friend Dr. Subhrakanti Chakraborty for being a constant source of inspiration and encouragement in my years in graduate school. Finally, I would like to thank all the staff members of the ISE family including Josh Hassenzahl and Bill Toulos, Tammy Freitas, Jen Morris for their readiness to help whenever I was in need.

Vita

2008	B.E. (Electrical Engineering) Jadavpur University, Kolkata, India
2008-2011	Executive Engineer, Siemens Limited, Kolkata India
2012 to present	Graduate Teaching Associate, Department of Integrated System Engineering, The Ohio State University, Columbus, Ohio USA
2014	M.S. (ISE), The Ohio State University, Columbus, Ohio USA
2016	Intern, Global Data Insight & Analytics, Ford Motor Company, Dearborn, MI, USA

FIELDS OF STUDY

Major Field: Industrial and Systems Engineering

Specialization: Operations Research

Publications

Roychowdhury, S., Allen, T. T., & Allen, N. B. (2017). A Genetic Algorithm with an Earliest Due Date Encoding for Scheduling Automotive Stamping Operations. *Computers & Industrial Engineering*.

Roychowdhury, S. (2014). Investigation of Flash-Free Die Casting By Overflow Design. (Masters thesis, The Ohio State University).

Roychowdhury, S., Brevick, J. R., and Reid, P. Jr. (2014). Investigation of Flash-free Die Casting. *NADCA Transactions*.

Conference Presentations

Roychowdhury, S. and Allen, T. T. (2016), Optimal Learning with Bayesian Adaptive Markov Decision Process for Cyber Vulnerability Analysis. Contributed INFORMS Abstract.

Roychowdhury, S. and Allen, T. T. (2015), Heuristic Methods For Automotive Stamping Scheduling. Contributed INFORMS Abstract.

Roychowdhury, S. and Allen, T. T. (2014), Sequential Kriging Optimization to Determine the Cost Effective Number of Early Voting Days. Invited INFORMS Abstract.

Roychowdhury, S. and Allen, T. T. (2014), Hybrid Sequential Kriging Optimization using Gradient Descent. Invited INFORMS Abstract.

Roychowdhury, S. and Allen, T. T. (2013), Addressing Multiple Responses Using Sequential Kriging Optimization. Contributed. JSM Abstract.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Data and Making Decisions

The volume of data doubles every 3 years with the rapid growth of computing power and advancement in sensor technology (Henke et al 2016). The use of big data and analytics can result in competitive advantage through increase of productivity by 5-6% over competitors (McAfee, Brynjolfsson, Davenport 2012). To remain competitive, organizations feel the necessity to incorporate data and analytics as a *core strategic vision* (Mayhew, Saleh, Williams 2016). To achieve that, machine learning and optimization tools need to be made available to decision makers at different levels of the organization. Our research is aimed at building such decision support systems for the key decision makers in organizations. To understand our motivation, let us briefly review the history and the basics of the field of Operations Research.

The use of mathematical tools to make strategic decisions started during World War II (Hillier, Lieberman 2010) and the field of study became known as "Operations Research". During the war, efficient distribution of scarce resources for military operations was of utmost importance and scientists were brought in to help with the

problems. Since the techniques proved to be effective in many cases, they found applications in different organizations and industries after the war. With time and research, the methods evolved and got enriched, and the application areas expanded through production planning, inventory management, supply chain, manufacturing, service sector, transportation and so on.

1.2 Deterministic and Stochastic Problems

The decision problems solved in Operations Research (OR) can be broadly put into two categories, i) deterministic and ii) stochastic. In deterministic problems, it is assumed that all the parameters are known, and there is no randomness involved. As George Box famously quoted that "All models are wrong, but some are useful", deterministic models are much simplified versions of the actual problems that they are addressing. For example, in Linear Programming (LP), the most basic model used in Operations Research, the objective function and the constraints are assumed to be linear with known parameters. That may not be the case however, LP has been used extensively to solve large decision problems, and the solutions have helped decision makers to use resources more efficiently than before. Even if a solution cannot be implemented exactly the way the algorithm suggests, it can still be used as a guide to make better decisions. There are many examples of how organizations have saved millions of dollars by implementing deterministic OR methods (Hillier, Lieberman 2010), and it is generally smarter to leverage these methods than to just use intuitive guesswork. Linear Programming, Integer Programming (IP), Dynamic Programming (DP) are some of the

key modeling techniques that are extensively used in OR. Deterministic methods are fast and can handle large number of variables and constraints. But on the other hand, the simplification of the problems makes it difficult to directly implement the solutions to real life situations. The trade-off between the complexity and the accuracy of the mathematical models is something an operations researcher often needs to consider while building a model for a specific problem.

In the "stochastic" problem formulations, the decision maker incorporates uncertainty involved in certain quantities. For instance, future demand of a commodity is not known beforehand at the time of deciding on production, however, a prediction on demand can be made using the past data. Certain parameters have inherent randomness in them, and taking that into account helps to make more realistic mathematical models of the problem. For example, availability of an important equipment on a production line can be regarded as a random variable, and taking into consideration a distribution over it's time to failure may help the management to prepare a maintenance schedule. Sometimes decision making can be periodic, where a decision maker must choose among the available alternatives and then has to wait for the outcome before making the next decision on the same process. Stochastic programming (SP) models with recourse are developed specifically to solve these problems. Solution procedures for SP are often derived using the techniques of LP, especially leveraging the property of duality of LP. The other methods of modeling an SP are Approximate Dynamic Programming (Powell, 2007) and Markov Decision Process (MDP) models (Puterman, 2014) which can be solved using Dynamic Programming techniques. Other methods like Monte Carlo

simulation are used to understand the dynamics of a stochastic process. More often, "approximate methods" and sampling techniques are used for stochastic problems, to keep the solution procedure computationally tractable. Once again, there is a tradeoff between how much stochasticity should be accounted for so that a reasonable solution can be reached in a decent amount of time.

1.3 Relation to This Research

In this research, we explore both the deterministic and stochastic domains of OR. We have implemented a deterministic model to solve a problem related to automotive manufacturing. For the stochastic problem, we have modeled a cyber vulnerability maintenance problem using Partially Observable Markov Decision Process (POMDP). Moving a step forward, we accounted for the parametric uncertainty in the model and implemented Bayes Adaptive MDP for a cyber security problem. It is important to understand why we call our methods "Data-driven Policies". Our goal is to build decision support tools, using which the decision maker can take actions based on the available data. For the production scheduling and inventory control problem, the decision maker is a scheduler, who must schedule the number of different parts to be processed by the stamping press looking into the available inventory, demand and storage capacity for each part. For the cyber maintenance problem, the decision maker is a network security administrator, who has to decide how many work hours or equipment cost to invest at a given vulnerability situation. In both of these problems, the tools that we have implemented are going to make use of the available data and then suggest an optimal

action to the decision maker. Note that there is no "one" solution for the problems, rather there could be many solutions based on the present conditions of the systems and the decision makers' knowledge about the systems. That is why we say that our methods generate "policies" which the decision makers can use over time. In addition to that, the BAMDP model that we have implemented has the capability to learn from the interactions with the system. We will discuss more elaborately in chapters 4, 5 and 6.

Based on our work, we have come up with 4 research questions that we are trying to address in this dissertation. We start with a deterministic production scheduling problem, connected to automotive manufacturing. This problem is more difficult than many classical scheduling problems because of inventory storage constraints and needed combinations of parts being made together. The question we are asking is:

1. "How to generate schedules, assuming known demand and production times, within reasonable times which will minimize inventory starvation and changeovers with storage and part constraints?"

The actual automotive problem has some degree of stochasticity in that the amounts of production are random because of quality problems and downtimes. Yet, because of the number of items being large, stochasticity is assumed to be not addressable. Our research explores the viability of standard solvers and branch and bound-based solvers used for relatively simple scheduling problems (Smith, 1956, Posner, 1985, Monma and Potts, 1989, Wu, Yin, and Cheng, 2011). We also seek tocombine meta-heuristic methods like Genetic Algorithm and Earliest Due Date heuristic to build decision support tools. Genetic Algorithms (GA) have found

applications as optimization tools in numerous fields. Especially, in problems which are non-convex and cannot be easily approximated by convex functions, conventional gradient based methods are not applicable. GAs randomly search through the domain of decision variables to come up with better populations of solution at every subsequent generation. The randomized search makes the process less likely to get stuck in local optimal solutions. However, for problems with many dimensions, GAs can be computationally expensive and may not converge to a good solution in reasonable amount of time. In such cases, it is best to leverage the structure of the problem first before applying the GA. The research presented in Chapter 2 is an example of how we have used the structure of a single machine scheduling problem to design Genetic Algorithm Generalized Earliest Due Date (GAGEDD) method which is effective for both production and inventory control for an automotive stamping scheduling process. The problem is deterministic in the sense that the demand is known for the period of scheduling. The method is used to build a decision support tool for the scheduler in charge of stamping operations for an automotive manufacturer. Chapter 3 compares the solutions of a typical automotive stamping problem using GAGEDD and other methods for different inventory scenarios.

Our second area of research addresses stochasticity, also large numbers of decision-periods, and the possibly that decision makers will learn during the planning horizon. Chapter 4 discusses the MDP and its stochastic variants like POMDP and BAMDP, which we have used as our solution methods. The area of application we have chosen is cyber security which includes many important sources of uncertainty or

stochasticity. With growing number of incidents in the cyber world that make headlines, cyber security is considered a domestic as well as international security threat with actors all around the world potentially launching attacks. Be it the works of rival nations or organizations, extortionists or hacktivists, victims of cybercrimes bear huge financial losses and loss of information security. McAfee estimates that cybercrime cost more than US$ 400Billion to the global economy every year (McAfee 2014). The same report explains that cyber-attacks are cheap with high incentives for the attackers, whereas the cost incurred by the defenders is huge. It is a critical business decision on how much an organization is willing to spend on cyber security. Even though the techniques used by cyber attackers are becoming more sophisticated and varied with time, yet, a lot of these attacks exploit known vulnerabilities.

To build effective strategies for network administrators to minimize the impact of cyber-attacks, we explore various methods derived from MDP. This brings to our next research question:

2. "How can we develop cyber maintenance policies when there is uncertainty in the state of compromise of a host?"

By the phrase "state of compromise", we mean whether a host is compromised, or breached by an attacker or not. A breached host can be exploited and used in a number of malicious ways, e.g. stealing information, disrupting control or being used to compromise other hosts. In many cases, it is not easy to detect if a host is breached and if left undetected, the compromised host can lead to greater financial and/or information losses. We build a partially observable Markov Decision Process model and observe how the

expected cost and optimal actions vary according to our belief about the state of compromise of the system. We make assumptions on the incumbent factors that build the transition probabilities and run our analysis for a typical scenario. Our main contribution is to identify a number of transition parameters and use them to build a POMDP model to derive cyber maintenance policies. These transition parameters can also provide valuable insights on how to improve incident logging. With comprehensive incident logging integrated with our POMDP decision support tool, more effective cyber maintenance systems can be developed. Chapter 5 explains the procedure elaborately and the optimal policies are shown for typical belief states.

In the POMDP formulation, we assume that the transition probabilities are known. In Chapter 6, we explore methods in which we assume that the transition probabilities are not known, and account for parametric uncertainty. Here we have used real life data obtained from the network security log of an organization. The transition counts from low vulnerability to high vulnerability states are derived using the Common Vulnerability Scoring System (CVSS). The available data shows various limitations, such as rare occurrence of certain state transitions under certain actions. To address the uncertainty due to unavailability of data, and install learning capabilities in the decision support system, our third research question is coined as below:

3. “With the available data on vulnerability state transitions along with its limitations, how can we account for the parametric uncertainty with reasonable confidence and make policies?”

We apply Bayes Adaptive Markov Decision Process Models (BAMDP, Duff (2002)) to derive cyber vulnerability maintenance policies under parametric uncertainty. We generate multiple scenarios from the available data and then create equivalent POMDPs to design maintenance policies. Our BAMDP approach is model based, since we start with transition probabilities derived from the collected data. To account for the uncertainty among the models, we generate a finite number of scenarios first, before solving the problem using POMDP. Although finite scenario models are faster than infinite scenario models, the computational burden increases with the number of scenarios. On the other hand, too few scenarios may lead to *spurious learning* by not taking into account a vast array of possibilities. The model effectively thinks that it is learning but there is no logical way that learning could happen on transitions other than what occurred. Our contribution in this area of research is to address this issue with the application of Orthogonal Array based Latin Hypercube (OALH) designs (Tang 1993) for sampling of the uncertain parameters to generate the scenarios. We use OALH of strength r, for cases with r uncertain parameters. This ensures that the design exploits the r-variate uniformity property of a strength r Orthogonal Array, and the scenarios sampled are free from bias. We seek to derive policies generated by models with different number of scenarios and discuss the viability of using finite scenario BAMDP methods for cyber maintenance problems.

Often decision makers need to take actions to manage not one system, but a fleet of systems identical in characteristics, e.g. a number of computers in a lab, a fleet of cars,

population of a region and so on. The question that can be posed in relation to this decision problem with parametric uncertainty is:

4. “How to design efficient models for multiple systems with identical characteristics?”

To answer this question, we extend our BAMDP based method for multiple systems and call it Multiple Identical Systems (MIdS). Our contribution is to motivate the idea of building models for multiple systems with identical characteristics. Decision makers can use this method to make policies to manage a whole group of identical systems. In section 6.5, we derive formulation for 2-state MIdS, and we demonstrate the method using a 2-state 2 systems model and derive policies by applying BAMDP. We believe MIdS can find extensive application where a large number of systems are involved and foster greater learning capability than single system models.

1.4 Summary

In this dissertation, we are discussing deterministic and stochastic methods to develop policies with applications in automotive manufacturing and cyber vulnerability maintenance. This introductory Chapter aims at providing a context on how we make decisions in our everyday life and provides a brief introduction on practice of using numerical methods to make big decisions. We have also identified four fundamental research questions that we are going to address in this dissertation.

Chapter 2: A Genetic Algorithm with an Earliest Due Date Encoding for Scheduling Automotive Stamping Operations

2.1 Introduction

Sheet metal stamping is an essential element of modern day manufacturing processes and is known for its high production rates and low labor costs (Lim, Venugopal and Ulsoy, 2014). The market for metal stamping is expected to exceed US$180B by 2022 (Grandview Research, 2015). This metal forming process finds extensive use in the automotive industry as well as in the production of commodities ranging from furniture to aircraft fuselage. In automotive manufacturing, stamping operations are often responsible for supplying many different part types to different downstream facilities (Barlatt, Cohn, Gusikhin, Fradkin, Davidson and Batey 2012). Rigorous production scheduling and workforce planning are needed to run the operations in the most efficient manner. The stamping scheduling problem considered here contains multiple elements related to part demand, stamping dies, and part storage. The elements include:

1. *Part inventory/tardiness*: To ensure that the line downstream does not starve as the result of insufficient inventory, known demand for every part type should be considered.
2. *Setups*: Die-set changeovers should be limited to reduce personnel costs and injury incidents.
3. *Die-set constraints*: Stamping dies must be mounted on the stamping press so that one or more part types can be produced simultaneously.
4. *Inventory constraints*: Limited storage space for the parts produced must be taken into account.

For cases with due date or "tardiness" objectives and conventional setup costs (setups that delay production completions), it has long been known that earliest due date (EDD) scheduling is optimal for parts that are not tardy in the context of many problem formulations (Smith, 1956, Posner, 1985, Monma and Potts, 1989, Wu, Yin, and Cheng, 2011). Therefore, it is reasonable to expect that EDD ordering of jobs may be desirable for stamping scheduling as long as the inventory level is high enough so that lateness is not an issue. However, if the inventory level is relatively low, the EDD schedules do not always provide good solutions. The focus here is to generate methods applicable to a variety of inventory conditions.

Many scheduling problems with setups and the general weighted tardiness scheduling problem are known to be NP-hard (Graham, Lawler, Lenstra and Kan 1979, Morton, 1993 and Posner, 2011; Ben Hadj-Alouane and Bean, 1997; Norman and Bean, 1997). NP-hard problems relate to formulations in which it may not be possible to find a

verifiably optimal solution in a reasonable amount of time depending on the numbers of variables. The challenge of NP-hard problems together with additional complications motivates the development of novel heuristics (Jerald, Asokan, Prabaharan, and Saravanan, 2005; Zegordi, Abadi, and Nia, 2010; Rahman, Sarker and Essam 2015; Celebi, 2015). All of these authors considered NP-hard scheduling formulation but none addressed stamping related die set and inventory constraints.

Several authors have addressed scheduling for automotive stamping with die set constraints. Barlatt, Cohn, Fradkin, Gusikhin and Morford (2009) proposed models based on composite variables to derive solutions for production planning problems without inventory constraints, using an automotive stamping facility scheduling example. Barlatt, Cohn and Guiskhin (2010) developed a "test and prune" algorithm for shift selection and task sequencing relating to stampings. Barlatt, Cohn, Gusikhin, Fradkin, Davidson and Batey (2012) investigated the stamping plant scheduling at the Ford Motor Company involving labor and die set constraints but no inventory constraints. Gencosman, Ozmutlu, Ozkan and Begen (2014) used mixed integer programming and constraint programming methods to address parallel stamping scheduling problems with die set constraints but also did not include inventory constraints.

Our case study features a large stamping press which attempts to consistently satisfy the demand of a downstream welding line. Prior to our project, the scheduling for the stamping press was performed manually. The data used here in the study derive from typical 110-hour and 50-hour operation planning windows for the plant. The operators in

charge of scheduling the press admitted difficulty in simultaneously addressing the starvation avoidance, inventory constraints, and setup reduction objectives.

This Chapter is organized as follows. In Section 2.2, we present the assumptions and the key elements of the problem along with a mathematical programming formulation. In Section 2.3, alternative methods from the literature including branch and bound, earliest due date (EDD) scheduling, and genetic algorithm are described. In Section 2.4, rigorous results are also provided for problem instances in which the methods based on generalized EDD scheduling will produce optimal schedules. In Section 2.5, a heuristic which combines genetic algorithm and earliest due date scheduling is proposed.

2.2 Problem Definition

Scheduling problems in the literature are often expressed in terms of jobs with processing times, completion times, due dates, and setup times. In our problem, we have significant inventory storage constraints for each part type. Therefore, it is more convenient to formulate our problem in terms of inventory positions at various times for each part. We assume that the demand is known for the entire scheduling period and initial inventory levels for all part types are given. We also assume that each part type has a limit on the number that can be produced in a single period.

In automotive stamping, different types of parts are produced by the same stamping die or “die-set” mounted on the press. Multiple related parts within the same die-set can be produced at the same time, which we call a “subfamily” of parts. One die-

set can accommodate at least one or more subfamilies. Switching from one subfamily to another within the same die-set is easy and does not affect the production, whereas switching from a subfamily in one die-set to a subfamily in another die-set, requires a significant investment in personnel time. At the same time, while the operation of switching dies is labor intensive, almost all of the labor can be done off-line while the press is running. This explains why we assume zero delay from a changeover.

The produced parts are stored in baskets or racks, based on their types. For example, part 1 from die-set 1 and part 22 from die-set 5 might have a similar shape and be stored in the same type of rack. Basket availability for a particular set of parts (based on their shapes) restricts how much of that part can be produced. When there is demand, baskets become available based on the rate at which the parts are consumed by the process downstream. We assume the basket capacities are given.

2.2.1 The Objective Function

The primary objective is to minimize the number of starved periods. If there is insufficient inventory for a particular part, it signifies that the downstream line would lose production. This starvation concern is similar to the classical tardiness objective considered in the literature (Monma and Potts 1989).

Because the changeover of die-sets is labor-consuming, a lower number of die-set changeovers is desirable. Therefore, our objective is to minimize the weighted sum of starved periods and the number of changeovers. The integer programming formulation presented next includes the features discussed above.

2.2.2 The Mathematical Model

We use the following notation for the problem parameters and variables.

Parameters:

N: Total number of part types

D: Total number of die-sets

K: Total number of baskets

T: Total time period

F: Total number of subfamilies

L_{fd}: Binary, 1 if subfamily f is in die-set d, 0 otherwise where $f \in \{1 \dots F\}$, $d \in \{1 \dots D\}$

S_{if}: Binary, 1 if part i is in subfamily f, 0 otherwise where $i \in \{1 \dots N\}$, $f \in \{1 \dots F\}$

B_{ki}: Binary element, 1 if part i goes to basket k, otherwise 0 where $i \in \{1 \dots N\}$,

$k \in \{1 \dots K\}$

β_k^{init}: Initial space availability in basket $k \in \{1 \dots K\}$

I_i^{init}: Initial inventory of part $i \in \{1 \dots N\}$

β_k^{max}: Capacity of basket $k \in \{1 \dots K\}$

Q_{it} : Demand of part i at time t, where $i \in \{1 \dots N\}$ and $t \in \{1 \dots T\}$

M : A sufficiently large number

C_c : Cost of changeover of die-set

C_s : Cost of starvation at each period

φ_i : Maximum production of part i in 1 hour

$\chi_{i,d}$: Nominal hours of production of part i and die set d $\quad i \in \{1 \ldots N\}$ and $d \in \{1 \ldots D\}$ (used in the solution methods)

Variables:

θ_{it} : Binary, 1 to select part i for production at time t, otherwise 0, where $i \in \{1 \ldots N\}$ and $t \in \{1 \ldots T\}$

x_{ft} : Binary, 1 to select subfamily f for production at time t, otherwise 0, where $f \in \{1 \ldots F\}$ and $t \in \{1 \ldots T\}$

I_{it} : Integer, inventory of part i at time t, where $i \in \{1 \ldots N\}$ and $t \in \{0 \ldots T\}$

G_{it} : Integer, production quantity of part i at time t, where $i \in \{1 \ldots N\}$ and $t \in \{1 \ldots T\}$

β_{kt} : Integer, basket space in basket k at time t, where $k \in \{1 \ldots K\}$ and $t \in \{0 \ldots T\}$

y_{dt} : Binary, 1 to select die-set d for production at time t, otherwise 0, $d \in \{1 \ldots D\}$, and $t \in \{1 \ldots T\}$

W_{it} : Binary, 1 if $I_{it} \leq 0$, otherwise 0, $i \in \{1 \ldots N\}$ and $t \in \{1 \ldots T\}$

z_{dt} : Binary, 1 if there is a changeover at period t for die-set d, otherwise 0, $d \in \{1 \ldots D\}$ and $t \in \{1 \ldots T\}$

Based on the above assumptions, we developed the following integer programming model for the problem:

$$\text{Minimize } Z = C_s \sum_{i=1}^{N} \sum_{t=1}^{T} W_{it} + C_c \sum_{d=1}^{D} \sum_{t=1}^{T} z_{dt} \qquad (1)$$

Subject to:

$G_{it} \leq \varphi_i \sum_{i=1}^{N} \theta_{it} \ \forall i \in \{1 \ldots N\}, \quad t \in \{1 \ldots T\}$ (Production amount constraints) (2)

$G_{it} \leq (\sum_{k=1}^{K} B_{ki}) \beta_{kt} \ \forall i \in \{1 \ldots N\}, \ t \in \{1 \ldots T\}$

$G_{it} \geq 0$, $G_{it} \in \mathbb{Z}^{+}$

$\sum_{d \in D} y_{dt} = 1 \ \forall t \in \{1 \ldots T\}$ (Die-set constraint) (3)

$\sum_{f \in \{1..F\}} x_{ft} \ \leq 1 \ \forall t \in \{1 \ldots T\}$ (Subfamily constraints) (4)

$x_{ft} \leq \left(\sum_{d=1}^{D} L_{fd}\right) y_{dt} \ \forall \ t \in \{1 \ldots T\}$

$\theta_{it} \leq \left(\sum_{f=1}^{F} S_{if}\right) x_{ft} \ \forall \ t \in \{1 \ldots T\}$ (Part constraint) (5)

$x_{ft} \in \{0,1\}, \ \ y_{dt} \in \{0,1\}$

$I_{it} = I_{i\,t-1} + G_{it} - Q_{it} \ \forall i \in \{1, .. N\}, t \in \{1 \ldots T\}$ (Inventory constraints) (6)

$I_{it} \leq M(1 - W_{it}) \ \forall i \in \{1 \ldots N\}, t \in \{1 \ldots T\}$

$I_{it} \geq -MW_{it} + 1 \ \forall i \in \{1 \ldots N\}, t \in \{1 \ldots T\}$

$I_{i0} = I_{i}^{init} \ \forall i \in \{1 \ldots N\}$

$W_{it} \in \{0,1\} \ \forall i \in \{1 \ldots N\}, t \in \{1 \ldots T\}$

$I_{it} \in \mathbb{Z} \ \ \forall i \in \{1 \ldots N\}, t \in \{1 \ldots T\}$

$z_{dt} \geq (y_{d\,t+1} - y_{dt}) \;\; \forall\, d \in \{1 \ldots D\}, t \in \{1 \ldots T-1\}$ (Changeover constraints) (7)

$z_{dt} \leq y_{d\,t+1} \forall\, d \in \{1 \ldots D\}, t \in \{1 \ldots T-1\}$

$z_{dt} \leq 1 - y_{dt} \forall\, d \in \{1 \ldots D\}, t \in \{1 \ldots T-1\}$

$z_{dt} \in \{0,1\}$

$\beta_{kt} = \beta_{kt-1} + (\sum_{i=1}^{N} B_{ki}) Q_{it} - (\sum_{i=1}^{N} B_{ki}) G_{it} \;\forall\, k \in \{1 \ldots K\}, t \in \{1 \ldots T\}$ (Basket constraints) (8)

$\beta_{k0} = \beta_k^{init} \;\forall k \in \{1 \ldots K\}$

$\beta_{kt} \leq \beta_k^{max} k \in \{1 \ldots K\}, t \in \{1 \ldots T\}$

The objective in Equation (1) balances starvation and setup labor costs. Constraint set (2) guarantees that the production amount of each part at every period is less than the minimum of the maximum allowable limit and the available storage space. Constraint (3) suggests that only one die-set can be chosen at each period. The constraint set (4) signifies that at most one subfamily can be selected at any given period and that subfamily has to be from the die-set in operation at that period.

Equation (5) guarantees that the part in production is chosen from the subfamily already selected. Constraint set (6) connects the production amount, part demand and part inventory. It also counts the number of starved periods (periods for which $I_{it} \leq 0$). Constraint set (7) provides a way to count the number of die-set changeovers. The relation among basket space availability, production amount and the demand of each part is shown in constraint set (8).

2.3 Alternative Solution Methods

In this section, we describe four alternative methods for solving the integer program in Equations (1) through (8). These are branch and bound, an earliest due date heuristic, and two types of genetic algorithms.

2.3.1 Branch and Bound

The integer program in Section 2.2 can be solved using branch and bound, e.g., Pochet and Wolsey (2006). The branch and bound algorithm starts with the solution of the linear relaxation of the initial model. The linear program obtained from the relaxation is easy to solve, but may render fractional optimal values for integer variables. In that case, the solution of the linear relaxation works as a *lower bound* of the original formulation.

The original feasible region is then partitioned and the algorithm proceeds by solving the linear relaxations of these *branches.* Any solution in which the relaxed integer variables have integer values is feasible to the original problem, and its objective function value provides an *upper bound* of the optimal objective function value. With complete enumeration, the final solution provided by the algorithm is exact, but if the enumeration is truncated, it provides an approximate solution. In the latter case, *duality gap* is used to measure the quality of the solution.

The running time of this algorithm depends on the size of the problem as well as the formulation. Commercially available software products like CPLEX pre-solve the initial model to obtain an improved initial lower bound. CPLEX also add custom constraints or "cuts" automatically for faster enumeration. In the case study described in Chapter 3, we implemented branch and bound using IBM ILOG CPLEX 12.6.2 on a i7-3770S CPU 3.10Ghz, with 8.00GB of memory.

2.3.2 Generalized Earliest Due Date Methods

In earliest due date (EDD) scheduling, parts with the earliest due dates are produced first (Smith, 1956). In our problem, the due dates are the periods in which the inventory would become zero or negative such that the downstream department would be starved.

2.3.2.1 Urgent Demand Scores

To address these complications, we propose a "generalized earliest due date" (GEDD) method. This method assigns continuous urgency scores relating to the due dates of both the subfamilies and the die-sets. Consider that each subfamily and die-set is, in general, composed of r part types. For each part type i, there is a number of periods, t_i, until the inventory would become zero or negative assuming there is no additional production. We use parentheses to indicate the ordered numbers of periods from lowest to

highest in the relevant set: $t_{(1)}$,$t_{(2)}$,… The "urgent demand score" (UDS) associated with either the subfamily or die-set is:

$$UDS = t_{(1)} + s\frac{t_1+\cdots+t_r}{r}. \quad (9)$$

where s is a small number, e.g., 0.001 used in our numerical examples. For example, consider a subfamily containing two parts and the inventory positions in Table 1. Then, t_1 = 7, t_2 = 5, $t_{(1)}$ = 5, and $t_{(2)}$ = 7 and the UDS = 5.006. For parts with no demand within the horizon, T, we assume $t_i = 2T$.

Table 2.1 Example of part UDS and Die-set UDS

	7am	8am	9am	10am	11am	12am	1pm
Part 1 Inventory	1200	800	800	400	200	100	-100
Part 2 Inventory	900	600	300	100	-100	-200	-400

2.3.2.2 The GEDD Heuristic

The generalized earliest due date (GEDD) method has steps as follows.

Step 1: Start the time period at $t = 1$.
Step 2: Select the die-set with minimum UDS for production. The duration is given by $q\delta_d$ where q is an integer multiple for the given nominal hours δ_d of operation with the die-set d.
Step 3: Select the subfamily f with minimum UDS value within the selected die-set d.
Step 4: The production amount G_{it} for each part i in the selected subfamily f is the minimum of the hourly rate of production φ_i of the part and the available basket space β_{it}, hence $G_{it} = \min(\beta_{it}, \varphi_i)$. Update the UDS for each part, subfamily, and die-set.
Step 5: Steps 3-4 are repeated for the duration of δ_d or until the final period is reached. If it is reached, stop. Otherwise, set $t = t + \delta_d$ and go to *Step 2*.

GEDD Heuristic

2.3.3 A Random Keys Elitist Genetic Algorithm with Bernoulli Cross-Overs

Meta-heuristics are general purpose optimization methods relevant to many problem types. Genetic algorithms (GAs) are meta-heuristics that have, in recent years, been widely adopted for solving scheduling problems (Bean, 1994; Husbands, 1994; Norman and Bean, 1997; Allahverdi, Ng, Cheng and Kovalyov 2006). Alternative types of genetic algorithms have been proposed such as gendered methods for scheduling problems (Zegordi, Abadi, and Nia, 2010). Because Equations (1)-(8) define a multiple choice integer program, we select the multiple choice genetic algorithm in Ben Hadj-Alouane and Bean (1997) for our comparison. In particular, set (3) and (4) contain two types of multiple choice constraints (on die-set and subfamily selections). The Ben Hadj-Alouane and Bean (1997) method is reviewed next to facilitate our comparison.

2.3.3.1 Overall Framework

The Ben Hadj-Alouane and Bean (1997) algorithm starts with initializing a population of "chromosomes" or candidate solutions which are essentially vectors with each element as uniform (0,1) random numbers. Each chromosome is then mapped to the decision variable values and the objective function is evaluated. The algorithm then proceeds from one generation to the next by copying over the top N_e solutions. Then, Bernoulli crossover and tournament selection are used to create N_c solutions in the next iteration.

Bernoulli crossover involves selecting two solutions randomly (equal probabilities) in the current solution. Then, two new solutions are created by Bernoulli trial for each element in the vectors with probability, p_B. In our examples, we use $p_B = 0.8$. The tournament then picks the superior of the two created solutions and places it into the next generation. Finally, the remaining solutions are generated using uniform (0,1) random numbers (immigrants). The details of the steps are as follows:

Step 1: (Initialize) Create the first population using random numbers, #generation=1.
While {#generation ≤ total number of generations}
Step 2: **Loop Over** every chromosome in the population
Map the population to schedule and evaluate the objective function.
End Loop
Step 3: Sort the population based on objective function value.
Step 4: Copy the top N_e chromosomes to next generation.
Step 5: Perform Bernoulli crossover to generate N_c solutions in the next generation.
Step 6: Generate $N_p - N_e - N_c$ solutions randomly (immigrants).
End While

Algorithm: GA

2.3.3.2 Initialization

The initial population is created with chromosomes of length $3T$, where T is the number of periods. Each element (allele) of the chromosome vector is a uniformly distributed random number between 0 and 1.

2.3.3.3 Solution Mapping Overview

The proposed methods differ only in the "mapping" which is the translation of a generic vector to a stamping schedule. Mapping procedures transform candidate solutions which are vectors of numbers between zero and one into solutions to the scheduling problems. This can be critical to the success of the derived schedules (Bean, 1994; Norman and Bean, 1997). For the purposes of comparison, we consider two mapping approaches. The first is described here and the second uses earliest due date order which is the basis for the proposed genetic algorithm with generalized earliest due date (GAGEDD) method described in the next section.

The first mapping uses the first number in the chromosome, $a(1)$, to multiple-choice select the die-set number. With D die-sets, the Ben Hadj-Alouane and Bean (1997) selection is $\lfloor Da(1) + 1 \rfloor$, where $\lfloor i \rfloor$ is the floor operation. For example, with $D = 5$ die-sets, $a(1) = 0.01$ would select the first die-set, since $a(1) < \frac{1}{5}$.

Then, the next allele selects the duration that this die-set is used. We discuss the details of the duration selection below. The third allele selects the first part subfamily within the selected die-set. This is similar to the die-set multiple choice selection. This procedure is repeated until the die-set duration is reached. An example is given in the

next section. Then, the next die-set is selected and the procedure is repeated until all T periods in the planning horizon are specified. The remaining portions of the chromosome play no role in the schedule construction.

2.3.3.4 Mapping of Durations

In the selection of the durations for die set d, we seek to leverage expert knowledge of the stamping operations process and the nominal number of production hours for a given part j which is given as $\chi_{j,d}$. Assume that the maximum of the nominals for all the associated parts (the set L_d) is

$\rho(d) = [\max_{j \in L_d} \chi_{j,d}]$. So for allele i and die-set d, the duration, τ_d, is:

$$\tau_d = \begin{cases} 2 & for\ 0 \leq a(i) \leq 0.2 \\ \rho(d) & for\ 0.2 < a(i) \leq 0.5 \\ \lfloor 1.5\ \rho(d) \rfloor & for\ 0.5 < a(i) \leq 0.75 \\ 2\ \rho(d) & for\ 0.75 < a(i) \leq 1 \end{cases}. \quad (10)$$

The subsequent τ_d elements of the chromosomes will be used to determine which subfamily of the die-set will be produced for each of the τ_d hours. For example, assume that there are 5 die-sets, and the first die-set has 2 subfamilies. The nominal hours of production for the first die-set is $\rho(1) = 4$. In this example, the chromosome fragment in the top vector of Fig. 1, is mapped to a partial schedule shown in the bottom vector. The chromosome fragment is encoded in the following way. Die-set 1 is the selected die-set, which would be mounted for 4 hours. For the next 4 periods (hours), subfamilies 1 and 2 would be selected for production in the order shown (2, 1, 1, 2). This procedure is repeated until the number of periods of scheduling is covered. Expert knowledge is used to limit the duration options, e.g., die-sets are almost never mounted for fewer than two

periods or more than twice the nominal. Investigating a wider variety of durations is a topic for future research.

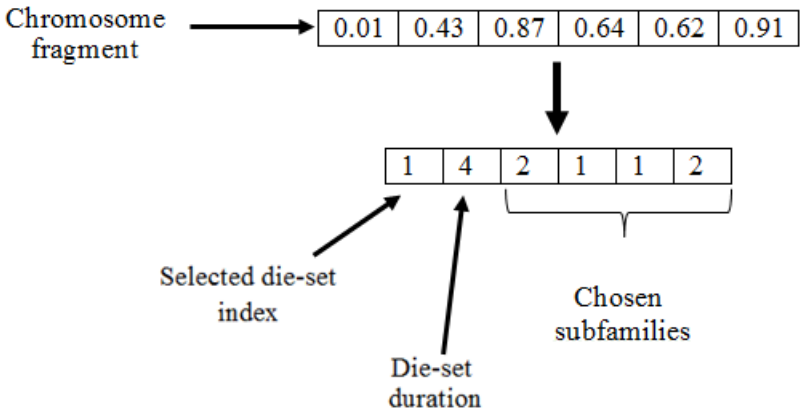


Figure 2.1 Illustration of the first mapping of die sets in parts in a genetic algorithm.

2.3.3.5 Evaluation of Objective Function

The objective function in Equation (1) is the weighted sum of the number of starved periods and the number of changeovers. The associated costs are C_s and C_c respectively. In our numerical studies, we weighted the two counts approximately at 10:7 for all the methods (Branch & Bound, GA, GEDD and GAGEDD in Chapter 3). The weights or relative costs are decided based on expert opinion on stamping operations, and they are kept the same in the objective functions for all the solution methods so that the methods can be compared with one another. This effectively gives higher weight to starvation since every part could, hypothetically, cause starvation in multiple periods.

2.4 Rigorous Results

The following results are extensions of the results in Monma and Potts (1989). The first result relates to the ordering of parts within die-sets or "die-set fragments" in optimal schedules.

Theorem 1. For the objective in Equation (1), there exists an optimal schedule, in which the die-set fragments containing no tardy jobs are ordered according to the non-decreasing order of their respective die-set UDS values.

Proof. Let us assume there is an optimal schedule in which the die-set fragments are not ordered in the non-decreasing order of UDS. Without loss of generality let us assume that die-set i and $i+1$ are ordered such that $UDS_i > UDS_{i+1}$. If the order of these two die-sets is inter-changed such that $(i+1)^{th}$ die-set becomes the i^{th} die-set and vice-versa, then the parts with higher urgency are not produced any later than they already were in the original schedule. Also the number of die-set changeovers remains the same. Hence ordering the die-set fragments according to their non-decreasing UDS value will also be optimal.

The second result points to the ordering of subfamilies within die-set fragments in the optimal schedule.

Theorem 2. For the objective in Equation (1), there exists an optimal schedule in which within the die-set fragments, the subfamilies containing no tardy jobs are ordered according to the non-decreasing order of their respective UDS values.

Proof. Let us assume there is an optimal schedule in which subfamilies in a die-set fragment are not ordered in the non-decreasing order of UDS. Without loss of generality let us assume that subfamilies i and $i+1$ are ordered such that $UDS_i > UDS_{i+1}$. If the order of these two subfamilies are inter-changed such that $(i+1)^{th}$ subfamily becomes the i^{th} subfamily for production within the selected die-set fragment and vice-versa, then the parts with higher urgency are not produced any later than they already were in the original schedule. Since the orderings of subfamilies are changed within individual die-sets, the number of die-set changeovers remains the same. Hence the ordering of the subfamilies within a die-set fragment according to their non-decreasing UDS value will also be optimal.

These results do not establish the optimality of GEDD schedules for stamping operations scheduling and are irrelevant for jobs that cause starvation in the optimal schedules. Yet, consider an optimal solution to the problem in Equations (1)-(8). The two results do apply to the jobs in that optimal schedule which never have any negative inventories over the horizon. For these jobs, the schedule will resemble, subjectively, a schedule that the GEDD heuristic generates for some value of q, e.g., $q = 2$. This implies that the associated subfamilies will follow UDS ordering and the die-set fragments will also have UDS ordering. Only the durations for the die-set fragments may differ between the optimal solution and the GEDD-generated schedule.

2.5 **Genetic Algorithm Generalized Earliest Due Date Method (GAGEDD)**

The proposed method combines GEDD heuristic with genetic algorithm resulting in Genetic Algorithm Generalized Earliest Due Date (GAGEDD). This method uses a different random key mapping based on the GEDD structure to generate solutions that follow the UDS ordering of die-sets and part families. In a manner similar to the first mapping (described in Sections 2.3.3 and 2.3.4), this mapping procedure alternatively selects the die-set and the duration. Here $a(i)$ refers to the i^{th} element of the chromosome vector and $i = 1 \ldots 2T$. The subfamily is chosen by the minimum UDS given in Equation (9). We use $\mathcal{D}$ to denote the set of die-sets.

The mapping which generates the "genetic algorithm generalized earliest due date" (GAGEDD) method is detailed below. This approach of generating a schedule is similar to the GEDD method. The difference is that for values greater than γ the genetic algorithm will override the GEDD and select a die-set according to the probability determined by the allele value instead of selecting the die-set with the lowest UDS value.

Initialize $i = 1$ (the selected allele) and $t = 1$ (the period).
While $\{t \leq T\}$
Select die-set selection (d) using.
If $\{a(i) \leq \gamma\}$ Then, $d = \text{argmin}_{d \in \mathcal{D}} UDS_d$.
Else $d = \left\lfloor D \frac{a(i)-\gamma}{(1-\gamma)} + 1 \right\rfloor$.
$i = i + 1$.
Select the duration (τ_d) using Equation (10), $k = 1$, and $i = i + 1$.
While $\{k \leq d\}$
Select the subfamily with minimum UDS value within the selected die-set.
Select the production amount $G_{it} = \min(\beta_{it}, \varphi_i)$ and $t = t + 1$.
End While
End While

Algorithm GAGEDD

2.6 Summary

In this Chapter we have modeled an automotive stamping scheduling problem as an integer program. Then we have described a method to use random key genetic algorithm (GA) to solve this problem. We have also identified a way to account for the complications created by the die-sets and subfamilies in the stamping problem. Since it is a production scheduling problem, the problem structure is compatible with the earliest due-date (EDD) heuristic, and we have developed a generalized earliest due-date (GEDD) for the problem. Next we have introduced a hybrid method GAGEDD, which uses the GEDD heuristic optimized by the GA so that the overall cost is minimized. In the next chapter, we will compare the results generated by different methods on an instance of the automotive stamping problem.

Chapter 3: An Automotive Stamping Case Study

3.1 Case Study Results

In this chapter, we have implemented the methods described in the previous chapter, to schedule production of a stamping press of an automotive manufacturing plant. There are 49 different parts to produce, in 23 different die-sets, 36 subfamilies and the storage consists of 14 different types of baskets. To compare the results of the different solution methods, we have varied two factors of the problem, the number of periods and the initial inventory. The number of periods is varied at two levels, 50 and 110. The initial inventory is varied at three levels, 25, 50 and 75% of the total demand of the parts. The data for demand, die-set, subfamily and basket have been directly taken from a typical workday scenario of the manufacturing plant.

Table 3.1 shows a replicated full factorial computational experiment. It includes the outcomes of 6 different methods when applied to the scenarios of the case study. The two main outputs are the number of starved periods and the number of die-set changeovers. These are compared along with the corresponding runtime of the methods.

The best solutions in terms of the overall objective function value are shown in the bold font. The first method, B&B, is a conventional branch and bound algorithm, implemented in CPLEX version 12.6.2 using the mathematical model described in Section 2.2. Admittedly, there are many ways to implement the model and more efficient implementations may exist. Yet, for both 50 and 110 period scenarios, the runtime for our best implementation exceeded 50,000 seconds. So we restricted the runtime to 10,000 and 20,000 seconds for 50 and 110 period scenarios respectively. Table 2 shows the best value at the cutoff time.

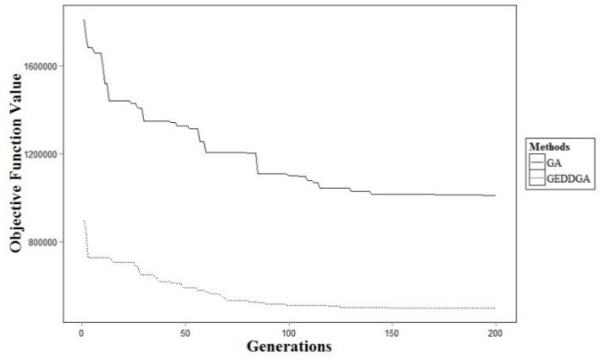
Next, the GEDD heuristic is used with $q = 1$. This means die-sets are mounted on the press for the specified nominal periods designated for each die-set by the plant operators. The same heuristic is repeated with $q = 2$, allowing the die-sets to be mounted for twice the specified number of nominal periods. Both B&B and GEDD heuristic are deterministic. The genetic algorithm (GA) method of Ben Hadj-Alouane and Bean (1997) and GAGEDD have inherent randomness which explains why 10 replications are run for each of the scenarios. . The average values from 10 replications are shown with the standard deviations in parentheses.

Table 3.1 Comparison of alternative solution methods for six scheduling problems with 49 parts.

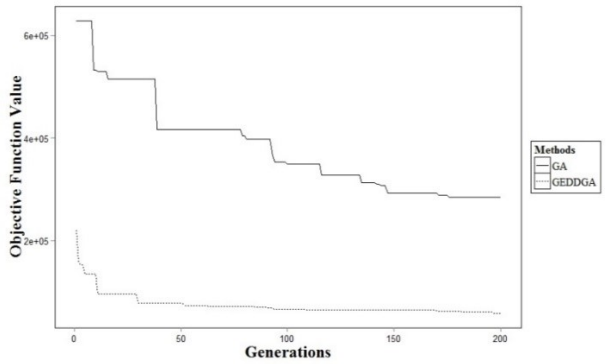
	(%) Initial Inventory	(*N*) Time Periods	B&B	GEDD (*q*=1)	GEDD (*q*=2)	Ben-Alouane & Bean (GA)	GAGEDD ($\alpha = 1, \gamma = 0.75$)	B&B^2
	25	50	**511.00 (0.00)**	907.00 (0.00)	1058.00 (0.00)	870.70 (33.69)	632.10 (17.79)	533.00 (0.00)
# Starved	50	50	**86.00 (0.00)**	333.00 (0.00)	887.00 (0.00)	435.90 (29.15)	136.70 (21.68)	282.00 (0.00)
Periods	75	50	**0.00 (0.00)**	57.00 (0.00)	163.00 (0.00)	100.80 (15.02)	0.50 (0.81)	2.00 (0.00)
Average	25	110	603.00 (0.00)	840.00 (0.00)	1068.00 (0.00)	947.30 (56.29)	**442.10 (29.91)**	4321.00 (0.00)
(SD)	50	110	**0.00 (0.00)**	1327.00 (0.00)	1514.00 (0.00)	151.80 (50.18)	11.60 (9.09)	2937.00 (0.00)
	75	110	0.00 (0.00)	0.00 (0.00)	3.00 (0.00)	1.00 (2.19)	**0.00 (0.00)**	0.00 (0.00)
	25	50	**36.00 (0.00)**	27.00 (0.00)	21.00 (0.00)	17.70 (1.42)	28.70 (1.49)	41.00 (0.00)
#	50	50	**39.00 (0.00)**	26.00 (0.00)	8.00 (0.00)	19.20 (1.47)	35.50 (2.77)	47.00 (0.00)
Changeovers	75	50	**23.00 (0.00)**	28.00 (0.00)	21.00 (0.00)	20.00 (0.89)	26.20 (0.60)	25.00 (0.00)
Average	25	110	80.00 (0.00)	37.00 (0.00)	30.00 (0.00)	30.80 (2.52)	**58.10 (3.39)**	5.00 (0.00)
(SD)	50	110	**36.00 (0.00)**	24.00 (0.00)	12.00 (0.00)	31.40 (1.62)	44.30 (3.41)	4.00 (0.00)
	75	110	24.00 (0.00)	29.00 (0.00)	27.00 (0.00)	26.80 (1.89)	**24.00 (0.00)**	40.00 (0.00)
	25	50	>10,000	1.00 (0.00)	2.00 (0.00)	1,233.70 (14.91)	477.50 (6.17)	500.00 (0.00)
Run	50	50	>10,000	2.00 (0.00)	1.00 (0.00)	1,224.70 (20.51)	486.90 (9.58)	500.00 (0.00)
Time (sec)	75	50	>10,000	2.00 (0.00)	2.00 (0.00)	1,236.40 (14.30)	465.10 (4.91)	500.00 (0.00)
Average	25	110	>20,000	4.00 (0.00)	3.00 (0.00)	3,537.40 (38.46)	1,247.90 (9.12)	1200.00 (0.00)
(SD)	50	110	>20,000	2.00 (0.00)	2.00 (0.00)	3,643.40 (44.34)	1,204.00 (9.35)	1200.00 (0.00)
	75	110	>20,000	2.00 (0.00)	2.00 (0.00)	3,769.70 (12.43)	1,025.20 (9.34)	1200.00 (0.00)

The GAGEDD is run with parameters $\alpha = 1$ and $\gamma = 0.75$. The number of generations for GAGEDD is set to 200 and population size is set to 200. As shown in Table 3.1, GAGEDD results into fewer starved periods than both the GA and the GEDD heuristics. GAGEDD also takes significantly less time than GA, for both 50-period and 110-period problems. The branch and bound (B&B) method provides the best solutions in terms of number of starved periods and overall objective function values for 4 out of 6 cases and ties with GAGEDD in the case with 75% initial inventory and 110 periods. But execution times for B&B are much longer than GAGEDD, which could be prohibitive for real applications. To compare GAGEDD and branch and bound methods under time restrictions, we ran the branch and bound for 500 seconds for 50 period cases and 1200 seconds for 110 period cases. The results are shown in the last column ($B\&B^2$) of Table 2. Apparently GAGEDD outperforms $B\&B^2$ for all the cases except for the case of 25% initial inventory and 50 period.

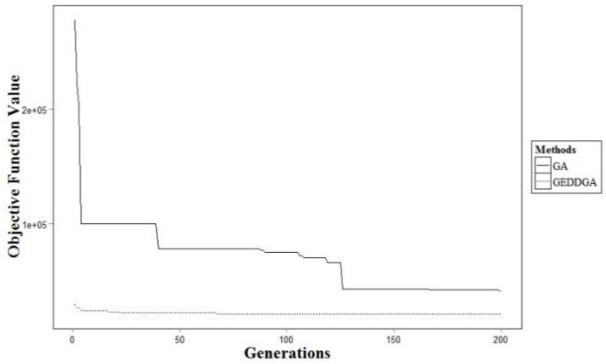
Figure 3.1 shows the rate of reduction in objective function value with respect to number of iterations for GA and GAGEDD in 110 period scenarios with 25%, 50% and 75% initial inventory.



(a)



(b)



(c)

Figure 3.1 Change in objective function value with number of generations in GAGEDD and GA with initial inventory 25% (a), 50%, (b) and 75(%) of total demand over 110 period (c).

As the result of the prioritization step, the GAGEDD starts with a much smaller objective function value, and the value remains lower than that generated by GA through the designated number of iterations. The GA, GEDD and GAGEDD are coded in Visual Basic for Application on MS Excel 2010, and run on the same machine as mention in Section 2.3.1.

3.2 Analysis of Case Study Results

Analysis of Variance for all three responses shows that all the main effects and interactions are significant with p-values less than 0.001. This is to be expected since only the genetic algorithm methods are associated with randomness. Therefore, virtually all visible differences in the interaction plots are associated with statistical significance.

To observe the effects of different controlling parameters, interaction plots are generated from the full factorial design of Table 3.1. The number of starved periods and number of changeovers are chosen as response variables. Figure. 3.2 shows the interaction among the factors (initial inventory, number of periods and solution methods) with the response variable as the number of starved periods.

From the plot in the top left corner, it is apparent that the lowest number of starvations occurs when the initial inventory is 75% of the total demand, and also with the B&B method and with GAGEDD. There is little interaction between the methods and the number of periods as shown in the bottom left plot. The branch and bound method provides the best solutions for both 50 and 110 period problems, whereas GAGEDD provides the best solution only for the 110 period problems. The bottom right plot shows

there is marginal interaction between initial inventory and number of periods. The best solutions predominantly occur when the initial inventory is 75% of the projected demand.

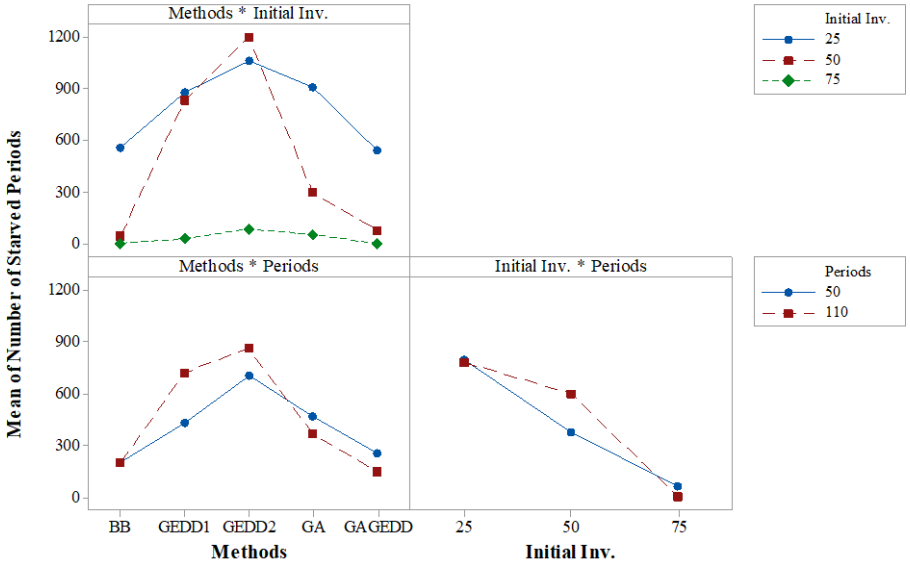


Figure 3.2 Interaction plot with number of starved periods as responses.

Figure 3.2 shows the interaction plots for the same factors with the response variable being the number of changeovers. From the top left plot, it is evident that for this response, there is strong interaction between the initial inventory and methods. This occurs because GEDD performs well (probably optimally, see Section 3.1) when there is ample inventory.

Note that the GEDD 2 heuristic provides the best solutions in terms of number of changeovers. However, GEDD 2 also generates solutions with high numbers of starved periods. This is due to the fact that the ordering is only optimal assuming when there are

no starved periods. As a result, for cases with starvation the overall solution quality is not as good as branch and bound or GAGEDD.

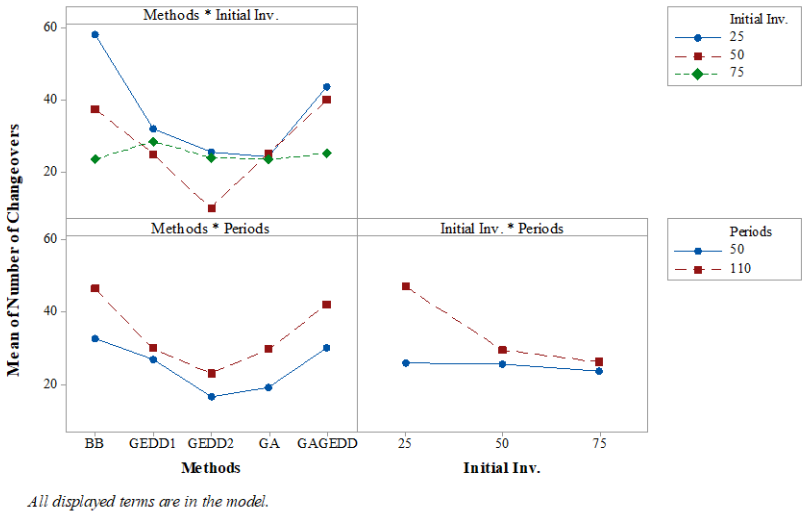


Figure 3.3 Interaction plots with number of changeovers as response.

From both Table 3.1 and figures 3.3-4, it is obvious that the best solutions are provided by B&B and GAGEDD. The B&B method provides the bests results in terms of the number of starved periods and overall objective function value (except in the case with 25% initial inventory and 110 periods), but it may not be a practical choice since the runtimes for branch and bound are above 10,000 seconds. Upon restricting the execution time to 500 seconds for 50 period cases and 1200 seconds for 110 period cases, branch and bound ($B\&B^2$) shows inferior performance than GAGEDD in 5 out of 6 cases as shown in Table 3.1. Table 3.2 shows the number of objective function enumerations for $B\&B^2$ is much higher than that of GAGEDD.

Table 3.2 Comparison of number of objective function enumerations for branch and bound and GAGEDD.

(%) Initial Inventory	(*N*) Time Periods	B&B	B&B^2	GAGEDD
25	50	28,427,089	1,285,128	40,000
50	50	29,310,372	1,248,696	40,000
75	50	44,416,486	2,112,701	40,000
25	110	22,041,129	332,778	40,000
50	110	26,487,990	3,580,581	40,000
75	110	34,376,425	3,841,485	40,000

3.3 Discussion

In this study, we consider a single machine manufacturing scheduling problem with known demand data motivated by automotive stamping operations. Different parts are to be processed by the machine and there are constraints related to the arrangements of parts and subfamily of parts within die-sets. For the real-world case study, we seek methods that generate acceptable solutions within reasonable amount of time (e.g. 20 minutes or 1200 seconds for 110 period cases). The aim is to provide the stamping schedulers with a decision support system that they can use every morning to plan the production for the day(s) ahead. They should also have enough time for reruns whenever necessary.

We compare four methods using a full factorial computational experiment. Relating to speed, a simple heuristic based on earliest due dates dominates. Yet, solution

quality was poor for cases of interest in which the available inventory amounts cause some level of starvation. Also, it is understandable that CPLEX and the integer solver eventually obtain the exact optimal solution since our objective is linear and zero *duality gap* is possible. The generic genetic algorithm that we studied provided acceptable solutions in some cases but was computationally slow. Our comparison of the computation costs for branch and bound and the generic genetic algorithms suggests that the proposed GAGEDD heuristic better fits the real-world requirements. The GAGEDD methods is *the only method to consistently achieve near optimal solutions in twenty or fewer minutes which is practically needed by the schedulers in the stamping company.* Moreover, it is possible to increase the number of generations and populations easily if more time can be spent to get better solutions during the time of scheduling.

There are a number of opportunities for future research. Manufacturing processes such as forging, die-casting, and injection molding share at least some of the characteristics with metal stamping process (Groover, 2011) described in this article. All of these processes include expensive dies with changeover costs and inventory constraints for finished parts. Therefore, while the specific formulation addressed here relates to automotive stamping operations, we believe our method offers the possibility of extensions and additional applications.

The methods can also be extended to explore similar problems with stochastic demand. It may be interesting to broaden the scope of optimization by involving purchasing decisions relating to baskets and inventory storage. By studying the scheduling problems, some useful reductions or increases in capacity might be identified.

Operations like automotive stamping have complicated part family interrelationships, setup cost structures, and inventory constraints. It is unlikely, perhaps that all the machines and sub-lines in a complex organization can be scheduled successfully in a fully centralized way. Yet, it may be advisable to explore a range of centralization levels. The elements of the formulation presented here can be extended to multi-line, multiple organization scheduling problems.

It is not entirely appropriate to compare the execution times for the two methods, since branch and bound is implemented in CPLEX and GAGEDD is coded in VBA which is generally associated with inefficient computation. Yet, Table 3.2 shows the number of times the objective functions are enumerated for each of the experiments. For GAGEDD, both the generation size and population size are limited to 200, limiting the number of enumerations to 40,000. Some computational effort is spent for mapping of the chromosomes for each of these enumerations. On the other hand, in case of branch and bound methods, the number of enumerations ranges from 22 million to over 44 million for B&B and from 0.3 million to over 3.8 million for B&B^2, with a linear programming problem being solved in each iteration. This shows that the computational cost for branch and bound is much higher than GAGEDD.

Chapter 4: Review of Markov Decision Process and Variants

4.1 Introduction

In everyday life, we make a sequence of decisions which are associated with either costs or rewards. Our objectives are commonly to minimize the expected cost or maximize the expected rewards over an extended period of time. In some cases, the rewards are manifested at a later point of time, while the action incurs some immediate cost. We base our judgements on our knowledge of the particular situation, and the chances of earning the rewards. This procedure can be modeled as a Markov Decision Process (MDP) (Puterman, 2014). In this chapter, we review the Markov Decision Processes (MDPs)and their extension to more uncertain domains. Uncertainty sources can include not knowing which transitions will occur, the state of the system, or the parameters accurately. We also review Partially Observable Markov Decision Process (POMDP) where there is uncertainty about the state of the system, and a point based solution procedure PERSEUS (Spaan, Vlassis 2005) to solve POMDP models. We then discuss Bayes Adaptive Markov Decision Processes (BAMDPs) which take into account the uncertainty in parameters. We reviewed the background of BAMDP and

reinforcement learning methods to put it in context to our solution approaches for cyber vulnerability maintenance problems described in chapters 5 and 6.

4.2 Markov Decision Processes

In sequential decision problems, decisions are made at certain time epochs. The assumption is that the system retains the Markovian property, i.e. the state of the system at the present time epoch is only dependent on the state and the action taken at the previous epoch. The possibility of transitioning from one state to another can be quantified using transition probabilities. Let us suppose that the system has N states, $s_1, .., s_N$, and there are U possible actions $a_1, .., a_U$ that can be taken at each epoch. The probability of transitioning from state i to state j upon taking action a is p_{ij}^a, where $i = 1, .., N$, $j = 1, .., N$, $a = 1, .., U$. Let S be the set of all states and A be the set of all actions. If r_{ij}^a is the reward for transitioning from state i to state j upon taking action a then the expected reward at some epoch t will be being in state i and taking action a will be $r_t(i, a) = \sum_{j=1}^{N} p_{ij}^a r_{ij}^a$.

In MDP a policy $a|i$ suggests the action a should be taken at state i. Denote the random system states as $Y_1, \ldots, Y_\infty$. The objective function for this problem is the expected sum of the discounted rewards:

$$\max_{\pi(1),\ldots,\pi(s)} \mathrm{E}_{Y_1,\ldots Y_\infty}\left[\sum_{i=1}^{\infty} \gamma^i (r_{Y_{i-1}Y_i}^{\pi(Y_{i-1})} + \epsilon)\right] \qquad (1)$$

where ϵ is a random reward generally taken with mean zero on top of the expected rewards, r_{ij}^a.

The optimal value function at epoch t is given by the Bellman equation

$$V_t^*(i) = \max_a \sum_{j=1}^{N} p_{ij}^a (r_{ij}^a + \gamma V_{t-1}^*(j)) \text{ for all } i = 1, .., N. \quad (2)$$

In the above equation, $0 < \gamma \leq 1$ is a discount factor.

MDPs can be solved using value iteration and policy iteration algorithms (Puterman 2014). We review these because they share elements with the more general alogirthms that we describe later. In value iteration (Bellman 1957), the algorithm loops through the states, updating the value of the states using the Bellman equation. The method finally converges to an optimal value, the associated policy becoming the optimal policy.

Initialize $V(s)$ arbitrarily
$\Delta = 0$
While $\{\Delta < \sigma\}$
  For each $s \in S$
    $v = V(s)$
    For each $a \; \forall \; i \in A$
      $Q(s,a) = \sum_k p(s,a,s_k) \, (R(s,a,s_k) + \gamma V(s_k))$
    End for
  $V(s) = \max_j Q(s,a)$
  $\Delta = \max(\Delta, |v - V(s)|)$
  End for
End While

Algorithm: Value iteration

In policy iteration (Howard 1960), policies are generated randomly, which are then evaluated and improved to converge to a final optimal policy. Let $\pi(s), s \in S$ be the policy which is for ordinary MDP, action $a(s) \in A$ to be taken at state s. The value of a

policy $\pi(s)$ is given by $V^{\pi}(s)$. The steps of the policy iteration algorithm are shown below (Otterlo, Wiering 2012).

Initialize $V(s) \in R, \pi(s) \in A \; \forall \; s \in S$ arbitrarily
Evaluate Policy (s) **:**
$\Delta = 0$
While $\{\Delta < \sigma\}$
    For each $s \in S$
        $v = V^{\pi}(s)$
        $V(s) = \sum_k p(s, \pi(s), s_k) \, (R(s, \pi(s), s_k) + \gamma V(s_k))$
        $\Delta = \max(\Delta, |v - V(s_i)|)$
    End for
End While
Improve Policy:
policy.stable=true
For each $s \in S$
        $\mu = \pi(s)$
        $\pi'(s) = argmax_a \sum_k p(s, a, s_k) \, (R(s, a, s_k) + \gamma V(s_k))$
        If $(s) \neq \pi'(s)$: policy.stable = false
End for
If policy.stable: **STOP ; Else Goto Evaluate Policy**

Algorithm: Policy Iteration

In the policy improvement step, the best action is greedily selected based on the value function. The optimal policy is reached when the policy cannot be improved any further. The policy iteration generates a series of alternating policies and value functions, and for MDPs with finite state and action spaces, it converges in finite time.

4.3 Partially Observable Markov Decision Process

When there is uncertainty about system state, the problem becomes a Partially Observable Markov Decision Process (POMDP). A probability distribution $b(i)$ gives the

probability of being in state i, which is called the belief state. In POMDP, instead of directly observing the system state, the decision maker observes an output $o \in O$, with a probability p_{jo}^{a} if the new state is $j \in \{1, .., N\}$ (Smallwood and Sondik 1973). A reward r_{ijo}^{a} is accrued if the system transitions from state i to state j upon taking action a and observing output o. Smallwood and Sondik (1973) assume that this reward is not directly observable. The belief state is updated after each transition, and the updated probability of being in state j is given by:

$$b_j' = \frac{\sum_i b_i p_{ij}^{a} p_{jo}^{a}}{\sum_{i,j} b_i p_{ij}^{a} p_{jo}^{a}} \quad \forall\, j = 1, .., N \tag{3}$$

The above expression can also be written in vector form as $b' = p(b|a, o)$. The denominator is the normalizing constant, which is also conditional probability of observing output o, given action a and belief state b and hence can be summarized as $p(o|a, b)$. The optimal value function in relation to the problem in equation (1) in POMDP is given by

$$V_t^*(b) = \max_a \sum_{i=1}^{N} \sum_{j=1}^{N} p_{ij}^{a} \sum_o p_{jo}^{a} \{r_{ijo}^{a} + V_{t-1}^{*} p(b|a, o)\} \tag{4}$$

Smallwood and Sondik (1973) show that the optimal value function $V_t^*(b)$ is piecewise linear and convex. This property is used to parameterize the value function by a set of vectors (α vectors). Hence the optimal value function can be represented as

$$V_t^*(b) = \max_k \sum_s \alpha_t^k(s) b(s) \tag{5}$$

for a set of vectors $\alpha_t^k = \{\alpha_t^1, \alpha_t^2, ..\}$.

Effective solution methods for POMDPs to derive the optimal "alpha vectors" in equalition (5) has remained an active area of research for quite a long time. Monahan

(1982) reviews some of the earlier models and solution methods for POMDPs. Smallwood and Sondik (1973) propose an algorithm for finite horizon POMDP policies. This algorithm generates the set of alpha vectors $\vartheta(t)$ at time t from $\vartheta(t-1)$ which is the set at time $t-1$. Linear inequalities are identified to represent the belief space and the optimal solution. They also propose a linear programming algorithm to identify tighter constraints for the belief space, to make the overall algorithm computationally efficient. White (1980) extends this algorithm with the help of known structural properties of certain classes of POMDPs, resulting in a more efficient solution method. Sondik (1978) develops a policy iteration algorithm for infinite horizon discounted cost POMDPs. The idea of *finitely-transient* policies are applied here, such that the associated infinite horizon discounted value function become piecewise linear. Then solutions can be derived using the same procedure as given in Smallwood and Sondik (1973).

Algorithms described above construct an approximate value function with the alpha vectors ($V_t^*(b) = \max_k \sum_s \alpha_t^k(s)b(s)$) and sequentially improve the value function by including new vectors to the set of alpha vectors ϑ_t. The central idea behind these methods is that any vector α_t generated from the set ϑ_{t-1} at a belief state b will be added to ϑ_t only if $V_t(b) = V_t^*(b)$. The algorithms' task is to find a belief-point b for which this relation does not hold, or to prove that such a belief-point does not exist. Cassandra, Kaelbling and Littman (1994) propose the Witness Algorithm for efficient execution of this task. This algorithm creates a linear program that returns a belief-point which is "witness" to the occurrence of $V_t(b) = V_t^*(b)$. This witness point is then used to determine a new vector to be added to ϑ_t

The linear programming based algorithms may prove to be computationally expensive and difficult to implement for large problems and infinite horizons. The practical circumstances, often the system never transitions to a large section of the belief space, hence the reachable belief space is comparatively much smaller. New methods such as Point Based Value Iteration (PBVI) have been developed considering only the reachable belief simplex. PBVI (Pineau, Gordon, Thrun, 2003) is an approximate method which selects a small set of belief-points and tracks the values and the derivative of these points. The process of exploring stochastic trajectories reduces the number of belief-points and generates good solution for large problems in reasonable time. "PERSEUS" is one such PBVI developed by Spaan and Vlassis (2005). It is a randomized point based value iteration algorithm which operates on a large set of belief-points sampled through the interactions between the agent and the environment. The algorithm generates and updates the set of non-optimal alpha vectors ("backup" stages) through iterations and ensures that at each backup stage, the value of each point in the belief set is improved or stays the same. The advantage of this algorithm is that a single backup operation may improve the values of many points, thereby speeding up the solution process for large problems. We have used PERSEUS in the next chapter to solve our POMDP formulation of the cyber vulnerability maintenance problem in the next chapter.

We derive the formulation of the backup stage for PERSEUS below. Let us suppose that the reward $r(s, a)$ only depends on the destination state $s' \in S$ and action $a \in A$. From stage t of the value iteration and a discount factor γ, the value of stage $t + 1$ is derived by (using (1), and (2)):

$$V_{t+1}(b) = \max_a [\sum_{s'} b(s') r(s', a) + \gamma \sum_o p(o|a, b) V_t(b')]$$

$$= \max_a [\sum_{s'} b(s') r(s', a) + \gamma \sum_o \max_{\{g^k_{a,o}\}} \sum_{s'} b(s') g^k_{a,o}(s')] \quad (6)$$

where $g^k_{a,o}(s) = \sum_{s'} p^a_{s'o} \, p^a_{ss'} \alpha^k_t(s')$.

The backup vector at belief-point b is given by

$$backup(b) = argmax_{g^b_a} \sum_s b(s) g^b_a(s) \text{ where} \quad (7)$$

$$g^b_a(s) = r(s, a) + \gamma \sum_o argmax_{g^k_{a,o}} \sum_{s'} b(s') g^k_{a,o}(s') \quad . \quad (8)$$

The steps of the PERSEUS algorithm are given below.

Generate Belief-points Generate a set of reachable belief-points $\boldsymbol{B}$ by random walk.
Initialize Set $\vartheta_{t+1} = \emptyset$, $\widehat{\boldsymbol{B}} = \boldsymbol{B}$.
Sample Belief-point b from $\widehat{\boldsymbol{B}}$ uniformly at random.
Backup Calculate $\alpha = backup(b)$.
Update If $\sum_s \alpha(s) b(s) > V_t(b)$
include α to ϑ_{t+1}, else include $\bar{\alpha} = argmax_{\alpha^k_t} \sum_s \alpha^k_t(s) b(s)$.
Filter Update $\widehat{\boldsymbol{B}}$ with unimproved points $\widehat{\boldsymbol{B}} = \{b \in \boldsymbol{B}, V_{t+1}(b) < V_t(b)\}$.
Check If $\widehat{\boldsymbol{B}}$ is empty, **STOP** else go to **Sample**

Algorithm: PERSEUS

The random walk in the first step can be considered as a sample of interaction between the agent and the environment. A small number of alpha vectors can improve $V_t(b)$, for all the belief-points $b \in \boldsymbol{B}$. PERSEUS randomly samples a single belief-point b from $\widehat{\boldsymbol{B}}$, generates an alpha vector $\alpha = backup(b)$ for this point and then checks adds the alpha vector to the set ϑ_{t+1} if $\sum_s \alpha(s) b(s) > V_t(b)$. The value of $V_{t+1}(b)$ is updated with the new alpha vector and usually, many other belief-points are improved with a single backup operation. The steps are repeated as long as all the points in $\widehat{\boldsymbol{B}}$ are not improved.

4.4 Bellman Error

Shani, Brafman, Shimoni (2006) discusses the idea of Bellman error, which is the improvement in value that can be gained from an update of belief-state from b. The Bellman error for an update to a new belief-state b' is given by

$$e(b) = \max_a [\sum_{s'} b(s')r(s', a) + \gamma \sum_o p(o|a, b)V(b')] - V(b) \quad (9)$$

where $V(b)$ is defined in equation (5). In the PVI algorithm proposed by Shani, Brafman, Shimoni (2006), the belief-states are updated in the order of decreasing Bellman error. We have used a similar idea to check the quality of solution generated by PERSEUS. We call this criteria $RMSE_{Bellman}$, which is given by the square root of the mean of the sum of square errors over all the belief-points in B. To derive the expression of $E_{Bellman}$, we first need to look at the Bellman optimality equation as given in Spaan, Vlassis (2006),

$$V^*(b) = \max_a [\sum_{s'} b(s')r(s', a) + \gamma \sum_o p(o|a, b)V^*(b')] \quad (10)$$

Since PERSEUS is an approximate method, this above equality may not always be reached in a reasonable span of time. Hence after every iteration i, with a set of α-vectors α_i, we generate two quantities for each belief-point b, the left-hand side (LHS) value and the right-hand side (RHS) value of the above equation for iteration i.

$$V^i_{LHS}(b) = \max_k \sum_s \alpha^k_i(s)b(s)$$

$$V^i_{RHS}(b) = \max_a [\sum_{s'} b(s')r(s', a) + \gamma \sum_o p(o|a, b) \max_k \sum_s \alpha^k_i(s)b'(s)]$$

Here and in chapters 5 and 6, we create a summary measure based on the individual Belman errors called root mean squared error of Bellman ($RMSE_{Bellman}$) given by

$$RMSE^{i}_{Bellman} = \sqrt{\sum_b \left(V^{i}_{LHS}(b) - V^{i}_{RHS}(b)\right)^2 / n_B} \tag{11}$$

(where n_B is the number of belief-point considered).

Ideally the optimal solution with the belief-set B should result in $RMSE_{Bellman} = 0$. As $i \to \infty$, $RMSE^{i}_{Bellman} \to 0$.

We can also come up with similar criteria in terms of actions. The action $a^{i}_{LHS}(b)$ from LHS can be derived from the action corresponding to the α-vector yielding maximum value for belief-point b. The $a^{i}_{RHS}(b)$ from RHS is given by the action yielding maximum value.

$$a^{i}_{LHS}(b) = a(argmax_{\alpha^{k}_{i}} \sum_s \alpha^{k}_{i}(s) b(s))$$

$$a^{i}_{RHS}(b) = \underset{a}{\text{argmax}}[\sum_{s'} b(s') r(s', a) + \gamma \sum_{o} p(o|a, b) \max_{k} \sum_{s} \alpha^{k}_{i}(s) b'(s)]$$

The sum of the difference between LHS and RHS actions over the belief-set in iteration i is given by

$$\Delta^{i}_{a} = \sum_b |a^{i}_{LHS}(b) - a^{i}_{RHS}(b)| \tag{12}$$

For an optimal solution, both $RMSE^{i}_{Bellman} = 0$ and $\Delta^{i}_{a} = 0$ (from 8). We cannot prove that equation (12) is a sufficient condition for optimality, since we have only used a sample of reachable belief-points to form the belief-set B in an algorithm like PERSEUS. However, $RMSE^{i}_{Bellman} \sim 0$ and $\Delta^{i}_{a} = 0$ are necessary conditions assuming that the system

has a stead state optimal policy. In effect, $\Delta_a^i = 0$ is more important, since the policy is determined by what action to take for a certain belief-point. In some cases, multiple actions can lead to same expected reward, if all the actions yield same expected reward. In chapter 5 and 6, although we have not used these two criteria directly in PERSEUS generate the α-vectors, we have used them in the stopping criteria to ensure quality of our solutions.

4.5 Bayes Adaptive Markov Decision Process

In both MDP and POMDP, it is assumed that the transition probabilities p_{ij}^a and the reward or observation probabilities r^a are known. However, in more general cases these parameters can be unknown as well. Then, there is an added expectation in the optimization formulation in equation (1) over these parameters. Duff (2002) considers this problem which he calls "*Optimal (Parametric) Learning"* where the agent learns about the possible environments through action.

The formulation of the problem could be "model-based" (as in equation (1)) or "model free". In the model-based approach, the decision maker starts with scenarios for transition probabilities, observation probabilities, and expected reward. In the model free approach, there is no specific distribution for these unknown quantities. The decision maker observes the reward of his action at the current time step, updates his beliefs about the operating environment and picks the next action with the goal of optimizing the expected reward over the planning horizon of the experiment. Duff (2002) uses both to demonstrate the applicability of Bayesian models in sequential decision processes. While

Duff (2002) , Poupart (2006) only consider cases in which the number of possible environments or scenarios is infinite, Hou (2014) proposes a formulation with finite number of scenarios. While finite scenario models are faster to solve, they will have the inherent problem of "spurious" learning because, with finite scenarios, transitions for one action from one state could effectively trigger learning about other states and/or actions which are linked by the scenarios. The infinite scenario models on the other hand assume one step at a time learning, which is much slower. Here we explore the idea of using Orthogonal Array based Latin Hypercube design (Tang 1993) to sample the scenarios, to derive models that are fast and free of spurious learning. We also explore the opportunities for learning more than a single transition-at-a-time afforded by finite scenario-based methods.

4.5.1 Model-based Methods

Model-based reinforcement learning is indirect, in the sense that the method starts with a certain model of the environment. The decision maker takes actions based on the model with the goal of maximizing the expected reward over the time horizon. The model parameters are first learned through interactions with the environment and observation of rewards (Ray and Tadepalli 2010). Once the model parameters are learned with some degree of confidence, general MDP solution algorithms like value iteration and policy iteration can be implemented to generate the optimal policy. In case the model parameters are unknown, the current model can be updated after observing the environment, and value or policy iteration could be run offline to solve the updated MDP.

The action suggested by the MDP may be then taken to interact with the environment and receive information for the next update step. This procedure is associated with significant computationally cost. Dyna (Sutton 1990) and Adaptive Real-time Dynamic Programming (ARTDP) (Barto, Bradtke and Singh 1995) are learning algorithms which use efficient update rules to lower computational burden. Dyna is an architecture in AI, which is an attempt to integrate learning, planning and taking actions. RTDP incorporates asynchronous DP for learning and execution, without the exhaustive nature of offline DP algorithms.

Policy gradient methods also fall under the domain of model-based learning. Wang and Dietterich (2003) build a number of incomplete models of MDP from training experiences to derive the policy gradient in closed form. Their algorithm switches between *pruning, exploration,* and *gradient ascent search* and requires fewer training examples than methods implementing Monte Carlo trials for gradient estimation. Abbeel, Quigel and Ng (2006) present a hybrid algorithm which uses an approximate model to solve problems with high dimensional continuous state space. This method learns the real system with only a handful of real-life trials based on a relatively "inaccurate" model. It first evaluates a policy on a real-life trial and them improves it using derivative estimates to reach near optimal solutions.

Choice of the correct MDP models is an issue with model-based so-called "reinforcement learning" (RL) as are all MDP formulations and variants because they relate to agents attempting to maximize their expected rewards. The trade-off between exploration and exploitation poses direct influence over the solution quality. Kearns and

Singh (2002) provide algorithms with polynomial bound in which the agent explicitly decides about exploiting the known part and exploring the uncertain part of the MDP (E^3). In E^3 the learning process continues up to a point at which it cannot be continued any more efficiently, then the agent implements the learned model to generate optimal policy. Brafman and Tennenholtz (2002) propose the R-MAX algorithm, in which the choice between exploration and exploitation remains implicit. In R-MAX, the agent keeps a possibly inaccurate model of its environment and takes action based on the optimal policy generated from this model. It is an improvement over the E^3 algorithm, in the sense that it is not biased towards exploration at earlier stages like the E^3 algorithm. The optimal policy derived from an inaccurate model in R-MAX has the important property of being either optimal or resulting in efficient learning.

4.5.2 Model-Free Methods

In model-free learning, the solution does not start with an estimate of the actual MDP. It is a more direct approach where the agent starts taking actions based on estimated values, without a MDP model. In one class of methods, the value of a state is estimated based on the immediate reward and the estimated discounted value of the next state. These methods fall under the category of temporal difference learning (Wiering, Otterlo 2012). These methods are also categorized as online learning methods, since the agent interacts with the environment first and then updates the estimates of the model parameters. One aspect of model-free learning is the need of balance between exploration and exploitation. In exploitation, the agent repeats the actions that have yielded the best

rewards in past. But for long term reward, the agent also needs to explore outcomes of the actions not tried as many times before.

Watkins' Q-learning algorithm is an example of model-free reinforcement learning (Watkins 1989). In Q-learning, $Q(s,a)$ represents the expected discounted reward of taking action a in state s. If $Q^*(s,a)$ is the optimal expected return, the Bellman equation with Q-value is given by,

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} p(s,a,s') \max_{a'} Q^*(s',a')$$

where $r(s,a)$ is the immediate reward in state s when action a is taken, $p(s,a,s')$ is the probability of transitioning from state s to state s' upon taking action a and γ is the discount factor. As shown in Kaelbling (1996), Q-learning estimates the Q values online following the rule,

$$Q^*(s,a) \coloneqq Q(s,a) + \alpha(r + \gamma \max_{a'} Q^*(s',a') - Q(s,a))$$

where r is the observed reward when system transitions from state s to state s' on taking action a.

One common exploration strategy is the ε-greedy policy, where the agent chooses its current best action with probability $1-\varepsilon$, and the exploratory step with probability ε. In Boltzmann exploration strategy, the action selection probabilities depend on their relative Q-values. The probability of picking action a can assume an expression as below:

$$P(a) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_i e^{\frac{Q(s,a_i)}{\tau}}}$$

In the above expression, τ is equivalent to the temperature parameter. With the increase of τ the randomness of action selection increases. Lower values of τ will tend to exploit the best action greedily. Duff (2002) proposes a model-free reinforcement learning algorithm to calculate the Gittins indices for multi-arm bandit problems.

4.5.3 Finite Scenario BAMDP

The methods described above have found extensive use in robot movement control and multi-arm bandit type problems. Here the decision maker (robot) takes an action, observes the consequences (rewards or outputs) and updates its belief state before taking the next action. However, this one-step at a time learning could be slow. In such circumstances, finite scenario formulations may prove to be useful (Duff, 2002). There are some studies with finite parameter or multi matrix cases where the assumption is that the MDP is generated from a possible set of known transition matrices and they can be reduced to POMDPs (Silver, 1963, Duff, 2002, Hou, 2015). The advantage of doing this is a more intuitive state space (states are combinations of usual "natural states" and scenarios being true) and potentially faster learning in that users might effectively eliminate all but one model scenarios in a small number of steps..

Delage and Mannor (2010) presents a sampling procedure for transition probabilities p_{ij}^{a} using Dirichlet priors from the corresponding frequencies of transitions f_{ij}^{a}. It is convenient to assume Dirichlet prior, since given a set of observed transitions from the multinomial distribution, the analytical solution to the posterior can be found using the Multinomial-Dirichlet conjugate relation. However, the number of scenarios

chosen will influence the solution quality. If the number of chosen scenarios q is small, there is a possibility of spurious learning. In cases where number of instances of transitioning from state i to state j is low compared to the other transition frequencies or zero, a model with small number of scenarios will almost neglect these instances altogether. The overall solution might be more optimistic in this case. In other words, this would be a case of over exploitation.

The problem of insufficiency in learning has received some attention in research. In the literature related to dynamic pricing, Rothschild (1974) proposes the use of Bayesian adaptive control theory to identify optimal sequences of offer prices. This approach, as recognized in Rothschild (1974), is associated with a positive probability of incomplete learning. In the long run, the learning mechanism may not always generate the most profitable price. This phenomenon, called *price dispersion* happens with positive probability even when all the players start with true beliefs initially. Xia and Dube (2007) addresses this problem in Bayesian adaptive control formulation and builds a framework using simulated annealing which avoids price dispersion. This framework shows convergence to the true optimal exploration policy to the true optimal policy with complete information.

In our case, the problem of spurious learning can be mitigated by considering a large number of scenarios. However, when the scenario matrices are consolidated to form a POMDP (as shown later in this section), the size of the problem may render it difficult to solve. To balance between effective learning and computation time, we here propose to integrate into BAMDP scenario selection with Orthogonal Array Latin Hypercube

(OALH) design. Additional details and examples are in Chapter 6. In this chapter, the focus is on reviewing the relevant sampling methods.

Ordinary Latin hypercube sampling (LHS) was first introduced by Mckay, Beckman and Conover (1979) and it is widely used in experimental design with continuous variables. In LHS, the probability distributions of each of the random variables involved are divided simultaneously in equal segments, and then one sample from each segment is drawn. Orthogonal Arrays (OA) produce uniform samples for multidimensional spaces. LHS is a special case of OA since it produces uniform sampling in one dimension. We have used the OALH design formulated by Tang (1993) in our study. These designs preserve the stratification properties of LHS, as well as the stratification over r-dimensional margins for strength r OA based Latin hypercubes. These designs show improved performance for computer experiments and possess better numerical integration properties than ordinary Latin Hypercubes. We use OALH to sample probability of state transitions for cases with higher uncertainty, i.e. transitions that are rare or the ones of which we do not have any data.

4.5.4 Formulation of BAMDP Model:

Let us keep the same notations as used in the previous sections, and assume there are N states in which the system can reside, and the decision maker has U actions to choose from. But since there is uncertainty in the transition probabilities, let us assume there are q scenarios, and that the N original states are natural states. Let the probability of being in one of these scenarios be $P(k),\ k \in \{1,..,q\}$. As shown in Delage and

Mannor (2010), given a set of observations f_{ij}^{a} from natural states i to j where $(i,j) \in \{1,..,N\}$, upon taking action $a \in \{1,..,U\}$, the transition probabilities can be sampled from a Dirichlet distribution:

$$p_{ij}^{a}(k) \sim Dirichlet\left(f_{ij}^{a} + \beta(k)\right), k \in \{1,..,q\}, i,j \in \{1,..,N\} \quad (12)$$

where $p_{ij}^{a}(k)$ is the transition probability for scenario k and $\beta(k)$ is the Dirichlet prior for scenario k.

From above, the stochastic process can be formulated as follows. Let S_t be the natural state at time t, which is the result of transitioning from natural state S_{t-1} at time epoch $t-1$ upon taking action a_{t-1}. For a certain scenario $k \in \{1,..q\}$, the stochastic process of the MDP follows a multinomial distribution (Hou, 2015).

$$S_t | S_{t-1}, a_{t-1}, \boldsymbol{p}(k) \sim Multinomial\left((p(k)^{a_{t-1}})_{S_t,:}\right) \quad (13)$$

where "$S_{t-1},:$" refers to the entire row of state S_{t-1} of the relevant transition matrix. This means destination state of the transition from state S_{t-1} upon taking action a_{t-1} is multinomially distributed with parameters given by the probability values in the row corresponding to state S_{t-1} in the probability transition matrix of action a_{t-1}.

Hou (2015) shows that the above problem can be converted to a POMDP. Considering the q scenarios, we will have a total of $N \times q$ states, which we call the pure states. The transition probability matrices and the rewards can be formed as below:

$$\widehat{\boldsymbol{p}}^{\boldsymbol{a}} = \begin{bmatrix} \boldsymbol{p^a(1)} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{p^a(2)} & \dots & \mathbf{0} \\ & & \dots & \\ & & \dots & \\ \mathbf{0} & \mathbf{0} & & \boldsymbol{p^a(q)} \end{bmatrix}$$

$$\widehat{\boldsymbol{r}}_{\boldsymbol{o}}^{\boldsymbol{a}} = \begin{bmatrix} \boldsymbol{r_o^a(1)} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{r_o^a(2)} & \dots & \mathbf{0} \\ & & \dots & \\ & & \dots & \\ \mathbf{0} & \mathbf{0} & & \boldsymbol{r_o^a(q)} \end{bmatrix}$$

where $\boldsymbol{p^a}(k)$ is the transition probability matrix in scenario k upon taking action a, and $\boldsymbol{r_o^a}(k)$ is the reward matrix upon taking action a and observing o in scenario k. The belief-points b for being in natural state X_t at epoch t is given by:

$$\hat{b}(s,t) = \begin{cases} P(k) & \text{if } s = (k-1)N + X_t \\ 0 & \end{cases}$$

where $P(k)$ is the probability of scenario k and s is the pure state of the system.

Once we formulate the BAMDP as a POMDP we can use any of the solution procedures described in section 4.3. In the next chapter, we will use PERSEUS to solve the BAMDP models for cyber vulnerability maintenance problems.

4.6 Summary

This chapter is a review on MDP based methods that are used to make decisions in a stochastic environment. We started with the basic MDP and discussed the value and policy iteration algorithm. Then we discussed the purview of POMDP, in which there is uncertainty about the system state. We reviewed different POMDP solution techniques,

in particular, PERSEUS, which we will use in our cyber vulnerability maintenance application in the next chapters. Next, we explored the domain of BAMDP and reinforcement learning, and methods which also take into account parametric uncertainty inherent in the problem. In the next two chapters, we will show how POMDP and BAMDP formulations can be used to model cyber maintenance problems.

Chapter 5: Cyber Vulnerability Maintenance using POMDP

5.1 Introduction

With new technology, come new responsibilities of managing those innovations to prevent them from being utilized to harm people and organizations. It is no less true in case of present day dependencies on computers and internet. Starting from minor attacks on personal computers to thwart an individual's day to day work, to stealing of sensitive information concerning national security, cyber attackers' spectrum of influence is as wide as the means to commit such crimes.

Since life in digital age is enmeshed with the cyber world in virtually every possible way, be it an individual's health and financial records, or sensitive information about future strategies of an organization, operations of power grids, banking systems, military establishment, we have exposed ourselves to this new form of crime which was not even a concern even 20 years ago. Several incidents of data breaches to reveal personal information of account holders with various internet based service providers, theft of large sum of money from banks, compromise of information of Navy sailors have been reported in 2016 alone (ComputerWeekly.com, 2016). Reports on major

cyberattacks that crippled electrical utilities in Ukraine in December 2016 raised concern among policy makers all over the world (Smith 2016). The recent discussions on how cyber-crime has possibly influenced the 2016 U.S. presidential election highlight severity of the issue even further. It has become imperative to create well-defined mechanisms to prevent these attacks from causing large scale disruptions impacting daily life and economy.

The U.S. Department of Defense has recognized cybersecurity as a high priority (Kott, 2014). The importance of treating cybersecurity as a "science" and develop comprehensive R&D effort has been emphasized by the US President's National Science and Technology Council (2011). This has engendered increase in funding formal research and policy development in the area of cybersecurity.

Cybersecurity is emerging as a prospective application area in the field of Operations Research. Since this new front of vulnerabilities comes with its own overhead, efficient deployment of financial and labor resources would ensure higher sustainability for any organization. Interestingly, an estimate shows that more than 90% of the reported attacks are preventable with available patches and control schemes (Cockburn 2009). Efficient classification and identification of the major sources of vulnerabilities can lead to development of effective strategies to significantly reduce the extent of certain attacks. Gil et al. (2014) explored threat logs from a university and found causal relations between network services used by the hosts and susceptibility to certain attacks using a genetic epidemiology approach. Network security for smart electrical grids is another important area of research. For instance, Chen, Sanchez-

Aarnoutse and Bufford (2011) have proposed a Petri net based method for modeling cyber physical attacks on smart grids. Srivastava, Morris and Ernster (2013) have presented graph theory based methods for smart grid related cyber physical systems.

Application of MDP based methodologies has gained some importance in the world of cyber security over the past few years. Holloway (2009) presents a network security model with self-organized agent swarms (SOMAS) which is based on POMDP. Afful-Dadzie, Allen (2014) developed a Markov Decision Process based approach for generation and graphical evaluation of cybersecurity policies. They proposes the "Sufficiency Model Action Clarification" method and implements it to develop optimal cyber vulnerability maintenance policies. This method also attempts to address the issue of data insufficiency and parametric uncertainty. With the ever-evolving dynamics of cyber-crime, defense mechanisms with inherent and autonomous learning capabilities are becoming more relevant. The research presented here is an attempt to develop such adaptive control policies in cybersecurity.

The aim of this chapter is to develop control policies under uncertainty for cybersecurity. We introduce MDP based methods to explore policies in cybersecurity. We start with a POMDP model where the uncertainty of the system state derives from the limited knowledge of whether the system is compromised or not.

5.2 Methods

We propose a partially observable Markov Decision Process for a cybersecurity case study to generate control policies when there is uncertainty in the state of the system.

Hou (2015) states, based on the CVSS score an individual host can be in one of 5 states of vulnerability, viz, low, medium low, medium high, high and critical. In our analysis, we will consider only two vulnerability states, low and high. We will account for the uncertainty about the state of compromise of a host. If a host is compromised, then it is assumed that an attacker has gain control of it. A compromised host can be exploited in many ways, for instance it can provide access to sensitive information to the attacker, it can be used to compromise other hosts, or if it is a part of a control network, it can be used to disrupt entire control systems such as power grids. Not all the hosts in an organization contain sensitive data or has access to other critical systems. But they can still be used to infect other computers, or launch attacks which require multiple hosts e.g. DDoS attacks.

In most cases, it is not difficult to determine the state of vulnerability of a host, the CVSS scores can be used to get a relative sense. Hou (2015) considers that a single host can have more than one vulnerabilities and the host vulnerability is equivalent to the status of its worst vulnerability. However, it may not be always obvious if a host is breached or not. Especially when a host runs on an out of date OS, or does not get updated or scanned regularly, it can potentially be compromised and not detected. Microsoft Advanced Threat Analytics estimates that median number of days an attacker can stay undetected in a network is 146. In view of this, we have created six different states $S_i, i = \{1,2 \dots 6\}$ for a host, high and low states in convolution with three compromise states, viz. not compromised, compromised known and compromised unknown. There are three general actions $a_i, i = \{1,2,3\}$ to control the system, a_1 is to

leave the host for "automatic patching", which means a human administrator will not intervene. This action is associated with the lowest cost, but has limited ability to detect compromise or reduce vulnerability of a system. The second action a_2 is "manual intervention" where an administrator does "take action" based on the state of a host. This can range from a few hours of work to restore a system back to an acceptable state, to a complete replacement of a host. Hence a_2 is more expensive than a_1 in terms of personnel time and/or equipment cost. The third action a_3 is *inspect and act,* where an administrator first runs forensic inspection to check if the host is compromised then takes action. We assume that a_3 will have the same effects as that of a_2 on the transition probabilities, in addition, it will reveal the state of compromise to the administrator. Although the cost of inspection makes a_3 more expensive than the other two, action a_3 reduces the probability of transitioning to "compromised unknown" states to zero. The list of actions is given below in a concise fashion:

a_1: Automatic patching, no manual intervention.

a_2: Manual intervention

a_3: Manual intervention with forensic inspection.

To build the state transition matrix, we consider a number of parameters which may influence the transition probabilities. The parameters themselves are probabilities and their values depend on the action chosen. The list of parameters is given below all of them are probabilities of the events described alongside for actions a_1 and a_2. :

$p_{vw}^{a_i}$: Vulnerability will get worse.

$p_{vw}^{a_i\,\prime} = 1 - p_{vw}^{a_i}$.

$p_{vb}^{a_i}$: Vulnerability will get better.

$p_{vb}^{a_i\prime} = 1 - p_{vb}^{a_i}$.

$p_{lck}^{a_i}$: Low vulnerability and it is known that the host is compromised.

$p_{lcu}^{a_i}$: Low vulnerability and it is unknown that the host is compromised.

$p_{lcn}^{a_i} = 1 - p_{lck}^{a_i} - p_{lcu}^{a_i}$: Low vulnerability and host is not compromised.

$p_{hck}^{a_i}$: High vulnerability and it is known that the host is compromised.

$p_{hcu}^{a_i}$: High vulnerability and it is unknown that the host is compromised.

$p_{hcn}^{a_i} = 1 - p_{hck}^{a_i} - p_{hcu}^{a_i}$: High vulnerability and host is not compromised.

$p_{fk}^{a_i}$: Known comprise will be fixed.

$p_{fk}^{a_i\prime} = 1 - p_{fk}^{a_i}$.

$p_{fu}^{a_i}$: Unknown comprise will be fixed.

$p_{fu}^{a_i\prime} = 1 - p_{fu}^{a_i}$.

Table 5.1 Transition probabilities for actions 1 and 2

		Not compromised		Compromised Known		Compromised Unknown	
		Low	High	Low	High	Low	High
Not compromised	Low	$p_{vw}^{a_i\prime} p_{lcn}^{a_i}$	$p_{vw}^{a_i} p_{lcn}^{a_i}$	$p_{vw}^{a_i\prime} p_{lck}^{a_i}$	$p_{vw}^{a_i} p_{lck}^{a_i}$	$p_{vw}^{a_i\prime} p_{lcu}^{a_i}$	$p_{vw}^{a_i} p_{lcu}^{a_i}$
	High	$p_{vb}^{a_i} p_{hcn}^{a_i}$	$p_{vb}^{a_i\prime} p_{hcn}^{a_i}$	$p_{vb}^{a_i} p_{hck}^{a_i}$	$p_{vb}^{a_i\prime} p_{hck}^{a_i}$	$p_{vb}^{a_i} p_{hcu}^{a_i}$	$p_{vb}^{a_i\prime} p_{hcu}^{a_i}$
Compromised Known	Low	$p_{vw}^{a_i\prime} p_{fk}^{a_i}$	$p_{vw}^{a_i} p_{fk}^{a_i}$	$p_{vw}^{a_i\prime} p_{fk}^{a_i\prime}$	$p_{vw}^{a_i} p_{fk}^{a_i\prime}$	0	0
	High	$p_{vb}^{a_i} p_{fk}^{a_i}$	$p_{vb}^{a_i\prime} p_{fk}^{a_i}$	$p_{vb}^{a_i} p_{fk}^{a_i\prime}$	$p_{vb}^{a_i\prime} p_{fk}^{a_i\prime}$	0	0
Compromised Unknown	Low	$p_{vw}^{a_i\prime} p_{fu}^{a_i}$	$p_{vw}^{a_i} p_{fu}^{a_i}$	0	0	$p_{vw}^{a_i\prime} p_{fu}^{a_i\prime}$	$p_{vw}^{a_i} p_{fu}^{a_i\prime}$
	High	$p_{vb}^{a_i} p_{fu}^{a_i}$	$p_{vb}^{a_i\prime} p_{fu}^{a_i}$	0	0	$p_{vb}^{a_i} p_{fu}^{a_i\prime}$	$p_{vb}^{a_i\prime} p_{fu}^{a_i\prime}$

Table 5.2 Transition probabilities for action 3 (note all parameters are for 2)

		Not compromised		Compromised Known		Compromised Unknown	
		Low	High	Low	High	Low	High
Not compromised	Low	$p_{vw}^{a_2\prime}p_{lcn}^{a_2}$	$p_{vw}^{a_2}p_{lcn}^{a_2}$	$p_{vw}^{a_2\prime}p_{lck}^{a_2}$	$p_{vw}^{a_2}p_{lck}^{a_2}$	$p_{vw}^{a_2\prime}p_{lcu}^{a_2}$	$p_{vw}^{a_2}p_{lcu}^{a_2}$
	High	$p_{vb}^{a_2}p_{hcn}^{a_2}$	$p_{vb}^{a_2\prime}p_{hcn}^{a_2}$	$p_{vb}^{a_2}p_{hck}^{a_2}$	$p_{vb}^{a_2\prime}p_{hck}^{a_2}$	$p_{vb}^{a_2}p_{hcu}^{a_2}$	$p_{vb}^{a_2\prime}p_{hcu}^{a_2}$
Compromised Known	Low	$p_{vw}^{a_2\prime}p_{fk}^{a_2}$	$p_{vw}^{a_2}p_{fk}^{a_2}$	$p_{vw}^{a_2\prime}p_{fk}^{a_2\prime}$	$p_{vw}^{a_2}p_{fk}^{a_2\prime}$	0	0
	High	$p_{vb}^{a_2}p_{fk}^{a_2}$	$p_{vb}^{a_2\prime}p_{fk}^{a_2}$	$p_{vb}^{a_2}p_{fk}^{a_2\prime}$	$p_{vb}^{a_2\prime}p_{fk}^{a_2\prime}$	0	0
Compromised Unknown	Low	$p_{vw}^{a_2\prime}p_{fk}^{a_2}$	$p_{vw}^{a_2}p_{fk}^{a_2}$	$p_{vw}^{a_2\prime}p_{fk}^{a_2\prime}$	$p_{vw}^{a_2}p_{fk}^{a_2\prime}$	0	0
	High	$p_{vb}^{a_2}p_{fk}^{a_2}$	$p_{vb}^{a_2\prime}p_{fk}^{a_2}$	$p_{vb}^{a_2}p_{fk}^{a_2\prime}$	$p_{vb}^{a_2\prime}p_{fk}^{a_2\prime}$	0	0

Table 5.1-2 shows the transition probabilities derived from the given parameters. The expressions for a_3 are derived with the parameters for a_2, since it is assumed that the effects of both are same for all three actions for the compromised states of not compromised and compromised known. For the compromised unknown states, we consider that action a_3 will reveal the state of compromise at the inspection stage. Hence we calculate the transition from the compromised unknown states to not compromised states taking into account that the probability of a known compromise will be fixed ($p_{fk}^{a_2}$). It is also worth noting that for actions $a_1$and a_2, we are assuming there is zero probability of transitioning from the compromised known states to compromised unknown states to vice-versa. On the other hand, for a_3, the inspection step reveals the state of compromise, hence there is zero probability of transitioning into the compromised unknown states once this action is taken.

We consider four possible observations, low and high vulnerability scores with no information on the state of compromise. Table 5.3 shows the probabilities of the observations given the states.

Table 5.3 Observation probabilities

		No Information		Compromised Known	
		Low	High	Low	High
Not compromised	Low	1	0	0	0
	High	0	1	0	0
Compromised Known	Low	0	0	1	0
	High	0	0	0	1
Compromised Unknown	Low	1	0	0	0
	High	0	1	0	0

The cost is broken into four different components. We assume that the cost of the first "breach" of a not compromised host is much higher than the cost of continued compromised state. The estimated costs are given in table 5.4. The costs incurred from state transitions upon taking different actions are given in tables 5.5-7.

Table 5.4 Cost of actions and compromise situations

Situation	Cost ($)
Action average cost	150
First compromised average cost	10000
Continue compromised average	2000
Inspection Cost	150

Table 5.5 Transition cost for action 1

		Not compromised		Compromised Known		Compromised Unknown	
		Low	High	Low	High	Low	High
Not compromised	Low	0	0	-10000	-10000	-10000	-10000
	High	0	0	-10000	-10000	-10000	-10000
Compromised Known	Low	0	0	-2000	-2000	-2000	-2000
	High	0	0	-2000	-2000	-2000	-2000
Compromised Unknown	Low	0	0	-2000	-2000	-2000	-2000
	High	0	0	-2000	-2000	-2000	-2000

Table 5.6 Transition cost for action 2

		Not compromised		Compromised Known		Compromised Unknown	
		Low	High	Low	High	Low	High
Not compromised	Low	-150	-150	-10150	-10150	-10150	-10150
	High	-150	-150	-10150	-10150	-10150	-10150
Compromised Known	Low	-150	-150	-2150	-2150	-2150	-2150
	High	-150	-150	-2150	-2150	-2150	-2150
Compromised Unknown	Low	-150	-150	-2150	-2150	-2150	-2150
	High	-150	-150	-2150	-2150	-2150	-2150

Table 5.7 Transition cost for action 3

		Not compromised		Compromised Known		Compromised Unknown	
		Low	High	Low	High	Low	High
Not compromised	Low	-300	-300	-10300	-10300	-10300	-10300
	High	-300	-300	-10300	-10300	-10300	-10300
Compromised Known	Low	-300	-300	-2300	-2300	-2300	-2300
	High	-300	-300	-2300	-2300	-2300	-2300
Compromised Unknown	Low	-300	-300	-2300	-2300	-2300	-2300
	High	-300	-300	-2300	-2300	-2300	-2300

We make assumptions for the values of the parameters used to make the transition probabilities. The values are given in table 5.8.

Table 5.8 Parameter values

Parameters	Values	Parameters	Values
$p_{vw}^{a_1}$	0.100	$p_{vw}^{a_2}$	0.010
$p_{vb}^{a_1}$	0.100	$p_{vb}^{a_2}$	0.800
$p_{lcn}^{a_1}$	0.998	$p_{lcn}^{a_2}$	0.990
$p_{lck}^{a_1}$	0.001	$p_{lck}^{a_2}$	0.005
$p_{lcu}^{a_1}$	0.001	$p_{lcu}^{a_2}$	0.005
$p_{hcn}^{a_1}$	0.994	$p_{hcn}^{a_2}$	0.985
$p_{hck}^{a_1}$	0.005	$p_{hck}^{a_2}$	0.010
$p_{hcu}^{a_1}$	0.001	$p_{hcu}^{a_2}$	0.005
$p_{fk}^{a_1}$	0.100	$p_{fk}^{a_2}$	0.100
$p_{fu}^{a_1}$	0.100	$p_{fu}^{a_2}$	0.900

5.3 Results

Using all the data listed above, we run the POMDP using the PERSEUS algorithm. Since there is partial observability in the states, so a policy $\pi(b|a)$ in this case would be an optimal action a, given a belief state b. We sampled 1,000 belief states through a random walk, starting from a non-informative belief state (probability of each state $\frac{1}{N}$ where N is the total number of states). Each belief states were iterated ten times to account for the randomness embedded in the value iteration of PERSEUS. Table 5.9 provides the average and standard deviations of the optimal values at ten typical belief

states encountered in the random walk. The optimal action at each belief state is the same for all iterations.

Table 5.9 Typical Policies from analysis of cyber security by POMDP with values in $.

	Not compromised		Compromised Known		Compromised Unknown		Percentage of Similar Belief States	Action	Value ($/host)
	Low	High	Low	High	Low	High			
1	1.000	0.000	0.000	0.000	0.000	0.000	1.500	a_1	-573.16
2	0.000	0.000	1.000	0.000	0.000	0.000	0.900	a_2	-913.09
3	0.000	0.000	0.000	1.000	0.000	0.000	1.100	a_2	-935.22
4	0.909	0.000	0.000	0.000	0.091	0.000	0.400	a_3	-963.82
5	0.840	0.000	0.000	0.000	0.160	0.000	0.300	a_3	-970.32
6	0.785	0.000	0.000	0.000	0.215	0.000	0.200	a_3	-975.25
7	0.525	0.000	0.000	0.000	0.475	0.000	93.100	a_3	-999.22
8	0.167	0.167	0.167	0.167	0.167	0.167	0.100	a_3	-1036.90
9	0.048	0.000	0.000	0.000	0.952	0.000	1.300	a_3	-1043.22
10	0.000	0.515	0.000	0.000	0.000	0.485	1.100	a_3	-1049.850

Table 5.9 shows a few typical belief states encountered in the random walk 1,000 belief states. The column "Percentage of Similar Belief States" shows the percentage of times a similar belief state (all probability values within ∓ 0.05) appeared in the 1,000 belief states sampled for the POMDP. The optimal policies indicate that if the system is equally likely to be in any of the six states, which is also the starting belief state for this example, action a_3 (inspect and act) is optimal (row 8). It is also evident, that even a small probability of being in the compromised unknown states will prompt action a_3 (row 4, 5, 6).

The most frequently visited belief state generated by the random walk has around 0.5 probability of being in each of not compromised low vulnerability and compromised unknown low vulnerability states, which also demands action a_3 (row 7). When there is certainty that the system is compromised, action a_2 is optimal (row 2, 3). This intuitively makes sense as in fact, if the administrator is certain that a host is compromised, she takes the action to restore the system, without the need to incur inspection cost. On the other hand, if there is more uncertainty about the state of compromise of the host, it is more economic to choose the "manual intervention and forensic inspection" alternative. When it is certain that the host is in the not compromised low vulnerability state, it is best to choose action a_1, which is automatic patching without any manual intervention (row 1). This belief state-action pair also incurs the least cost.

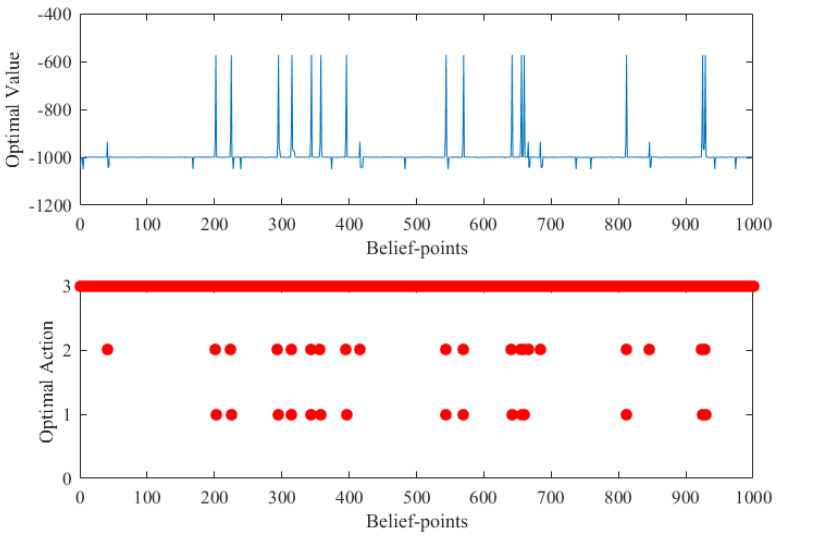


Figure 5.1Values and actions for different belief-states generated by POMDP

Table 5.6 shows that majority of the time, there in some uncertainty about the state of compromise. In such circumstances, it is best to choose action a_3. Figure 5.1 shows once in a while the system switches to belief states with known state of compromise and in these situations, actions a_1 or a_2 can be taken depending whether there the host is compromised or not. Overall, the POMDP suggests a conservative policy in cybersecurity, which can be summarized as "if there is even a little possibility that a host is compromised, it is best to do inspection and take action accordingly."

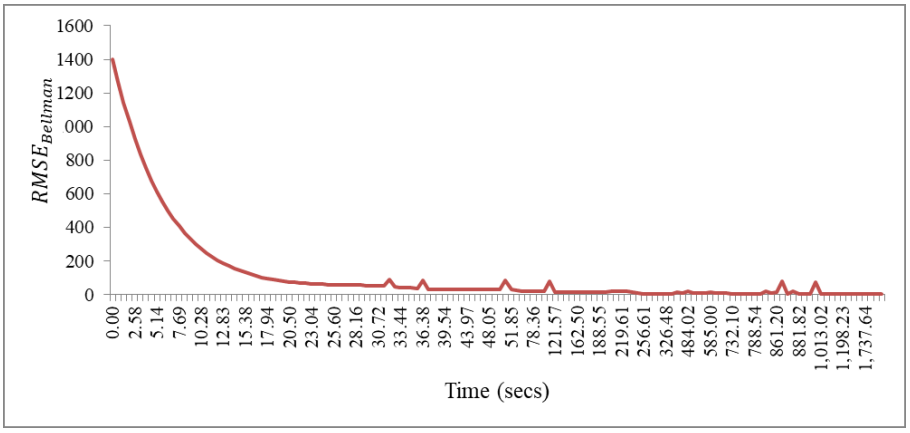


Figure 5.2 RMSE (Bellman) over iterations showing convergence of PERSEUS

In section 4.4 we have seen the expressions for Bellman Error, and have proposed the idea of using root mean square of the Bellman Error ($RMSE_{Bellman}$) over all the belief-points as an indicator for convergence of PERSEUS. Figure 5.2 shows how $RMSE_{Bellman}$ asymptotically converges as PERSEUS runs through completion.

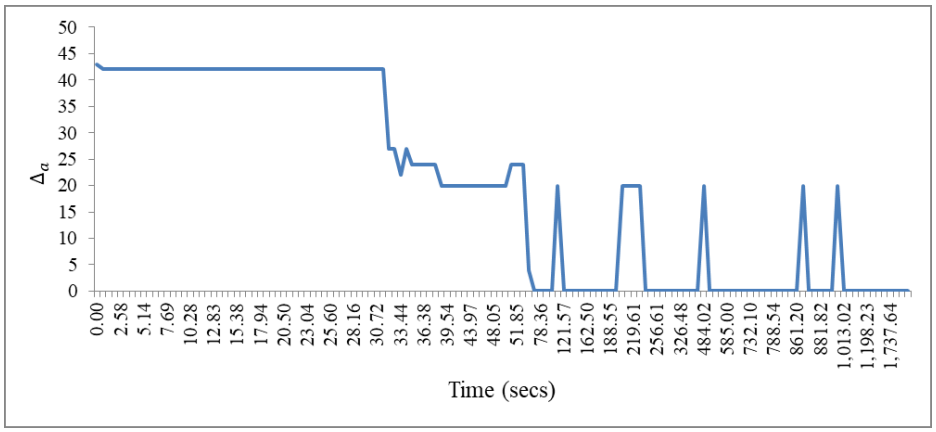


Figure 5.3 Sum of difference of actions associated with Bellman Error over iterations

Figure 5.3 shows the sum of difference of actions Δ_a of the two sides of Bellman equation (section 4.4, equation 12) plotted with the iterations of PERSEUS. The uneven characteristics are caused by the random sampling of belief-points for the backup step of PERSEUS. The optimal solution has $\Delta_a^i = 0$.

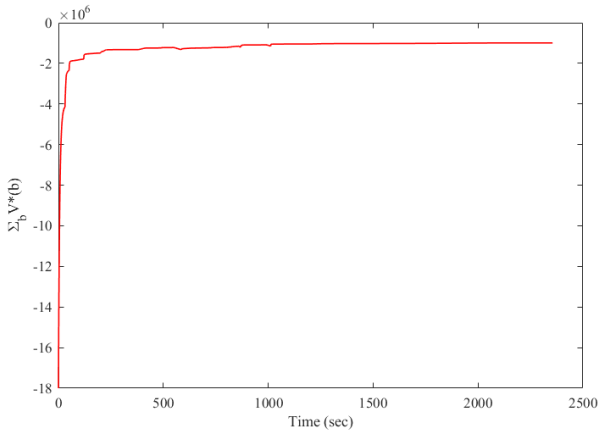


Figure 5.4 Sum of Optimal Values of all belief-points over iterations

Spaan and Vlassis (2006) use the sum of optimal values over all the belief-points as a convergence criterion for PERSEUS. Figure 5.4 shows a plot of sum of all the optimal values with the iterations. It shows the sum initially increases and then converges to a fixed value after a certain period of time.

5.4 Sensitivity Analysis

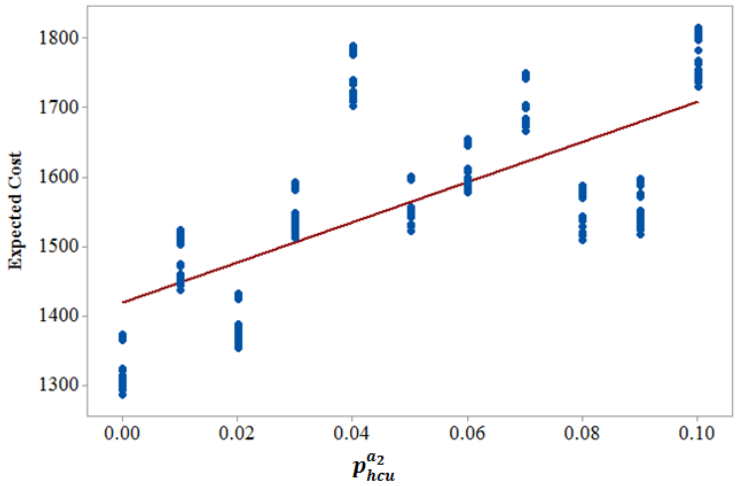


Figure 5.5 Expected cost for belief-states with positive probability of unknown state of compromise against probability of high vulnerability unknown compromise with action

To understand the effect of acting when the system has high vulnerability and the state of compromised is unknown we have run 11 different models varying the parameter $p_{hcu}^{a_2}$ from 0 to 0.1. Figure 5.2 shows how the expected cost (Value) changes with $p_{hcu}^{a_2}$ for the belief-points with positive probability of unknown state of compromise. We filtered the set of belief-points in this manner in order to highlight the cases with substantial probability of unknown state of compromise. The optimal action for all these cases is a_3 (inspect and act). Figure 5.2 reveals that although there is some variation for

each of the values considered for $p_{hcu}^{a_2}$, there is a clear trend that the cost (negative reward) increase with the increase of $p_{hcu}^{a_2}$.

5.5 Summary

In this chapter, we have introduced a POMDP model for cyber vulnerability maintenance analysis. The belief states for POMDP reflect the uncertainty due to the state of compromise of the host. We have built the transition probabilities from simpler factors such as probability if the vulnerability of a host is going to become worse upon taking certain actions, or better. We have solved the POMDP problem using PERSEUS and policies are derived for a number of belief-states. Typically, for belief-states with positive probability for only not compromised low vulnerability states, automatic patching is optimal. However, if there is even a small probability for unknown state of compromise, manual intervention with forensic inspection is the optimal action. For cases with positive probability only for known compromised states, a_2 for manual intervention is optimal. We have also shown convergence of PERSEUS in terms of $RMSE_{Bellman}$ and sum of optimal values over all the belief-points considered.

Next we have run a sensitivity analysis to determine the effect of changing one of the parameters over the solutions. We have found that increasing the probability of "being in a high vulnerability state in an unknown state of compromise upon taking action" does not necessarily change the nature of the policies derived before, but it increases the overall cost.

We believe that this POMDP based cyber vulnerability maintenance policies can help network administrators to manage the networks more efficiently. The parameters that we have assumed in the model can give us some insights on what type of data to log, so that we don't have to assume all the values and can use actual data to formulate similar models. In the next Chapter, we introduce BAMDP models for cyber vulnerability maintenance analysis which account for the parametric uncertainty stemming from data insufficiency.

Chapter 6: BAMDP for Cyber Maintenance

6.1 BAMDP Multiple Scenarios

In the previous chapter, we have used a POMDP model for cybersecurity maintenance analysis. We assumed that the transition probabilities and the observation probabilities are known. However, in reality, these parameters could be uncertain, or may change over time. Transition data could be available in some cases (e.g., for automatic patching action, system transition from low vulnerability to low vulnerability state) but in some other cases (high to low vulnerability), there might be no available data. This prompts us to explore methods which account for parametric uncertainty. The BAMDP models described in the previous chapter recognizes the issue of parametric uncertainty. In this section, we will implement three finite scenario methods of BAMDP.

The first method will consider a small number of scenarios (viz. 3), and the second method will consider a large number of scenarios (viz. 100). The first example with small scenarios would show the effects of *spurious learning.* This method will not sample from the scenarios which are less likely, and in effect will have a more *optimistic outlook* overall. The second example with a large number of scenarios will mitigate the spurious

learning issue, however, it might be debatable "how many scenarios could be considered a large number of scenarios." To address this problem, we will explore a third method, where we will sample the uncertain parameter from an Orthogonal Array Latin Hypercube (OALH) design. The OALH design exploits the r-variate uniformity, where r is the strength of the OA.

6.2 Test Problem

The data we have used in our example relate to the real-world network vulnerability situation of an organization. Based on the CVSS scores, we have assumed two natural states of vulnerability, low and high. There are three actions that the administrator can take,

a_1: Automatic patching, no manual intervention.

a_2: Research accept (inspect and accept the risk and if patch is found, fix the issue manually)

a_3: Overhaul or complete system restore.

The transition data and the cost structure are given in tables 6.1-3. In this table, *normal risk* indicates low vulnerability state and *elevated risk* indicates high vulnerability state.

Table 6.1 Transition data for 1

Auto-patch	Normal Risk	Elevated Risk	Cost of Action ($)	Average Cost ($) of Being at High Vulnerability
Normal Risk	216861	3341	0	(0,0,0)
Elevated Risk	NA	NA	0	(1000,500,1500)

Table 6.2 Transition data for 2

Research Accept	Normal Risk	Elevated Risk	Cost of Action ($)	Average Cost ($) of Being at High Vulnerability
Normal Risk	NA	NA	50	(50,50,50)
Elevated Risk	1903	2442	50	(1050,550,1550)

Table 6.3 Transition data for 3

Overhaul	Normal Risk	Elevated Risk	Cost of Action ($)	Average Cost ($) of Being at High Vulnerability
Normal Risk	600	0	1,100	(1,100, 1,100, 1,100)
Elevated Risk	400	0	1,100	(2,100, 1,600, 2,600)

In tables 6.1-3, we have included a “Cost of Action” columns which shows our assumptions on the respective costs. The costs of actions are assumed to be known, since these are incurred by the deployment of personnel or buying new software/equipment. The automatic patching does not cost anything, whereas the “Research Accept” action incurs some minor costs. Compared to these, the cost of overhaul is much more expensive, but it also removes the high-risk states completely. The column “Cost of being at high vulnerability” states shows the random costs that are associated with high vulnerability states. We assume that the high vulnerability state poses high risk for hosts for being compromised. It is hard to estimate these costs, since not all vulnerabilities are

as critical and it is difficult to put a dollar value over the loss due to system breach. We have assumed 3 different average costs for being at high vulnerability states, 1000, 500 and 1500 units.

In tables 6.1-3, we see that no data is available for state of *elevated risk* with action a_1 and the state of $normal\ risk$ with action a_2. We have made reasonable assumptions of about these numbers assigning much lower frequencies. The augmented data is shown in tables 6.4-5

Table 6.4 Augmented transition data for 1

Auto-patch	Normal Risk	Elevated Risk
Normal Risk	216,861	3,341
Elevated Risk	0	10

Table 6.5 Augmented transition data for 2

Research Accept	Normal Risk	Elevated Risk
Normal Risk	10	1
Elevated Risk	1,903	2,442

We start with the case with three scenarios. We assume a Dirichlet prior for 0.25 for all the transitions, and the scenarios are generated using equation 6 in Chapter 4. Since we start with two natural states and three scenarios, we have six pure states altogether.

To account for the parametric uncertainty, we start with a 3 scenario problem. In this problem, we consider 3 parameters to be uncertain, viz. the transition probabilities

from high vulnerability state of action automatic patching and from low vulnerability for action research accept. The third uncertain parameter is the cost of being at high vulnerability states. We generate 3 different probability transition models by sampling from a Dirichlet distribution for these 3 parameters. The rest of the transition probabilities are given by the fraction of the number of transitions for a certain state transition with respect to the number of all transitions from the same source state.

6.2.1 Solution using Equivalent POMDP

We then create the equivalent POMDP combining these 3 models. We use PERSEUS to solve the POMDP and generate α-vectors in order to formulate policies for any given belief states. In order to cover a wide variety of belief-points, we used Latin Hypercube sampling to generate multiple initial belief-points and then used random walk from each of them to create the sample belief space. We then run the point-based value iteration of PERSEUS to generate the solutions. The steps are summarized as follows:

1. Create scenarios: Use Dirichlet sampling for uncertain parameters to create different scenarios.

2. Combine scenarios: Merge the scenarios to form an equivalent POMDP.

3. Create sample belief space: Sample multiple initial belief-points from a Latin Hypercube design and generate belief-points from each of them using random walk.

4. Run value-iteration: Run PERSEUS on the sampled belief space

5. Derive policy: Use the α-vectors generated from PERSEUS to create action strategy for any given belief-point

Steps for multiple scenario BAMDP

6.2.2 Reward Based Learning

In our BAMDP model for cyber vulnerability, we have created a provision for reward based learning following Hou (2015). We are assuming that the cost R_a of taking an action a is fixed, and the cost of transitioning to a high vulnerability state R_h is random. Hence for each scenario, the variable cost of transitioning to a high vulnerability state is sampled from a normal distribution with the average of the observed variable cost as mean and the standard deviation of the observed variable cost as standard deviation. The observation matrix $\boldsymbol{O^a}$ for each action a of the equivalent POMDP has dimension $N \times n_O N$. The observation probability for each state- action- scenario tuple (O_{sj}^a) is derived as the probability of being in a certain bin created by the intervals of the range of rewards. The details are given below:

n_O: number of observation levels

$N = 2$: number of natural states

U: number of actions

q: number of scenarios

n_{OC}: number of columns in the observation matrix

n_R: number of reward levels observed

R_a : Cost (negative reward) of action a (fixed) $a \in \{1, \dots, U\}$

R_h: Cost (negative reward) of being at high vulnerability state (variable)

$\bar{R}_h$: Average cost of high vulnerability

S_R: Standard deviation of observed reward

$R_{ms}^a \sim N(\bar{R}_h + R_a, S_R) \quad \forall m \in \{1, \dots, q\},\ s$: High Vuln$\}$: Cost of transitioning from (to?) state s (high) upon taking action a in scenario m

$R_{ms}^a = R^a, s$: Low Vuln : Cost of transitioning from (to?) state s (low) upon taking action a in scenario m

$R_{min} = \min\{R_{ms}^a\} + 3S_R$

$R_{max} = \max\{R_{ms}^a\} - 3S_R$

$Int = \frac{R_{max} - R_{min}}{n_O - 1}$

$L_{obs} = R_{min} + Int \times (obs - 1) \quad \forall\ obs \in \{1, .., n_O\}$

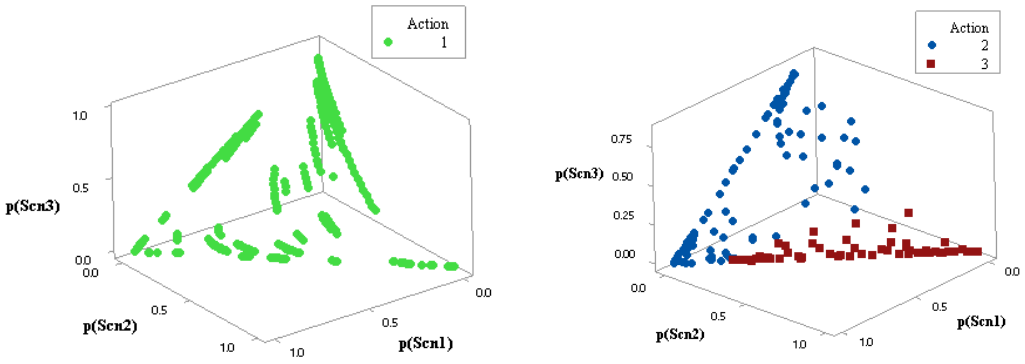
$U_{obs} = R_{max} + Int \times obs \quad \forall\ obs \in \{1, .., n_O\}$

The elements of the observation matrix $\boldsymbol{O^a}$ are given by

$O_{sj}^a = \Pr\{L_{obs} \le R_{ms}^a \le U_{obs}\}$ where $j = s + (obs - 1)N \quad \forall\ s \in \{1, .., N\}$

6.2.3 Results of 3 Scenario Problem using BAMDP

Figures 6.1 a and b show scatter plots of belief-points for low and high vulnerability states respectively.



(a) (b)

Figure 6.1 3D scatter plot for (a) low, (b) high vulnerability states

Figure 6.1(a) shows that when the system is in low vulnerability state, it is optimal to choose action 1 (automatic patching) irrespective of the scenarios. Figure 6.1 (b) shows that if the system is in high vulnerability state, actions 2 and 3 are chosen depending on the situation.

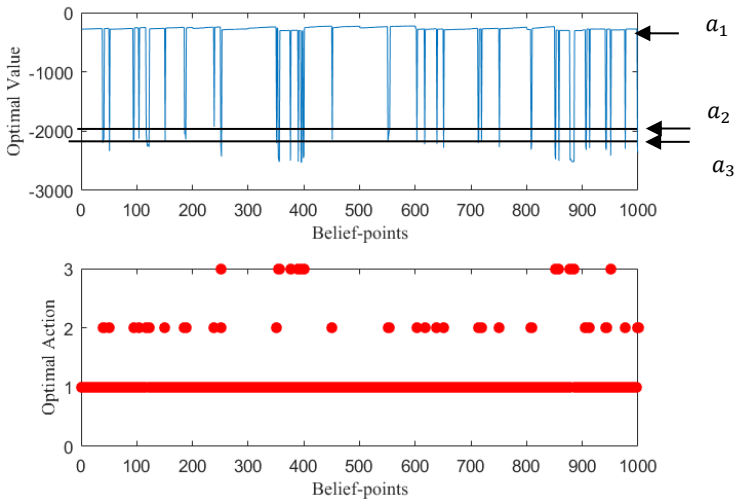


Figure 6.2 Value-action plot for 3 scenario

Figure 6.2 shows a plot of optimal values and actions for the belief-points used to solve the POMDP. Although this figure does not reveal much about the individual belief-points, it still shows that action 1 is chosen most of the time and is associated with lower cost or higher reward (value). Action 2 and action 3 are associated with higher cost, indiciated by the downward spikes in the top frame of the figure. The action names at the right side of the top frame show average values corresponding to different actions.

6.3 Simulation

To evaluate the effect of implementation of BAMDP, we created simulations to compare results from a random walk experiment and from optimal α-vectors. We started from a general uncertain belief-state, where each scenario is equally likely with a positive probability on the natural state of low vulnerability. For example, for the 3 scenario case, the initial belief-state for simulation is (0.33,0,0.33,0,0.33,0). At each step, an action is

chosen from the current belief-state and a new belief-state is generated by Monte Carlo (MC) sampling from the transition probability and observation matrices. The choice of action could be random or based on the optimal α-vector for the current belief-state. The steps of the simulation are given below.

---

1. Initial belief-state: Create initial belief-state where all scenarios are equally likely b_t

2. Initial pure state: Determine initial state s_t of the system by random sampling based on initial belief-state probabilities

3. Take action: Select action a_t randomly (for random walk) or from optimal α-vector for the current belief state

4. Next pure state: Sample next state s'_t from transition probabilities and a_t

5. Next observation: Simulate next observation o_t, based on observation probabilities and selected action

6. Reward: Observe reward value based on state transition, action and observation r_t

7. Bayesian update of belief-state: Update belief-state using equation (1) of Chapter 4 to b_{t+1}

8. Go to next step: Update $s_{t+1} = s'_t$, current belief-state with updated belief-state and go to step 2 until number of maximum steps reached

---

Steps for BAMDP simulation

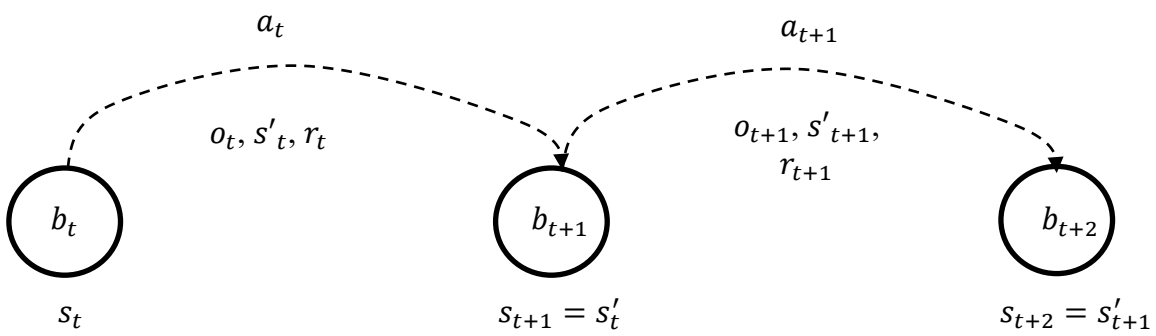


Figure 6.3 BAMDP simulation diagram

Figure 6.3 shows a schematic diagram of the steps in the BAMDP simulation. We have run the simulation for 10000 steps and replicated 10 times starting from the initial belief-state of (0.33, 0, 0.33, 0, 0.33, 0) for the 3 scenario problem. The histogram of the rewards generated by random walk is shown in figure 6.4 (a), and that by optimal α-vectors in figure 6.4 (b). From the histograms, it is clear that taking optimal action at each step will incur lower cost than if the actions are chosen randomly.

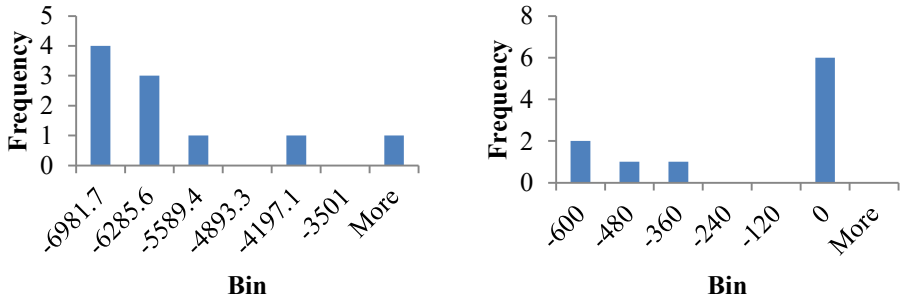


Figure 6.4 Histogram of expected reward from (a) Random walk, (b) Optimal alpha vectors

6.3.1 Average Learning Time

In the simulation runs, after a certain number of steps, the belief-states converge to a single scenario. That means the belief-states will have non-zero probabilities for the natural states belonging to only one scenario, and that scenario will remain true in the subsequent steps of the simulation. This indicates that the BAMDP leads to decrease the parametric uncertainty at every step with the Bayesian update of the belief-state and after a certain number of steps in the simulation, the parametric uncertainty is resolved and the true scenario is revealed. We come up with a measure called Average Learning Time (ALT) which provides the expected number of steps to converge to a single scenario from an initial belief-state where all the scenarios are equally likely. Let us suppose that ρ_{LL_t} is the parametric learning level at step t. For a system with 2 natural states and q scenarios, the belief-state will have $2q$ elements. Let p_{tj} be the probability of being in natural state 1 in scenario $j \in \{1, \dots, q\}$. Then, for a threshold value of ϵ

$$\rho_{LL_t} = \max_j \min[p_{tj}, 1 - p_{tj}] \qquad \forall\, j \in \{1, \dots, q\}$$

$$ALT = E\left(t \middle| \rho_{LL_t} > 1 - \epsilon\right)$$

Here ϵ should typically be around 0.01. We have calculated ALT as the average of the number of steps to reach $\rho_{LL_t} > 1 - \epsilon$ for 10 replications of the same simulations, each of which is run for 10000 steps.

Table 6.6 Learning time for BAMDP with 3 scenarios

Method	ALT	Std. Dev. of Learning Time
Random Walk	185	109.10
Optimal	1039	769.08

Table 6.6 provides the mean and the standard deviation of the learning times from the 10 replications of the BAMDP simulation for the 3 scenario case. It shows on an average, random walk (where actions are chosen randomly) takes fewer steps to reveal the true scenario than the optimal α-vectors. This can be explained by the trade of between exploration and exploitation in BAMDP. In random walk, since there is no objective to pick the action which will maximize the expected reward, the agent can learn fast by taking actions that are suboptimal. On the other hand, with the optimal α-vectors, there is only one action to be taken at each belief-state, which is the optimal action. Hence the agent does not learn about how the system would evolve if actions other than the optimal are taken for a certain belief-state. This contributes to the slow learning, but leads to higher expected return than random walk as shown in figures 6.4 (a) and (b).

6.4 More scenarios

Next we analyze the results generated by implementing a 100 scenario BAMDP on the same dataset. With 2 natural states and 100 scenarios, we have 200 pure states in the equivalent POMDP formulation. Since it is not possible to generate plots like figure

6.1 with 100 scenarios, we present in table 6.7 the distribution of the actions in the 1000 belief-points we used for PERSEUS to solve the equivalent POMDP.

Table 6.7 Optimal Value and Action by BAMDP for Belief-points evaluated in PERSEUS

Natural State	Optimal Value Avg. (Std.)	Percentage of Belief-points	Optimal Action
Low	-278.38 (59.93)	86.6%	a_1
High	-1952.83 (591.71)	10.1%	a_2
High	-2572.44 (96.83)	3.3%	a_3

Table 6.7 shows that out of 1000 belief-points, 86.6% have positive probabilities in low vulnerability natural states in all the scenarios. For such belief-points, a_1 or automatic patching is the optimal action. The rest of the belief-points have non-zero probabilities for the high vulnerability natural states. For these cases 10.1% of all the belief-states have a_2 or research accept as the optimal action, and 3.3% of all the belief-states as a_3 or complete overhaul as the optimal action associated with higher cost.

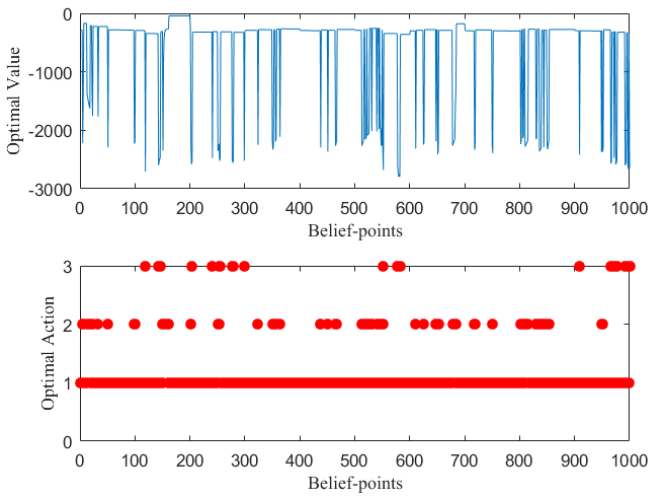


Figure 6.5 Optimal value and actions for 100 scenario BAMDP

Figure 6.5 shows the optimal expected rewards (values) and actions for the 1000 belief-points used in the PERSEUS.

Finally we present the BAMDP with Orthogonal Array Latin Hypercube Sampling (OALH) sampling. Latin Hypercube Sampling (McKay, Conover and Beckman 1979) is used for experimental designs and are known for stratifying each univariate margins simultaneously. OA based designs are also popular in industry, partly due to their uniformity properties. The OA based Latin Hypercubes as proposed by Tang (1993), preserves r-variate uniformity property of a strength r OA design. We have used strength r OALH designs to sample scenarios with r uncertain parameters.

We first construct an Orthogonal Array Latin Hypercube following Tang (1993). For the cases designated as "NA" in tables 6.1.a and b, (a_1: high → low, high → high; a_2 : low → low, low → high), and the negative reward associated with being in the state of high vulnerability, we generate samples from a Dirichlet distribution taking inverse over the values taken from the OALH design. The OALH design is of strength 3 to take into account the parametric uncertainty in these 3 quantities. Each scenario is sampled from each row of the OALH design. Since the design has 27 runs, we generate 27 scenarios, and as we have 2 natural states, our total number of pure states is 27 × 2 = 54. We solve the equivalent POMDP with 1000 belief-points generated from a random walk using PERSEUS.

Table 6.8 Optimal Value and Action for OALH BAMDP

Natural State	Optimal Value Avg. (Std)	Percentage of Belief-points	Optimal Action
Low	-289.34 (60.60)	89.5%	a_1
High	-2135.73 (120.40)	4.3%	a_2
High	-2532.26 (56.39)	6.2%	a_3

Table 6.8 gives an overview of the policies over the belief-points used in PERSEUS. Like the 100 scenario case, here also, most belief-points have positive probabilities in the low vulnerability natural states, with a_1 or automatic patching as the optimal action. The high vulnerability belief-states have optimal actions a_2 or research accept and a_3 or complete overhaul. Figure 6.6 shows the optimal values and actions for the belief-points used in PERSEUS. Like the 3 scenario and 100 scenario cases, it shows that the system stays in the low vulnerability state most of the time, and action a_1 would be optimal in these cases. Once in a while when the belief shifts towards high vulnerability requiring to take more expensive actions a_2 and a_3.

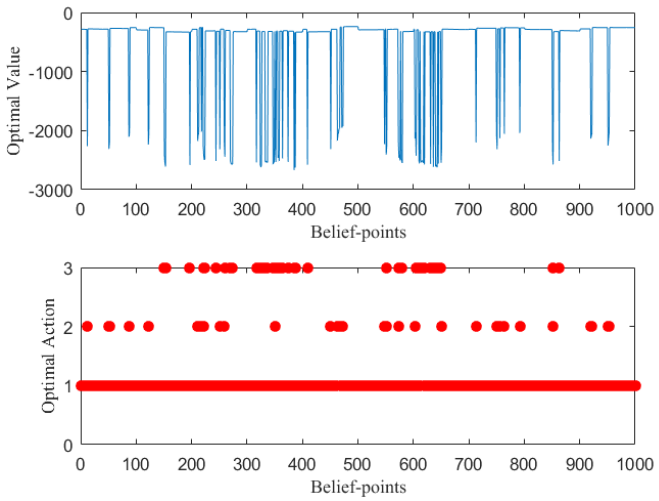


Figure 6.6 Optimal value and actions for OALH BAMDP

Figure 6.7 shows the boxplots of the optimal values corresponding to a_1, a_2 and a_3 for 3, 27 and 100 scenario cases that we have discussed so far.

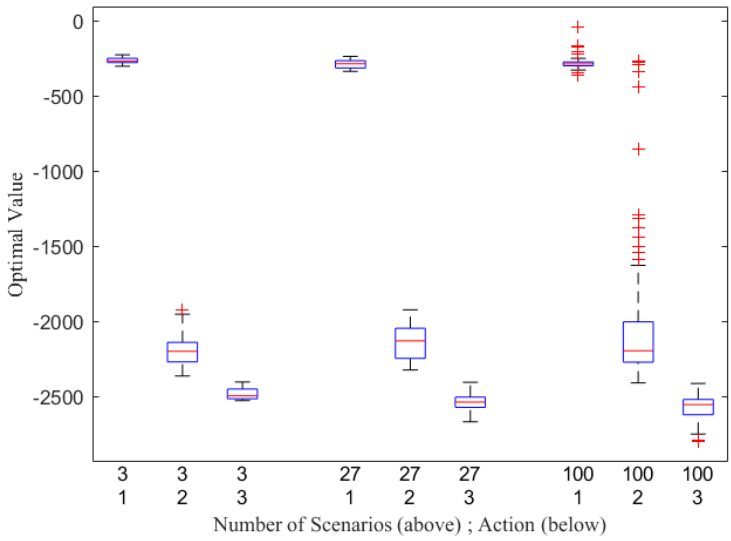


Figure 6.7 Boxplot of optimal values for different actions for 3, 27 and 100 scenario problems

Figure 6.8 shows a comparison between the three BAMDP methods described in this section. This is a plot of the sum of all optimal values for the 1000 belief-points generated by the random walk in PERSEUS. It gives us an idea on how the values converge in the three cases. Apparently the 3 scenario case shows fastest convergence and the 100 scenario case is the slowest.

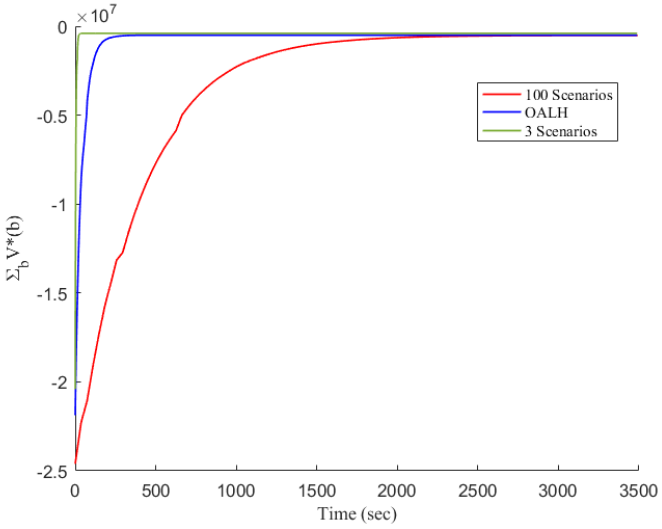


Figure 6.8 Plot of sum of optimal values over all belief-points with time

Although the 3 scenario BAMDP converges the fastest, it might lead to spurious learning, since a vast number of scenarios will remain unrealized. An infinite scenario model would be much closer representation of reality, but it will lead to slow convergence. BAMDP with OALH sampling will ensure that scenarios which are less

likely also get sampled. This mitigates the problem arises from using small number of scenarios, but also shows overall reasonable convergence properties.

6.5 Multiple Identical Systems

In real world, a decision maker may have to manage more than one identical system together. For instance, the decision could be related to the maintenance of a group of computers in a laboratory, or maintenance of a fleet of trucks, policy implementation on general population and so on. The actions only depend on the state of the system, they do not vary based on the individual systems. For these situations, we can formulate the decision problem covering all the identical systems together. We call this the multiple identical systems (MIdS) formulation. For example, if there are 10 computers to maintain, each of which can be in any one of the 2 original states, and there are 3 actions to choose from, we can set up the system states as percentage of computers at different states of vulnerability. Since all the systems are identical, we will only consider the different combinations possible, and treat all the permutations among the systems with the same percentage distribution of states as equal. The new states will indicate the percentage of systems that are in a particular original state, we call these the *compound states*. Also, we will only include states which will have an integer value for the number of systems in a certain original states. For example, if there are 10 computers, then 60% (6 computers) of them are in original state 1 and 40% (4 computers) are in original state 2 makes a feasible compound state $(0.6,0.4)$, on the other hand 55% in original state 1 and

45% in original state 2 would not make a feasible compound state, since the number of computers cannot be in fractions.

Let there are N original states, K identical systems. Then the total number of compound states will be given by $N_c = \binom{N+K-1}{N-1}$. Let $\lambda_i : \{0 \leq \lambda_i \leq 1; \sum_{i=1}^{N} \lambda_i = 1; \lambda_i K \in \{0\} \cup \mathbb{Z}^+\}$ be the fraction of the number of systems that are in state $i, i \in \{1, .., N\}$. Then a compound state can be represented as $S^c = \{\lambda_i\}_{i=1}^{N}$ where $c = \{1, .., N_c\}$.

It is interesting to note that since the systems are identical, for each combination $\{\lambda_i\}_{i=1}^{N}$, there will be $\binom{K}{\lambda_1 K, \ldots, \lambda_N K}$ permutations, where $\binom{K}{\lambda_1 K, \ldots, \lambda_N K} = \frac{K!}{(\lambda_1 K)!, \ldots, (\lambda_N K)!}$.

The *compound actions* are formed with the original states and original actions. So for our 2 state example, a compound action "11" would mean original action 1 when in original state 1 and also when in state 2. Consider the number of original actions to be U, the number of compound actions will be $U_c = N^U$. Table 6.6 shows an example involving 2 systems with 2 original states and 2 actions.

Table 6.9 Compound states for a 2 state 2 system multiple identical systems

Compound State	Proportions	System 1	System 2
1	(1,0)	Original State 1	Original State 1
2	(0.5,0.5)	Original State 1	Original State 2
3	(0,1)	Original State 2	Original State 2

Table 6.10 Compound actions for a 2 state 2 system multiple identical systems

Compound Actions	Original State1	Original State 2
11	Original Action 1	Original Action1
12	Original Action 1	Original Action2
21	Original Action 2	Original Action1
22	Original Action 2	Original Action2

When there are 2 original states, the compound states can be written as $\{\lambda, 1-\lambda\}$. Let compound state g is formed with $\{\lambda, 1-\lambda\}$ and state h is formed with $\{\lambda', 1-\lambda'\}$and compound action $a^c = \{a|i\}_{i=1}^{2}$, where a_i is the original action to be taken in original state i. Considering the state transitions are independent, the transition probabilities from compound state g to compound state h $(g, h \in \{1, .., N^c\})$ under compound action $a^c, a^c \in \{1, .., U^c\}$ for K identical systems can be calculated with the original state transition probabilities.

$$P_{gh}^{a_c} = \begin{cases} \left(p_{11}^{a|1}\right)^K for\ g = \{1,0\}, h = \{1,0\} \\ \left(p_{22}^{a|2}\right)^K for\ g = \{0,1\}, h = \{0,1\} \\ \sum_{\substack{i \in G \\ j \in H}} \prod_{k=1}^{K} p_{ij}^{a|i}\ otherwise \end{cases}$$

where $G \in \{\mathbf{1}_{\lambda_k}, \mathbf{2}_{1-\lambda_k}\}, H \in \{\mathbf{1}_{\lambda'_k}, \mathbf{2}_{1-\lambda'_k}\}$, considering $\boldsymbol{t}_n$is a vector with n elements, all being equal to t.

6.5.1 Results

We have created a MIdS model with 2 systems and 2 states with 3 scenarios. We have used the same transition data as given in tables 6.1-6.2. After creating the BAMDP with 3 compound states and 9 compound actions, we derive the equivalent POMDP and follow the same method as described in section 6.2. The optimal policies are derived from the α-vectors, which shows for the belief-points considered in PERSEUS, actions 11, 12 and 13 are the ones that came out to be optimal for different belief-points. It intuitively makes sense that all these compound actions consider auto-patching (original action 1) as optimal for low vulnerability states.

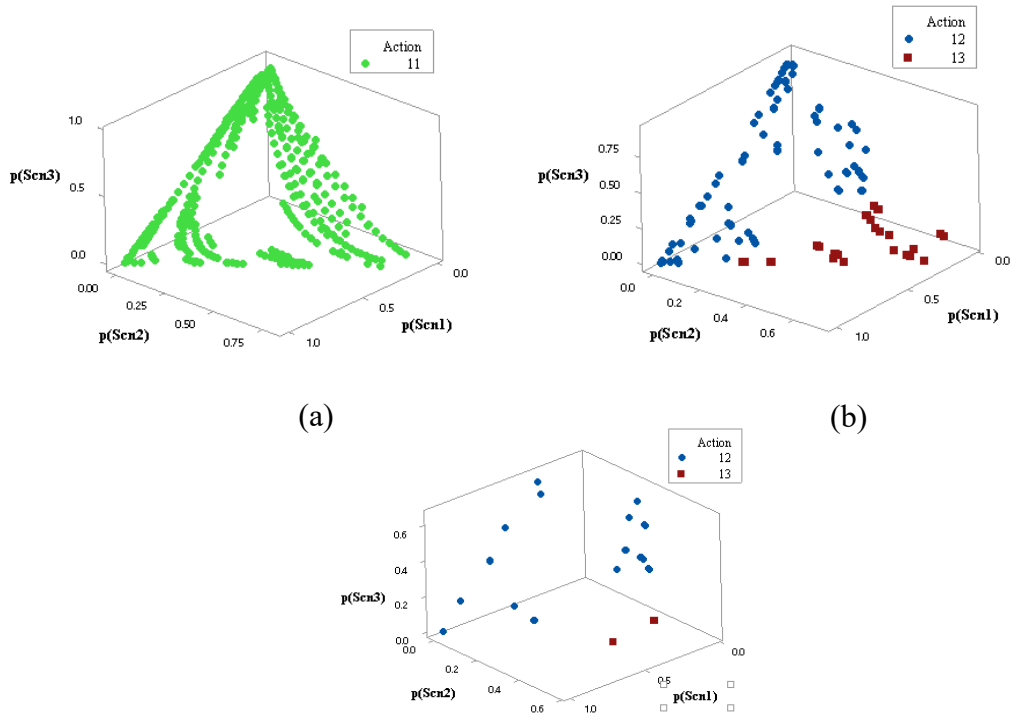


Figure 6.9 3D scatter plot for compound state (a) (1,0), (b) (0.5,0.5) and (c) (0,1)

Figures 6.9(a)-(c) show 3D scatter plots for the 3 different compound states respectively. For compound state 11, figure 6.9(a) shows that the optimal action is always 11, i.e. do automatic patching if the system is in low vulnerability state for all the scenarios. For compound state 11, both the hosts are in low vulnerability states, so it is optimal to take the action which is least expensive. 6.9(b) shows the compound state 12, which means one system is in low vulnerability state and the other system is in the high vulnerability state. Depending on the scenario compound actions 12 and 13 are optimal. It is important to note, that the cost of being at high vulnerability states, and the transition probabilities are different for different scenarios. Hence the optimal policies are dependent on the scenarios, which show that parametric uncertainty in the system makes a difference on the action policies. In Figure 6.9(c), for the compound state (0,1) there are very few data points, since this represents an extreme state where both the systems are at high vulnerability states. The effect of any action other than compound action 11 is to mitigate the high vulnerability state, which makes the random walk come out of the compound state of (0,1) quickly. Which explains why there are more points in compound states (1,0) and (0.5, 0.5) and so few points in (0,1).

6.6 Summary

In this chapter, we have implemented finite scenario BAMDP on a cyber vulnerability maintenance problem. Our method has the capability to learn through

interaction with the environment, and take action correspondingly. The decision-making agent is armed with the α-vectors generated by the BAMDP, and it takes action based on its perceived belief-state of the environment. We have shown that for belief-states with positive probability in low vulnerability states, the action automatic patching or a_1 is optimal. For the belief-states with positive probability in the high vulnerability natural states, for some cases research accept a_2 is optimal and complete overhaul or a_3 is optimal for the others. We then presented an Orthogonal Array Latin Hypercube based sampling method to sample the scenarios, and compared the results with 3 scenario and 100 scenario cases. The OALH based BAMDP has the advantage to avoid spurious learning and finding an optimal solution in a reasonable amount of time. Next we presented a method to determine maintenance policies where more than one identical systems are involved.

We call it "BAMDP for Multiple Identical Systems", an example of which is shown for a 2 systems 2 state 3 scenario problem. This method can be useful when a group of hosts are involved and a general policy is required for the entire group. This is also applicable for other processes where entities have identical properties with respect to the control problem, such as a group of vehicles, or a group of people and so on. BAMDP based vulnerability maintenance tools will help network administrators to make decisions more effectively. It can be used to build a more efficient alert system, which will notify the administrators about severity of system vulnerabilities with more precision. We believe this study can motivate more in depth research in using reinforcement learning methods to general policy making.

Chapter 7: Conclusions and Future Work

In this dissertation, we have explored deterministic and stochastic methods for data driven decision making. In Chapter 1, we discussed the motivation behind our research. We introduced the 4 questions that we are seeking to answer and put them in context. In this chapter, we will summarize our approach to answer these four questions, and discuss research avenues in relation to this work that can be pursued in future. Let us start with the deterministic problem.

7.1 GAGEDD for Automotive Stamping Scheduling

1. "How to generate schedules, assuming known demand and production times, within reasonable times which will minimize inventory starvation and changeovers?"

To answer this question, in Chapter 2 we have built an Integer Programming model for a typical stamping scheduling problem with known demand in automotive manufacturing. The model can be applied to other manufacturing processes like injection molding and die casting where die-sets are involved in the process. Our main contribution is to develop a hybrid solution method involving Genetic Algorithm and Earliest Due-date heuristic. We have implemented this method in a typical automotive stamping

scheduling scenario and compared the results with conventional methods like branch and bound and Genetic Algorithm. In Chapter 3, we have shown that when time is scarce, our method performs reasonably well, outperforming other methods. We have built software in Excel VBA which can be used as a decision support tool for similar scheduling purposes.

7.2 Future Research Related to GAGEDD

For the stamping scheduling problem, the immediate extension would be to consider stochastic demand. However, with stochastic demand, there would be more restrictions on the size of the problem that can be solved. The other issue with stamping scheduling is uncertainty over the down time of the press. We have not considered any downtime for the problem we solved, but in real life the stamping presses need to be shut down from time to time for maintenance and other operational reasons. Inventory management and scheduling of the machines can be done more effectively if these sources of uncertainties are accommodated in the model.

7.3 POMDP for Cyber Vulnerability Maintenance

Next, we explore stochastic methods for decision making in cyber security. We first start with a POMDP model to account for the uncertainty in the state of compromise of a host. Our research question for this topic is,

2. “How can we develop cyber maintenance policies when there is uncertainty in the state of compromise of a host?”

To answer this question in Chapter 4, we have reviewed MDP, and its extensions viz, POMDP and BAMDP. In Chapter 5, we first introduce parameters to account for the effect of different actions on the state of vulnerability and the state of compromise of a host. We assume that the state of vulnerability is observable, since the CVSS score of the vulnerabilities used as indicators of the state of vulnerability. However, it is not always obvious if a host is compromised or not. Our POMDP model operates over 3 actions, automatic patching, manual intervention and manual intervention with thorough forensic inspection (forensic scan). It should be noted that if the logging on the host is limited, the false negative probability of forensic scans could be quite high. We have used a point based value iteration method called PERSEUS (Spaan, Vlassis 2005) to solve the POMDP and derive optimal policies.

Our policies suggest that if there is even a small probability that the system is in the unknown state of compromise, inspection and manual intervention would be the optimal action. We have run a sensitivity analysis over the parameter of probability of going to a high vulnerability compromised state after manual intervention, and see an increasing trend in the expected cost.

7.4 Future Research Related to POMDP for Cyber Maintenance

Our POMDP method can be extend to build decision support tools with automated alert systems. This will help network administrators to effectively monitor the state of compromise of a host and take necessary actions. Our contribution in this area of research is to identify a number of transition parameters and use them to build a POMDP model for cyber maintenance policies. Moreover, we can explore ways to estimate these parameters that went into the model from real world data. Different cost structure can be incorporated in the model since not all vulnerabilities and breaches cost the same. The solver can be made faster by implementing other POMDP solution methods. Exploring different sampling methods for the belief-states to run PERSEUS may also help speed up the running time of the method. This method can be a motivation to build new event logging systems targeted to capture data to more accurately estimate the parameters used in this model. It can be integrated to create an intelligent alert system which network administrators can use.

7.5 BAMDP for Cyber Maintenance

In Chapter 6 we have implemented BAMDP (Duff 2002) to solve a cyber vulnerability maintenance problem. This solution method is motivated from the fact, that we had limited data available from the net-logs of an organization. Our objective is to account for the parametric uncertainty inherent in the problem due to the unavailability of

cost data and frequency of certain state transitions. The research question we ask for this topic is,

3. “With the available data on vulnerability state transitions along with its limitations, how can we account for the parametric uncertainty with reasonable confidence and make policies?”

We have explored finite scenario methods, where we have used Dirichlet sampling for the uncertain parameters to generate multiple scenarios. The scenarios are then merged to an equivalent POMDP and PERSEUS is used to solve the POMDP. We show through the 3D scatter plots of a 3 scenario problem how different scenarios might influence the optimal policies. This method can lead to the development of a decision support system where the decision maker can leverage his/her process knowledge for the uncertain parameters. Most of the research related to reinforcement learning and BAMDP find application in robotics and multi-arm bandit problems. Our contribution is a step towards taking advantage of these methods to make more general policies. In robot movement problems, typically, learning is one-step at a time, whereas in our extended formulation, every step learns more information on all the state transitions.

With finite scenario models, the problem of how many scenarios will be sufficient poses an important question. If the number of scenarios is too small, then there could be learning which does not correspond to real learning, i.e., “*spurious”* learning. If the number of scenarios is too large, the problem cannot be solved in a reasonable amount of time. To find a balance between these 2 extremes, our main contribution in this area of research is the application of Orthogonal Array based Latin Hypercube (OALH) designs

(Tang 1993) for sampling of the uncertain parameters to generate the scenarios. We demonstrate the method with an example using a strength 3 Orthogonal Array Latin Hypecube (OALH) to sample for the 3 uncertain parameters of the model. The design has 27 runs, so we created 27 scenarios each run catering to the 3 uncertain parameters of the model. We compare this model with a 3 scenario and a 100 scenario models, and show that 3 scenario converges the fastest, and the 100 scenario model is the slowest, while the 27 scenario OALH based method show reasonable convergence properties. The OALH sampling makes sure that the transition events that are less likely also gets sampled. This is part of the exploration and exploitation tradeoff inherent in BAMDP and reinforcement learning. OALH sampling provides more effective exploration, so that an optimal policy can be derived in a reasonable amount of time.

Our other contribution related to cyber maintenance problems is to model Multiple Identical Systems (MIdS) with BAMDP. The question we ask in connection with decision making for multiple systems is:

4. “How to design efficient models for multiple systems with identical characteristics?”

. Our contribution is to motivate the idea of building models for multiple systems with identical characteristics. When there are more than one systems involved and they are identical in characteristics for the control problem, we have shown how to build compound states and compound actions, which can be effectively used to control several systems together. This method has the potential to leverage even faster learning, since every step encounters multiple transitions. We have presented an example with 2 systems

and 3 scenarios, for which we have derived the policies involving compound states and actions.

7.6 Future Research BAMDP for Cyber Maintenance

There are many ways our research can be exploited to build more effective and faster learning models to derive decision policies. Markov Decision Process models are computationally expensive for problems with large dimensions. So naturally, BAMDP models are more difficult to solve. Research can be directed towards deriving more computationally efficient methods involving large number of diverse scenarios. For the decision problems similar to the cyber maintenance problems described in our research, we can already start with some known α-vectors which are associated with the optimal policies of a previous run of the process, so that we don't have to spend as much time deriving the new policies.

The data scarcity issues can be addressed in a more structured way using predictive tools like logistic regression. The Multiple Identical Systems method is only developed for 2 state processes, the method can be extended for processes with higher number of states. However, with large number of systems, the number of compound states will grow and the equivalent POMDP problem will be much more difficult to solve. Some actions are always dominated (e.g. action 22 in table 6.6b), so it would be interesting to see what happens if we do not consider them while solving the POMDP. In conclusion, the decision support tools must be usable by the decision makers in real time, so it would add

value to invest in research that would make the methods described here more computationally efficient.

################

List of References

Abbeel, P., Quigley, M., & Ng, A. Y. (2006, June). Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning* (pp. 1-8). ACM.

Allahverdi, A., Ng, C. T., Cheng, T. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. European Journal of Operational Research, 187(3), 985-1032.

Barlatt, A., Cohn, A., Fradkin, Y., Gusikhin, O., & Morford, C. (2009). Using composite variable modeling to achieve realism and tractability in production planning: an example from automotive stamping. IIE Transactions, 41(5), 421-436.

Barlatt, A. Y., Cohn, A. M., & Gusikhin, O. (2010). A hybridization of mathematical programming and dominance-driven enumeration for solving shift-selection and task-sequencing problems. Computers & Operations Research, 37(7), 1298-1307.

Barlatt, A. Y., Cohn, A., Gusikhin, O., Fradkin, Y., Davidson, R., & Batey, J. (2012). Ford motor company implements integrated planning and scheduling in a complex automotive manufacturing environment. Interfaces, 42(5), 478-491.

Barto, A.G., Bradtke, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. Artifcial Intelligence 72(1) (1995) 81-138

Barton, D., & Court, D. (2012). Making advanced analytics work for you. Harvard business review, 90(10), 78-83.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. ORSA journal on computing, 6(2): 154-160.

Bellman, R.E. (1957): Dynamic Programming. Princeton University Press, Princeton

Ben Hadj-Alouane, A., & Bean, J. C. (1997). A genetic algorithm for the multiple-choice integer program. Operations Research, 45(1): 92-101.

Blowers, M., & Williams, J. (2014). Machine learning applied to cyber operations. In *Network Science and Cybersecurity* (pp. 155-175). Springer New York.

Brafman, R. I., & Tennenholtz, M. (2002). R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, *3*(Oct), 213-231.

Cassandra, A. R., Kaelbling, L. P., & Littman, M. L. (1994, October). Acting optimally in partially observable stochastic domains. In *AAAI* (Vol. 94, pp. 1023-1028).

Çelebi, D. (2015). Inventory control in a centralized distribution network using genetic algorithms: A case study. Computers & Industrial Engineering, 87, 532-539.

Chen, T. M., Sanchez-Aarnoutse, J. C., & Buford, J. (2011). Petri net modeling of cyber-physical attacks on smart grid. IEEE Transactions on Smart Grid, 2(4), 741-749.

Cockburn, E. (2009). Websites Here, Websites There, Websites Everywhere..., But Are They Secure?. *The Quaestor Quarterly*, *4*(3), 1-4.

Duff, M. O. G. (2002). *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes* (Doctoral dissertation, University of Massachusetts Amherst).

Gencosman, B. C., Ozmutlu, H. C., Ozkan, H., Begen, M. A. (2014). Automotive stamping scheduling operations using mathematical programming and constraint programming. Industrial Engineering *Non-Traditional Applications in International Settings*, 37.

Gil, S., Kott, A., & Barabási, A. L. (2014). A genetic epidemiology approach to cyber-security. *Scientific reports*, *4*, 5659.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of discrete mathematics, 5: 287-326.

Grandview Research (2015).
http://www.grandviewresearch.com/industry-analysis/metal-stamping-market.

Groover, M. P. (2011). Introduction to manufacturing processes. Wiley Global Education.

Henke,N., Bughin,J., Chui, M., Manyika,J., Saleh,T., Wiseman, B., Sethupathy., G, (2016) McKinsey Global Institute
http://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/the-age-of-analytics-competing-in-a-data-driven-world

Holloway, E. M., Lamont, G. B., & Peterson, G. L. (2009, March). Network security using self- organized multi agent swarms. In Computational Intelligence in Cyber Security, 2009. CICS'09. IEEE Symposium on (pp. 144-151). IEEE.

Hou, C. (2015). Dynamic programming under parametric uncertainty with applications in cyber security and project management (Doctoral dissertation, The Ohio State University).

Howard, R.A(1960). Dynamic Programming and Markov Processes. The MIT Press, Cambridge
Kearns, M., & Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, *49*(2-3), 209-232.

Husbands, P. (1994). Genetic algorithms for scheduling. *AISB Quarterly*, 89: 38-45.

Jerald, J., Asokan, P., Prabaharan, G., & Saravanan, R. (2005). Scheduling optimisation of flexible manufacturing systems using particle swarm optimisation algorithm. The International Journal of Advanced Manufacturing Technology, 25(9-10): 964-971.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. Journal of artificial intelligence research, 4, 237-285.

Kott, A. (2014). Towards fundamental science of cyber security. In *Network Science and Cybersecurity* (pp. 1-13). Springer New York.

Lim, Y., Venugopal, R., & Ulsoy, A. (2014). Process control for sheet-metal stamping (pp. 978-1). Springer. ISBN.

McAfee 2014,
https://www.mcafee.com/us/resources/reports/rp-economic-impact-cybercrime2.pdf

McAfee, A., Brynjolfsson, E., Davenport, T. H., Patil, D. J., & Barton, D. (2012). Big data. The management revolution. Harvard Bus Rev, 90(10), 61-67.

Monma, C. L. & Potts, C.N. (1989). On the Complexity of Scheduling with Batch Setup Times. Operations Research, 37: 798–804.

Morton, T. (1993). Heuristic scheduling systems: with applications to production systems and project management (Vol. 3). John Wiley & Sons.

McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, *21*(2), 239-245.

National Science and Technology Council, Trustworthy cyberspace: strategic plan for the federal cybersecurity research and development program (2011), http://www.nitrd.gov/fileupload/files/Fed_Cybersecurity_RD_Strategic_Plan_2011.pdf

Norman, B. A., & Bean, J. C. (1997). Random keys genetic algorithm for job-shop scheduling. Engineering Design and Automation, 3: 145-156.

Pineau, J., Gordon, G., & Thrun, S. (2003, August). Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI* (Vol. 3, pp. 1025-1032).

Pochet, Y., & Wolsey, L. A. (2006). Production planning by mixed integer programming. Springer Science & Business Media.

Posner, M. E. (1985). Minimizing weighted completion times with deadlines. Operations Research, 33.3: 562-574.

Posner, M. E. (2011). Job Shop Scheduling. Wiley Encyclopedia of Operations Research and Management Science.

Potts, Chris N., & Van Wassenhove, L. N. (1992). Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. Journal of the Operational Research Society, 395-406.

Poupart, P., Vlassis, N., Hoey, J., & Regan, K. (2006, June). An analytic solution to discrete Bayesian reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning* (pp. 697-704). ACM.

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality* (Vol. 703). John Wiley & Sons.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Rahman, H. F., Sarker, R., & Essam, D. (2015). A genetic algorithm for permutation flow shop scheduling under make to stock production system. Computers & Industrial Engineering, 90, 12-24.

Ray, S., & Tadepalli, P. (2010). Model-Based Reinforcement Learning. *Encyclopedia of Machine Learning*, 690-693.

Rothschild, M. 1974. A two-armed bandit theory of market pricing.
Journal of Economic Theory 9 185–202.

Shani, G., Brafman, R. I., & Shimony, S. E. (2006, September). Prioritizing point-based POMDP solvers. In European Conference on Machine Learning (pp. 389-400). Springer Berlin Heidelberg.

Silver, E. A. (1963). *Markovian decision processes with uncertain transition probabilities or rewards* (No. ITR-1). MASSACHUSETTS INST OF TECH CAMBRIDGE OPERATIONS RESEARCH CENTER.

Smith, R. (2016, Dec 31). Business news: Fears over U.S. power grid --- recent cyberattacks in ukraine raise alarms over vulnerability of infrastructure here. *Wall Street Journal* Retrieved from
http://proxy.lib.ohio-state.edu/login?url=http://search.proquest.com.proxy.lib.ohio-state.edu/docview/1854260965?accountid=9783

Smith, W. E. (1956). Various optimizers for single-stage production. Naval Research Logistics Quarterly, 3(1-2), 59-66.
Sondik, E. J. (1978). The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations research*, *26*(2), 282-304.

Spaan, M. T., & Vlassis, N. (2005). PERSEUS: Randomized point-based value iteration for POMDPs. *Journal of artificial intelligence research*, *24*, 195-220.

Srivastava, A., Morris, T., Ernster, T., Vellaithurai, C., Pan, S., & Adhikari, U. (2013). Modeling cyber-physical vulnerability of the smart grid with incomplete information. IEEE Transactions on Smart Grid, 4(1), 235-244.

Sutton, R.S.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings of the Seventh International Conference on Machine Learning, Morgan Kaufmann (1990)216-224"Top 10 cybercrime stories 2016", ComputerWeekly.com,
http://www.computerweekly.com/news/450404344/Top-10-cyber-crime-stories-of-2016

Wang, X., & Dietterich, T. G. (2003). Model-based policy gradient reinforcement learning. In *ICML* (pp. 776-783).

Wessner CW., Wolff AW. (2012). National Research Council (US) Committee on Comparative National Innovation Policies: Best Practice for the 21st Century; editors. Rising to the Challenge: U.S. Innovation Policy for the Global Economy. Washington

(DC): National Academies Press (US); 2012.5, The New Global Competitive Environment. Available from: https://www.ncbi.nlm.nih.gov/books/NBK100306/

Wiering, M., & Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, Learning, and Optimization*, *12*.

White, C. C. (1980). Monotone control laws for noisy, countable-state Markov chains. European Journal of Operational Research, 5(2), 124-132.

Xia, C. H., & Dube, P. (2007). Dynamic Pricing in e-Services under Demand Uncertainty. Production and Operations Management, 16(6), 701-712.

Zhang, W., & Dietterich, T. G. (1995, August). A reinforcement learning approach to job-shop scheduling. In IJCAI (Vol. 95, pp. 1114-1120).
Wu, C. C., Yin, Y., & Cheng, S. R. (2011). Some single-machine scheduling problems with a truncation learning effect. Computers & Industrial Engineering, 60(4): 790-795.

Zegordi, S. H., Abadi, I. N., & Nia, M. A. (2010). A novel genetic algorithm for solving production and transportation scheduling in a two-stage supply chain. Computers & Industrial Engineering, 58(3), 373-381.