AN INTELLIGENT RESOURCE ALLOCATION DECISION SUPPORT SYSTEM WITH Q-LEARNING

YOW AI NEE

NATIONAL UNIVERSITY OF SINGAPORE

2009

AN INTELLIGENT RESOURCE ALLOCATION DECISION SUPPORT SYSTEM WITH Q-LEARNING

YOW AI NEE

(B.Eng.(Hons.), NTU)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2009

Acknowledgement

I would like to express my greatest sincere gratitude to my academic supervisor, Dr. Poh Kim Leng for his guidance, encouragement, and support throughout my work towards this thesis. I especially appreciate his patience with a part-time student's tight working schedule. Without his help and guidance this work would not be possible. I especially acknowledge Industrial Engineering and Production Planning departments in my working company for providing technical data and resources to develop solutions. Last but not least, I am very thankful for the support and encouragement of my family.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Symbols

s_t	environment state at time t
a_t	execution at time t
r	reward function
R(s,a)	reward of performing action a in state s
π	policy
V	value function
V^{π}	value of state under policy π
π^*	optimal policy
Q*(s,a)	value taking action a in state s and then performing π^*
Q(s,a)	an estimate of Q*(s,a)
γ	discount factor
α	learning rate used in Q-learning
λ	parameter controls the combination between bootstrapping and measuring rewards over time
φ	relaxation coefficient

R	set of resources
S	set of resource states
O	set of operations
T	set of tasks
C	precedence constraints
N	set of completed tasks by time periods
I	probability distribution with initial states allocated to resources

List of Abbreviations

Acronym	Meaning
ADP	approximate dynamic programming
ANN	artificial neural network
ANOVA	analysis of variance
AI	artificial intelligence
BOM	bill of materials
EMPtime	expected mean processing time
DP	dynamic programming
GA	genetic algorithm
IMS	intelligent manufacturing system
JSP	job-shop scheduling problem
MDP	markov decision processes
MC	monte carlo method
ML	machine learning
NDP	neuro-dynamic programming
RAP	resource allocation problem
RCPSP	resource constrained project scheduling problem
RL	reinforcement learning
SSP	stochastic shortest path
STAP	semiconductor testing accelerated processing
SVM	support vector machine
TD	temporal difference learning
TSP	traveling salesman problem

Summary

The dissertation aims at studying the learning effect of resource allocation problem (RAP) in the context of wafer testing industry. Machine learning plays an important role in the development of system and control application in manufacturing field with uncertain and changing environments. Dealing with uncertainties at today status on Markov decision processes (MDP) that lead to the desired task can be difficult and time-consuming for a programmer. Therefore, it is highly desirable for the systems to be able to learn to control the policy in order to optimize their task performance, and to adapt to changes in the environment.

Resource management task is defined in the wafer testing application for this dissertation. This task can be decomposed into individual programmable behaviours which "capacity-planning" behaviour is selected. Before developing learning onto system, it is essential to investigate stochastic RAPs with scarce, reusable resources, non-preventive and interrelated tasks having temporal extensions. A standard resource management problem is illustrated as reformulated MDP example in the behaviour with reactive solutions, followed by an example of applying to classical transportation problem. This reformulation has a main advantage of being aperiodic, hence all policies are proper and the space of policies can be safely restricted.

Different learning methods are introduced and discussed. Reinforcement learning method, which enables systems to learn in changing environment, is selected. Under this reinforcement learning method, Q-learning algorithm is selected for implementing learning on the problem. It is a technique for solving learning problems when the model of the environment is unknown. However, current Q-learning algorithm is not suitable for large-scale RAP: it treats continuous variables. Fuzzy logic tool was proposed to deal with continuous state and action variables without discretising.

All experiments are conducted on a real manufacturing system in a semiconductor testing plant. Based on the results, it was found that a learning system performs better than a non-learning one. In addition, the experiments demonstrated the convergence and stability of Q-learning algorithm, which is possible to learn in presence of disturbances and changes are demonstrated.

Chapter 1

Introduction

Allocation of the resources of a manufacturing system has played an important role in improving productivity in factory automation of capacity planning. Tasks performed by system in factory environment are often in sequential order to achieve certain basic production goals. The system can either pre-programmed or plan its own sequence of actions to perform these tasks. Facing with today's rapid market changes, a company must execute manufacturing resource planning through negotiating with customers for prompt delivery date arrangement. It is very challenging to solve such a complex capacity allocation problem, particularly in a supply chain system with a seller–buyer relationship. This is where we mostly have only incomplete and uncertain information on the system and the environment that we must work with is often not possible to anticipate all the situations that we may be in. Deliberative planning or pre-programming to achieve tasks will not be always possible under such situations. Hence, there is a growing research interest in imbuing manufacturing system not only with the capability of decision making, planning but also of learning. The goal of learning is to enhance the capability to deal and adapt with unforeseen situations and circumstances in its environment. It is always very difficult for a programmer to put himself in the shoes of the system as he must imagine the views autonomously and also need to understand the interactions with the real environment. In addition, the hand-coded system will not continue to function as desired in a new environment. Learning is an approach to these difficulties. It reduces the

required programmer work in the development of system as the programmer needs only to define the goal.

1.1 Importance of Learning

Despite the progress in recent years, autonomous manufacturing systems have not yet gained the expected wide-spread use. This is mainly due to two problems: the lack of knowledge which would enable the deployment of systems in real-world environments and the lack of adaptive techniques for action planning and error recovery. The adaptability of today's systems is still constrained in many ways. Most systems are designed to perform fixed tasks for short limited periods of time. Researchers in Artificial Intelligence (AI) hope that the necessary degree of adaptability can be obtained through machine learning techniques.

Learning is often viewed as an essential part of an intelligent system; of which robot learning field can be applied to manufacturing production control in a holistic manner. Learning is inspired by the field of machine learning (a sub field of AI), that designs systems which can adapt their behavior to the current state of the environment, extrapolate their knowledge to the unknown cases and learn how to optimize the system. These approaches often use statistical methods and are satisfied with approximate, suboptimal but tractable solutions concerning both computational demands and storage space.

The importance of learning was also recognized by the founders of computer science. John von Neumann (1987) was keen on artificial life and, besides many other things, designed self-organizing automata. Alan Turing (1950) who in his famous paper, which

can be treated as one of the starting articles of AI research, wrote that instead of designing extremely complex and large systems, we should design programs that can learn how to work efficiently by themselves. Today, it still remains to be shown whether a learning system is better than a non-learning system. Furthermore, it is still debatable as to whether any learning algorithm has found solutions to tasks from too complex to hand-code. Nevertheless, the interest in learning approaches remains high.

Learning can also be incorporated into semi-autonomous system, which is a combination of two main systems: the teleoperation (Sheridan, 1992) and autonomous system concept (Baldwin, 1989). It gains the possibility that system learns from human or vice versa in problem solving. For example, human can learn from the system by observing its performed actions through interface. As this experience is gained, human learns to react the right way to the similar arising events or problems. If both human and system's capabilities are fully optimized in semi-autonomous or teleoperated systems, the work efficiency will increase. This applies to any manufacturing process in which there exist better quality machines and hardworking operators; there is an increase in production line efficiency.

The choice of implementing learning approaches depends on the natures of the situations that trigger the learning process in a particular environment. For example, supervised learning approach is not appropriate to be implemented in situations, which learning takes place through the system's interaction with the environment. This is because it is impractical to obtain sufficiently correct and state representative examples of desired goal in all situations. Therefore it is better to implement continuous and on-line processes in the dynamic environment, i.e. through unsupervised learning approach. However, it may

have difficulty sensing the actual and true state of the environment due to fast and dynamic changes in seconds. Hence there is a growing interest in combining both supervised and unsupervised learning to achieve full learning to manufacturing systems.

1.2 Problem Domain: Resource Management

In this thesis, we consider resource management as an important problem with many practical applications, which has all the difficulties mentioned in the previous parts. Resource allocation problems (RAPs) are of high practical importance, since they arise in many diverse fields, such as manufacturing production control (e.g., capacity planning, production scheduling), warehousing (e.g., storage allocation), fleet management (e.g., freight transportation), personnel management (e.g., in an office), managing a construction project or controlling a cellular mobile network. RAPs are also related to management science (Powell and Van Roy, 2004). Optimization problems are considered that include the assignment of a finite set of reusable resources to non-preemptive, interconnected tasks that have stochastic durations and effects. Our main objective in the thesis is to investigate efficient decision-making processes which can deal with the allocation for any changes due to dynamic demands of the forecast with scarce resources over time with a goal of optimizing the objectives. For real world applications, it is important that the solution should be able to deal with both large-scale problems and environmental changes.

1.3 Motivations of Thesis

One of the main motivations for investigating RAPs is to enhance manufacturing production control in semiconductor manufacturing. Regarding contemporary

manufacturing systems, difficulties arise from unexpected tasks and events, non-linearities, and a multitude of interactions while attempting to control various activities in dynamic shop floors. Complexity and uncertainty seriously limit the effectiveness of conventional production control approaches (e.g., deterministic scheduling).

This research problem was identified in a semiconductor manufacturing company. Semiconductors are key components of many electronic products. The worldwide revenues for semiconductor industry were about US$274 billion in 2007. Year 2008 is predicting a trend of 2.4% increase in the worldwide market (Pindeo, 1995; S.E. Ante, 2003). Because highly volatile demands and short product life cycles are commonplace in today's business environment, capacity investments are important strategic decisions for manufacturers. Figure 1 shows the installed capacity and demand as wafer starts in global semiconductor over 3 years (STATS, 2007). It is clearly seen that capacity is not efficiently utilized. In the semiconductor industry, where the profit margins of products are steadily decreasing, one of the features of the semiconductor manufacturing process is intensive capital investment.

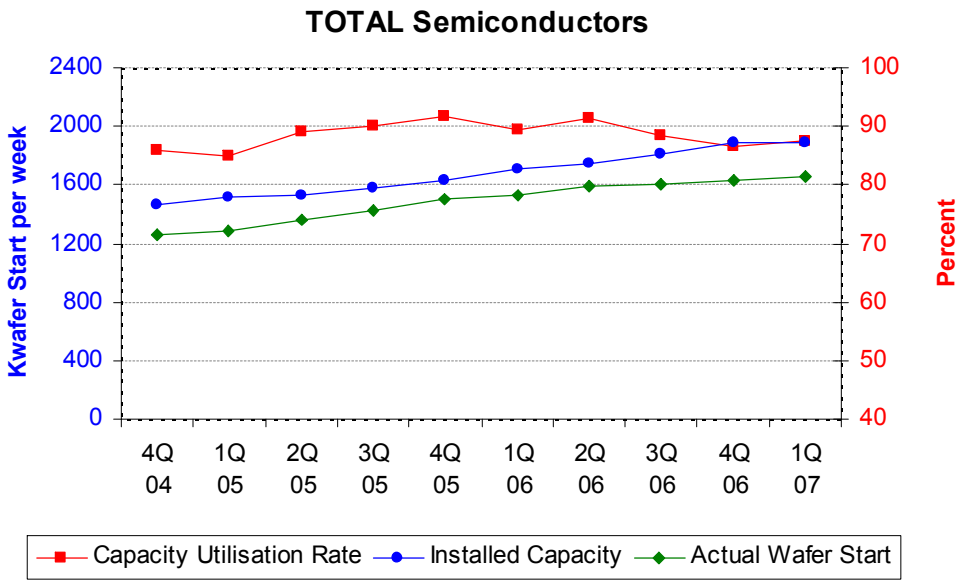


Figure 1.1: Capacity trend in semiconductor

Manufacturers may spend more than a billion dollars for a wafer fabrication plant (Baldwin, 1989; Bertsekas and Tsitsiklis, 1996) and the cost has been on the rise (Benavides, Duley and Johnson, 1999). More than 60% of the total cost is solely attributed to the cost of tools. In addition, in most existing fabs millions of dollars are spent on tool procurement each year to accommodate changes in technology. Fordyce and Sullivan (2003) regard the purchase and allocation of tools based on a demand forecast as one of the most important issues for managers of wafer fabs. Underestimation or overestimation of capacity will lead to low utilization of equipment or the loss of sales. Therefore, capacity planning, making efficient usage of current tools and carefully planning the purchase of new tools based on the current information of demand and capacity, are very important for corporate performance. This phenomenon of high cost of investment is needed for the corporate to close the gap between demand and capacity, is not limited to semiconductor company but is pervasive in any manufacturing industry. Therefore, many companies have exhibited the need to pursue better capacity plans and planning methods. The basic conventional capacity planning is to have enough capacity which satisfies product demand with a typical goal of maximizing profit.

Hence, resource management is crucial to this kind of high-tech manufacturing industries. This problem is sophisticated owing to task resource relations and tight tardiness requirements. Within the industry's overall revenue-oriented process, the wafers from semiconductor manufacturing fabs are raw materials, most of which are urgent orders that customers make and compete with one another for limited resources. This scenario creates a complex resource allocation problem. In the semiconductor wafer testing industry, a wafer test requires both a functional test and a package test. Testers are the

most important resource in performing chip-testing operations. Probers, test programs, loadboards, and toolings are auxiliary resources that facilitate testers' completion of a testing task. All the auxiliary resources are connected to testers so that they can conduct a wafer test. Probers upload and download wafers from testers and do so with an index device and at a predefined temperature. Loadboards feature interfaces and testing programs that facilitate the diagnosis of wafers' required functions. Customers place orders for their product families that require specific quantities, tester types, and testing-temperature settings. These simultaneous resources (i.e., testers, probers, and loadboards) conflict with the capacity planning and the allocation of the wafer testing because products may create incompatible relations between testers and probers.

Conventional resource management model is not fully applicable to the resolution of such sophisticated capacity allocation problems. Nevertheless, these problems continue to plague the semiconductor wafer testing industry. Thus, one should take advantages of business databases, which consist of huge potentially useful data and attributes implying certain business rules and know-how regarding resource allocation. Traditionally, people have used statistics techniques to carry out the classification of such information to induce useful knowledge. However, some implicit interrelationships of the information are hard to discover owing to noises coupled with the information.

Some main reasons to the challenging and difficult capacity planning decisions are addressed below:

- *Highly uncertain demand:* In the electronics business, product design cycles and life cycles are rapidly decreasing. Competition is fierce and the pace of product innovation is high. Because of the bullwhip effect of the supply chain (Geary, Disney

and Towill, 2006), the demand for wafers is very volatile. Consequently, the demand for new semiconductor products is becoming increasingly difficult to predict.

- *Rapid changes in technology and products:* Technology in this field changes quickly, and the state-of-art equipments should be introduced to the fab all the time (Judith, 2005). These and other technological advances require companies to continually replace many of their tools that are used to manufacture semiconductor products. The new tools can process most products including old and new products, but the old tools could not process the new products, and even if they can, the productivity may be low and quality may be poor. Moreover, the life cycle of products is becoming shorter. In recent years the semiconductor industry has seen in joint venture by companies in order to maximize the capacity. Fabs dedicated to 300 millimeter wafers have been recently announced by most large semiconductor foundries.
- *High cost of tools and long procurement lead time:* The new tools must be ordered several months ahead of time, usually ranging from 3 months to a year. As a result, plans for capacity increment must be made based on 2 years of demand forecasts. An existing fab may take 9 months to expand capacity and at least a year to equip a cleanroom. In the rapidly changing environment, forecasts are subject to a very high degree of uncertainty. As the cost of semiconductor manufacturing tools is high, it generally occupies 60% of capacity expenses. Thus, a small improvement in the tool purchase plan could lead to a huge decrease in depreciation factor.

Thus, the primary motivations for studying this wafer test resource allocation problem are:

- The problem is a complex planning problem at an actual industrial environment that has not been adequately addressed;

- Semiconductor test process may incur a substantial part of semiconductor manufacturing cost (Michael, 1996);
- Semiconductor test is important for ensuring quality control, and also provides important feedback for wafer fabrication improvement;
- Semiconductor test is the last step before the semiconductor devices leave the facility for packaging and final test; and
- Effective solution of the problem can reduce the cost of the test process by reducing the need to invest in new test equipment and cleanroom space (each test station may cost up to US$2 million).

In the thesis, both mathematical programming and machine learning (ML) techniques are applied to achieve the suboptimal control of a generalized class of stochastic RAPs, which can be vital to an intelligent manufacturing system (IMS) for strengthening their productivity and competitiveness. IMSs (Hatvany and Nemes, 1978) were outlined as the next generation of manufacturing systems that utilize the results of artificial intelligence research and were expected to solve, within certain limits, unforeseen problems on the basis of incomplete and imprecise information. Hence, this provides a solution approach that can be implemented into an industrial application.

1.4 Organization of Thesis

The structure of this thesis is as follows:

Chapter 2 provides a brief literature review to resource allocation is given, followed by a section on Markov decision processes (MDPs) which constitute the basis of the presented approach. This will discuss on the selection of a suitable learning method based on the

advantages of the learning method related to dealing with uncertainties concerning resource allocation strategy with a given an MDP based reformulation; i.e. real-time, flexibility and model-free. The selected learning method is reinforcement learning. It describes a number of reinforcement learning algorithms. It focuses on the difficulties in applying reinforcement learning to continuous state and action problems. Hence it proposes an approach to the continual learning in this work. The shortcomings of reinforcement learning and in resolving the above problems are also discussed.

Chapter 3 concerns with the development of the system for learning. This learning is illustrated with an example of resource management task where a machine capacity learns to be fully saturated with early delivery reactively. The problems in real implementation are addressed, and these include complex computation and real-time issues. Suitable methods are proposed including segmentation and multithreading. To control the system in a semi-structured environment, fuzzy logic is employed to react to real-time information of producing varying actions. A hybrid approach is adopted for the behaviours coordination that will introduce subsumption and motor-schema models.

Chapter 4 presents the experiments conducted by using the system. The purpose of the experiments is to illustrate the application of learning to a real RAP. An analysis of variance is performed based on the experimental results for testing how significant is the influence of learning compared to one without learning.

Chapter 5 discusses on the pros and cons of the proposed learning system. In addition, the overcoming of the current problems with learning on system is discussed.

Chapter 6 summarizes the contributions and concludes this work as well as recommends future enhancements.

Chapter 2

Literature Review and Related Work

Generally speaking, resource allocation learning is the application of machine learning techniques to RAP. This chapter will address the aspects of learning. Section 2.1 discusses the curse(s) of dimensionality in RAPs. Section 2.2 discusses the framework of stochastic resource allocation problem which is formulated with the reactive solution as a control policy of a suitably defined Markov decision process in Section 2.3 and Section 2.4. Section 2.5 provides background information on learning and Section 2.6 identifies and discusses different feasible learning strategies. This ends with a discussion in selecting the appropriate usable learning strategy for this research by considering necessary criteria. Section 2.7 introduces reinforcement learning. Since the difficulty in simulating or modeling an agent's interaction with its environment is present, it is appropriate to consider a model-free approach to learning. Section 2.8 discusses the reinforcement learning methods: dynamic programming, Monte Carlo methods and the Temporal-Difference learning. Section 2.9 classifies off-policy and on-policy learning algorithms of Temporal-Difference learning method. As in real world, the system must deal with real large-scale problems, and learning systems that only cope with discrete data are inappropriate. Hence Section 2.10 discusses Q-learning algorithm as the proposed algorithm for this thesis.

2.1 Curse(s) of Dimensionality

In current research, there are exact and approximate methods (Pinedo, 2002) which can solve many different kinds of RAPs. However, these methods primarily deal with the static and strictly deterministic variants of the various problems. They are unable to handle uncertainties and changes. Special deterministic RAPs which appear in the field of combinatorial optimization, e.g., the traveling salesman problem (TSP) (Papadimitriou, 1994) or the job-shop scheduling problem (JSP) (Pinedo, 2002), are strongly NP-hard and they do not have any good polynomial-time approximation algorithms (Lawler, Lenstra, Kan and Shmoys, 1993; Lovász and Gács, 1999). In the stochastic case, RAPs are often formulated as Markov decision processes (MDPs) solved and by applying dynamic programming (DP) methods. However, these methods suffered from a phenomenon that was named "curse of dimensionality" by Bellman, and become highly intractable in practice. The "curse" refers to the growth of the computation complexity as the size of the problem increases. There are three types of curses concerning the DP algorithms (Powell, Van Roy, 2004) which motivated many researchers proposing approximate techniques in the scientific literature to circumvent this.

In the DP's context, 'the curse of dimensionality' (Bellman, 1961) is resolved by identifying the working regions of the state space through simulations and approximating the value function in these regions through function approximation. Although it is a powerful method in discrete systems, the function approximation can mislead decisions by extrapolating to regions of the state space with limited simulation data. To avoid excessive extrapolation of the state space, the simulation and the Bellman iteration must be carried out in a careful manner to extract all necessary features of the original state space. In discrete systems, the computational load of the Bellman iteration is directly

proportional to the number of states to be evaluated and the number of candidate actions for each state. The total number of discrete states increases exponentially with the state dimension. The stochastic, stage-wise optimization problems addressed in this thesis have the state and action variable dimensions that cannot be handled by the conventional value iteration. In addition, the DP formulation is highly problem dependent and often requires careful defining of the core elements (e.g. states, actions, state transition rules, and cost functions).

Unfortunately, it is not trivial to extend classical approaches, such as branch-and-cut or constraint satisfaction algorithms, to handle stochastic RAPs. Simply replacing the random variables with their expected values and, then, applying standard deterministic algorithms, usually, does not lead to efficient solutions. The issue of additional uncertainties in RAPs makes them even more challenging and calls for advanced techniques.

The ADP approach (Powell and Van Roy, 2004) presented a formal framework for RAP to give general solutions. Later, a parallelized solution was demonstrated by Topaloglu and Powell (2005). The approach concerns with satisfying many demands arriving stochastically over time having unit durations but not precedence constraints. Recently, support vector machines (SVMs) were applied (Gersmann and Hammer, 2005) to improve local search strategies for resource constrained project scheduling problems (RCPSPs). A proactive solution (Beck and Wilson, 2007) for job-shop scheduling problem was demonstrated based on the combination of Monte Carlo simulation and tabu-search.

The proposed approach builds on some ideas in AI robot learning field, especially the Approximate Dynamic Programming (ADP) method which was originally developed in the context of robot planning (Dracopoulos, 1999; Nikos, Geoff and Joelle, 2006) and game playing, and their direct applications to problems in the process industries are limited due to the differences in the problem formulation and size. In the next section, a short overview on MDPs will be provided as they constitute the fundamental theory to the thesis's approach in stochastic area.

2.2 Markov Decision Processes

In constituting a fundamental tool for computational learning theory, stochastic control problems are often modeled by MDPs. Over the past, the theory of MDPs has grown extensively by numerous researchers since Bellman introduced the discrete stochastic variant of the optimal control problem in 1957. These kinds of stochastic optimization problems have demonstrated great importance in diverse fields, such as manufacturing, engineering, medicine, finance or social sciences. This section contains the basic definitions, the applied notations and some preliminaries. MDPs (Figure 2.1) are of special interest for us, since they constitute the fundamental theory of our approach. In a later section, the MDP reformulation of generalized RAPs will be presentedso that machine learning technique can be applied to solve them. In addition, environmental changes are investigated within the concept of MDPs.

MDPs can be defined on a discrete or continuous state space, with a discrete action space, and in discrete time. The goal is to optimize the sum of discounted rewards. Here, by a (finite state, discrete time, and stationary, fully observable) MDP is defined as finite, discrete-time, stationary and fully observable where the components are:

X *denotes a finite set of discrete states*

A *denotes a finite set of control actions*

$\boldsymbol{A} : \boldsymbol{X} \rightarrow \boldsymbol{P(A)}$ *is the availability function that renders each state a set of actions available in that state where P denotes the power set.*

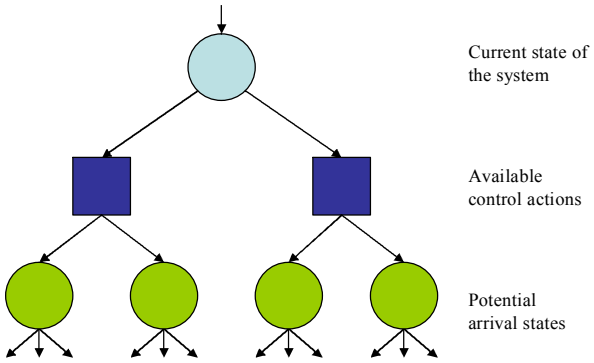
$p : \boldsymbol{X} \times \boldsymbol{A} \rightarrow \triangle(\boldsymbol{X})$ *is the transition-probability function where* $\triangle(\boldsymbol{X})$ *is the space of probability distributions over* ***X****. p(y | x, a) denotes the probability of arrival at state y after executing action* $a \in \boldsymbol{A}(x)$ *in state x.*

$g : \boldsymbol{X} \times \boldsymbol{A} \rightarrow \boldsymbol{R}$ *denotes the reward, or cost, function which is the cost of taking action a in state x.*

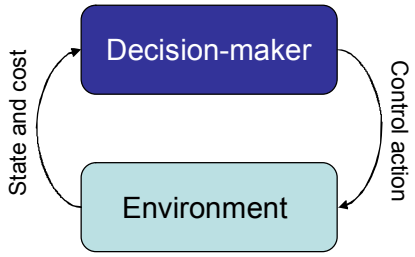
$\gamma \in [0, 1]$ *denotes the discount rate. If* $\gamma = 1$ *then the MDP is called undiscounted otherwise it is discounted.*

Once the problem has all these information available, this is known as a planning problem, and dynamic programming methods are the distinguished way to solve it. MDP is interpreted in learning viewpoint where we consider an agent acts in an uncertain environment. When the agent receives information about the state of the environment, x, the agent is allowed to choose an action $a \in \boldsymbol{A}(x)$ at each state. After the action is selected, the environment moves to the next state according to the probability distribution $p(x, a)$ and the decision-maker collects its one-step cost, $g(x, a)$. The aim of the agent is to find an optimal behavior that minimizes the expected costs over a finite or infinite horizon. It is possible to extend the theory to more general states (Aberdeen, 2003; Åström, 1965) and action spaces, but mathematical complexity will increase. Finite state and action sets are mostly sufficient for implemented action controls. For example, a stochastic shortest path (SSP) problem is a special MDP (Girgin, Loth, Munos, Preux, and Ryabko, 2008) in which the aim is to find a control policy such that it reaches a pre-

defined terminal state starting from a given initial state. Moreover, it plans to minimize the expected total costs of the path. A proper policy is obtained when it reaches the terminal state with probability of one. MDPs have an extensively studied theory and there exist a lot of exact and approximate solution methods, e.g., value iteration, policy iteration, the Gauss-Seidel method, approximate dynamic programming methods.



Temporal progress of the system



Interaction of the decision-maker and the uncertain environment

Figure 2.1: Markov decision processes

2.3 Stochastic Framework

A stochastic problem is characterized by an 8-tuple (***R,S,O,T,C,D,E,I***). In details the problem consists of Figure 2.2 shows the stochastic variants of the JSP and travelling salesman problem (TSP):

R *denotes a set of reusable resources.*

S *denotes a set of possible resource states.*

$\boldsymbol{O}$ *:* $\boldsymbol{T \subseteq O}$ *is the target of tasks which is defined as a set of allowed operations O given with a subset.*

$\boldsymbol{C \subseteq T \times T}$ *denotes a partial ordering by the precedence constraints between the tasks.*

$d : \boldsymbol{S} \times \boldsymbol{O} \rightarrow \triangle(\boldsymbol{N})$ is the durations of the tasks depending on the state of the executing resource, where $\boldsymbol{N}$ is the set of the executing resource with space of probability distribution.

$e : \boldsymbol{S} \times \boldsymbol{O} \rightarrow \triangle(\boldsymbol{S})$ is the uncertain effect which the state of the executing resource affect every task.

$i : \boldsymbol{R} \rightarrow \triangle(\boldsymbol{S})$ is the initial states of the resources that can be stochastic.

Since the ranges of randomizing functions *d, e* and *i* contain probability distributions, the corresponding random variables can be denoted as $\boldsymbol{D}$, $\boldsymbol{E}$ and $\boldsymbol{I}$, respectively. Thus, the notation $\boldsymbol{X} \sim f$ indicates that random variable $\boldsymbol{X}$ has probability distribution f.

$$\left.\begin{array}{l} D(s,o) \sim d(s,o) \\ E(s,o) \sim e(s,o) \\ I(r) \sim i(r) \end{array}\right\} s \in S, o \in O, r \in R \qquad (2.1)$$

The state of a resource can contain any relevant information, for example, its type and current setup (scheduling problems), its location and loading (transportation problems) or condition (maintenance and repair problems). Similarly, an operation (task) can affect the state in many ways, e.g., it can change the setup of the resource, its location or condition. The system must allocate each task to a resource. However, there may be cases when the state of a resource must be modified in order to be able to execute a certain task (for example, a transporter may need to travel to its source point first). In these cases non-task operations may be applied. They can modify the states of the resources without directly executing a task. It is possible to apply the non-task operation several times during the resource allocation process. However, the non-task operations are recommended to be avoided because of their high cost.

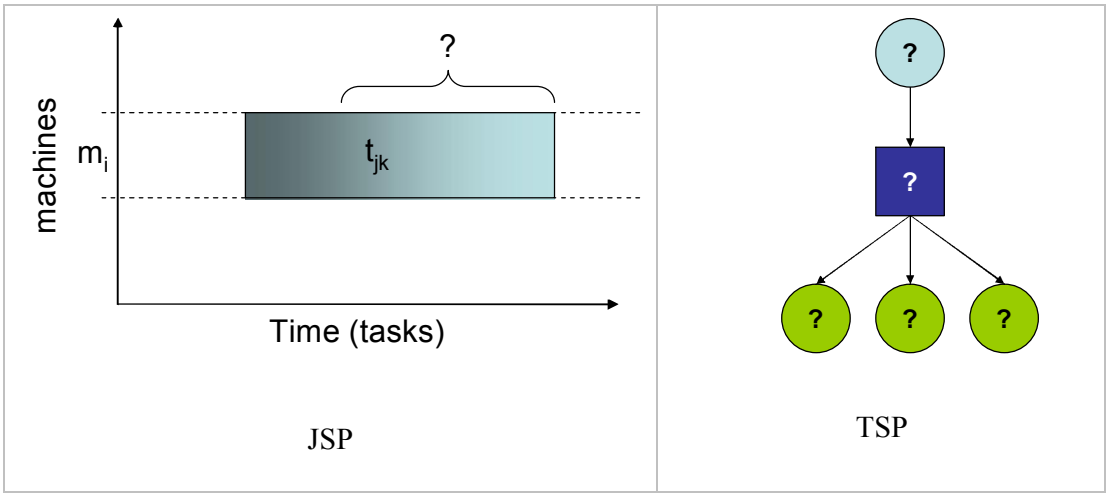


Figure 2.2: Examples of randomization in JSP and TSP

The performance of a solution in stochastic RAP is a random variable. Before introducing the basic types of resource allocation techniques classification, the concepts of "open-loop" and "closed-loop" controllers are addressed and illustrated in Figure 2.3. An open-loop controller, also called a non-feedback controller, computes its input into a system by using only the current state and its model of the system. Therefore, an open-loop controller does not use feedback to determine if its input has achieved the desired goal. It does not observe the output of the processes being controlled. In contrast, a closed-loop controller uses feedback to control the system (Sontag, 1998). Closed-loop control has a clear advantage over open-loop solutions in dealing with uncertainties. Hence, it also has improved reference tracking performance. It can stabilize unstable processes and reduce sensitivity to parameter variations.

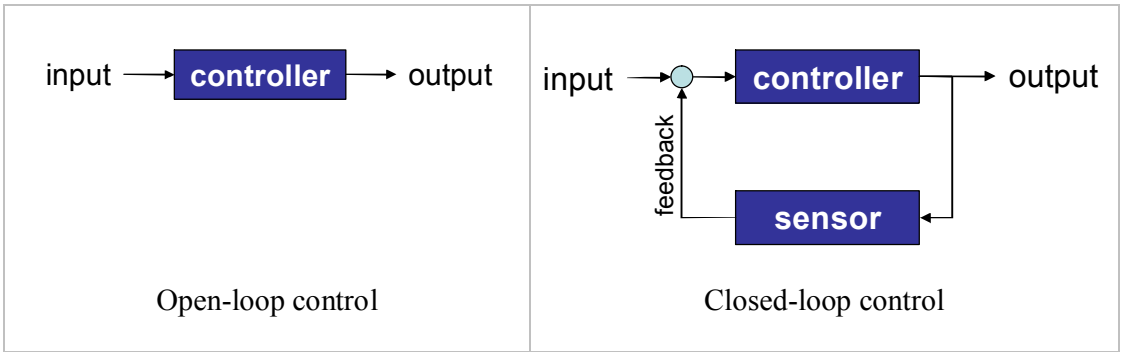


Figure 2.3: The concepts of open-loop and closed-loop controllers

In stochastic resource allocation there are some data (e.g., the actual durations) that will be available only during the execution of the plan. Based on the usage of this information, two basic types of solution techniques are identified. An open-loop solution that can deal with the uncertainties of the environment is called proactive. A proactive solution allocates the operations to resources and defines the orders of the operations, but, because the durations are uncertain, it does not determine precise starting times. This kind of technique can be applied only when the durations of the operations are stochastic, but, the states of the resources are known perfectly. Finally, in the stochastic case closed-loop solutions are called reactive. A reactive solution is allowed to make the decisions on-line, as the process actually evolves providing more information. Naturally, a reactive solution is not a simple sequence, but rather a resource allocation policy (to be defined later) which controls the process. The thesis mainly focuses on reactive solutions only. We will formulate the reactive solution of a stochastic RAP as a control policy of a suitably defined Markov decision process.

2.4 Learning

In this section, we aim to provide an effective solution to large-scale RAPs in uncertain and changing environments with the help of learning approach. The computer, a mere

computational tool, has developed into today's super complex micro-electronic device with extensive changes in processing, storage and communication of information. The main objectives of system theory in the early stages of development concerned the identification and control of well defined deterministic and stochastic systems. Interest was then gradually shifted to systems which contained a substantial amount of uncertainty. Having intelligence in systems is not sufficient; A growing interest in unstructured environments has encouraged learning design methodologies recently so that these industrial systems must be able to interact responsively with human and other systems providing assistance; and service that will increasingly affect everyday life. Learning is a natural activity of living organisms to cope with uncertainty that deals with the ability of systems to improve their responses based on past experience (Narendra and Thathachar, 1989).

Hence researchers looked into how learning can take place in industrial systems. In general, learning derives its origin from three fundamental fields: machine learning, human psychological (Rita, Richard and Edward, 1996) and biological learning. Here, machine learning is the solving of computational problems using algorithms automatically, while biological learning is carried out using animal training techniques obtained from operant conditioning (Touretzky and Saksida, 1997) that amounts to learning that a particular behavior leads to attaining a particular goal. Many implementations of learning have been done and divided by specific task or behaviour in the work of the above three fields. Now, among the three fields, machine learning offers the widest area of both research and applications/implementations to learning in robotics, IMS, gaming and etc. With that, we shall review on approaches to machine learning. In

the current state of-the-art, learning has few successful examples and is a very active area of research as learning is particularly difficult to achieve.

There are many learning definitions as posited by research; such as *"any change in a system that allows it to perform better the second time on repetition of the same task or another task drawn from the same population"* by Simon (1983), or *"An improvement in information processing ability that results from information processing activity"* Tanimoto (1994) and etc. Here, we adopt our operational definition by Arkin (1998) in adaptive behaviour context:

"Learning produces changes within an agent that over time enables it to perform more effectively within its environment."

Therefore, it is important to perform online learning using real-time data. This is the desirable characteristic for any learning method operating in changing and unstructured environments where the system explores its environment to collect sufficient feedback. Furthermore, learning process can be classified into different types (Sim, Ong and Seet, 2003) as shown in Table 2.1: unsupervised/supervised, continuous/batch, numeric/symbolic, and inductive/deductive. The field of machine learning has contributed to the knowledge of many different learning methods (Mitchell, 1997) which own its unique combination from a set of disciplines including AI, statistics, computational complexity, information theory, psychology and philosophy. This will be identified in a later section.

The roles and responsibilities of the industrial system can be translated into a set of behaviours or tasks in RAPs. *What is a behaviour?* A behaviour acquires information

about environment directly from data inputs (state) and is closely tied to the effectors that decide and carry out decision-made (action). In other words, a behaviour could be viewed as a function from state input into action output. A combination of behaviours can form a task.

Table 2.1: Descriptions of learning classifications (Sim, Ong and Seet, 2003)

Learning	Description
Unsupervised	No clear learning goal, learning based on correlations of input data and/or reward/punishment resulting from own behavior.
Supervised	Based on direct comparison of output with known correct answers.
Continuous	Takes place during operation in real world. (on-line)
Batch	A batch of learning examples is provided before making changes in behaviors. (off-line)
Inductive	Produces generalizations from input examples.
Deductive	Produces more efficient concept from the initial concept.
Numeric	Manipulates numeric quantities.
Symbolic	Relates representations with input data.

2.5 Behaviour-based Learning

In order to perform learning effectively, the system must have robust control architecture so as to cope with the environment complexity. Controlling a system generally involves complex operations for decision-making, data sourcing, and high-level control. To manage the controller's complexity, we need to constrain the way the system sources, reasons, and decides. This is done by choosing control architecture. There are a wide

variety of control approaches in the field of behaviour-based learning. Here, two fundamental control architecture approaches are described in the next subsections.

2.5.1 Subsumption Architecture

The methodology of the Subsumption approach (Brooks, 1986) is to reduce the control architecture into a set of behaviours. Each behaviour is represented as separate layers working on individual goals concurrently and asynchronically, and has direct access to the input information. As shown in Figure 2.4, layers are organized hierarchically. Higher layers have the ability to inhibit (I) or suppress (S) signals from the lower layers.

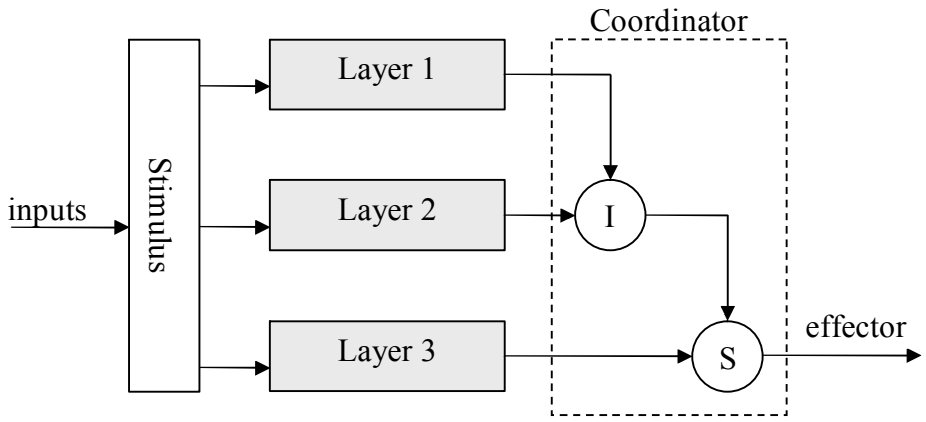


Figure 2.4: Subsumption Architecture

Suppression eliminates the control signal from the lower layer and substitutes it with the one proceeding from the higher layer. When the output of the higher layer is not active, the suppression node does not affect the lower layer signal. On the other hand, only inhibition eliminates the signal from the lower layer without substitution. Through these mechanisms, higher-level layers can subsume lower-levels.

2.5.2 Motor Schemas

Arkin (1998) developed motor schemas (in Figure 2.5) consisting of a behaviour response (output) of a schema which is an action vector that defines the way the agent reacts. Only the instantaneous executions to the environment are produced, allowing a simple and rapid computation. All the relative strengths of each behaviour determine the agent's overall response. Coordination is achieved through cooperative means by vector addition. Also no predefined hierarchy exist for coordination. The behaviours are configured at run-time based on the agent's intentions, capabilities, and environmental constraints.

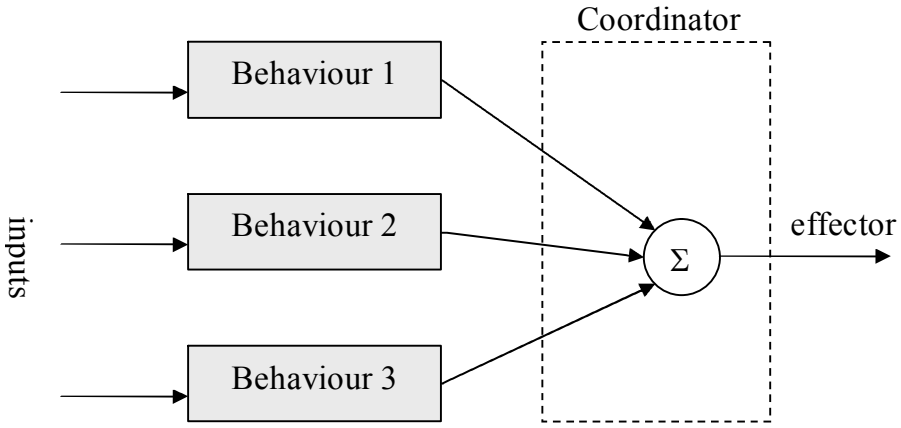


Figure 2.5: Motor Schemas approach

2.6 Learning Methods

Several solution methods are known and have succeeded in solving many different problems (Bertsekas and Tsitsiklis, 1996; Feinberg and Shwartz, 2003; Sutton and Barto, 1998) such as robotic control (Kalmár, Szepesvári and Lorincz, 1998), channel allocation (Singh and Bertsekas, 1997), transportation and inventory control (Van Roy, 1996), logical games and problems from financial mathematics, e.g., from the field of neuro-dynamic programming (NDP) (Van Roy, 2001) or reinforcement learning (RL) (Kaelbling, Littman and Moore, 1996), which compute the optimal control policy of an

MDP. In the following, four learning methods from a behaviour engineering perspective will be described with examples: artificial neural network, decision-tree learning, reinforcement learning and evolutionary learning. All of them have their strengths and weaknesses. The four different learning methods and their basic characteristics can be summarized in Table 2.2. Reinforcement learning approach is selected to give an effective solution to large-scale RAPs in uncertain and dynamic environments. Reinforcement learning allows a system to build and improve behaviour through trial and error on computing a good policy. The system must learn to act in a way that brings rewards over time. In next the sections, reinforcement learning is introduced in detail.

2.6.1 Artificial Neural Network

Artificial Neural Network (ANN) has the ability to learn by example and generalize their behaviour to new data. It is classified as supervised learning method, which requires large sets of representative patterns to characterize the environment during training. It is difficult to obtain the training patterns which contain no contradictory input output pairs. In this approach it is believed that intelligence arises in systems of simple, interacting components (biological or artificial neurons) through a process of learning and adaptation by which the connections between components are adjusted. Problem solving is parallel as all the neurons within the collection process their inputs simultaneously and independently (Luger, 2002).

ANN (Neumann, 1987) is compounded of a set of neurons which become activated depending on some inputs values. Each input, x, of the neuron is associated with a numeric weight, w. The activation level of a neuron generates an output, o. Neuron outputs can be used as the inputs of other neurons. By combining a set of neurons, and

using non-linear functions. Basically, there are three layers in an ANN (in Figure 2.6) – input layer, hidden layer and output layer. The inputs are connected to the first layer of neurons and the outputs of the second layer of neurons with nonlinearity correspond to the outputs of the last layer of neurons. The weights are the main ways of long-term storage, and learning usually takes place by updating the weights.

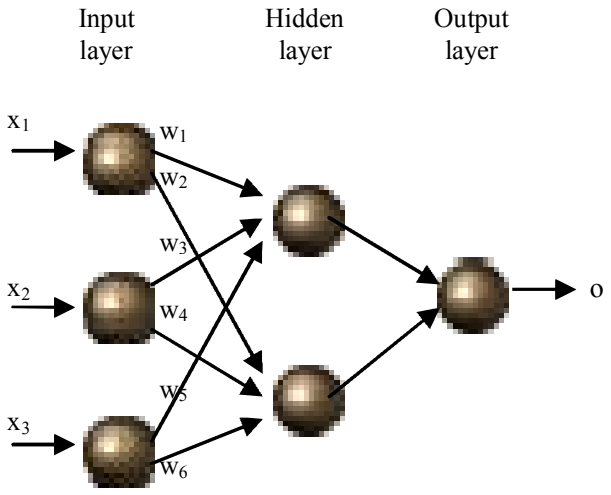


Figure 2.6: Layers of an artificial neural network

In order to develop ANN, it is essential to design the number of neurons needed in each layer, the appropriate types of neurons and the connection between the neurons. Next, the weights of the network are initialized and trained by learning for a task. Arkin (1998) has provided a good overview on the application of learning in neural network. For achieving good learning results, the number of units needs to be chosen carefully. Too many neurons in the layers may cause the network to overestimate the training data, while too few may reduce its ability to generalize. These selections are done through the experience of the human designer (Luger, 2002). Since an agent in a changing unstructured environment will encounter new data at all time, complex off-line retraining procedures would need to work out and considerable data amount would need to be stored for

retraining (Vijayakumar and Schaal, 2000). A detailed theoretical aspect of neural networks can be found in (Luger, 2002).

2.6.2 Decision Classification Tree

Decision classification tree learning is one of the most widely used and practical methods for inductive inference. It is a method for approximating discrete valued functions which is robust to noisy data and capable of learning disjunctive expressions (Mitchell, 1997). A decision tree takes input as an object or situation that is described by a set of properties and outputs a yes or no decision. Therefore decision trees also represent Boolean functions including functions with a larger range of outputs (Russel and Norvig, 1995). A number of different decision trees can be produced with a given set of training examples. Using a contemporary version of Ockham's razor that accepts the simplest answer to correctly fit the data, a decision tree can be chosen. In this case the smallest decision tree has correctly classified the data. Figure 2.7 shows an example of a simple decision tree.

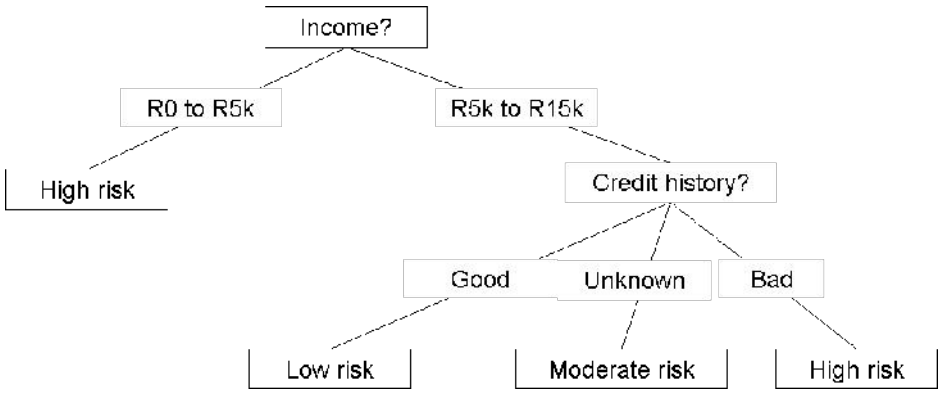


Figure 2.7: A decision tree for credit risk assessment

2.6.3 Reinforcement Learning

Reinforcement Learning (RL) (Kaelbling, Littman and Moore, 1996) in short, is a form of unsupervised learning method that learns behaviour through trial-and-error interactions with a dynamic environment. The learning agent senses the environment, chooses an action and the environment will give a reinforcement (reward or punishment) to it (as in Figure 2.8). The agent uses this information to optimize its policy to choose actions.

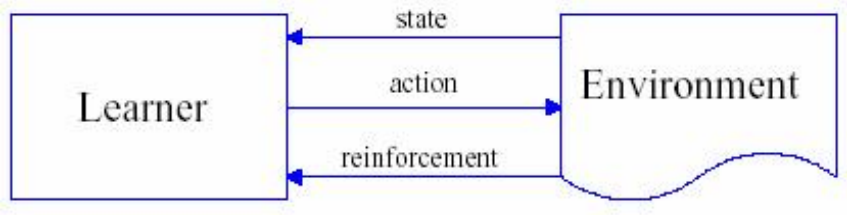


Figure 2.8: Interaction between learning agent and environment

RL is one of the most commonly used learning methods in control systems because it is numeric, inductive and continuous (Luger, 2002). This is an amazing method to program agent by reward and punishment without specifying the specific actions for achieving the goal; in short – easy and flexible in programming. It also has attracted rapidly increasing interest for its convergence properties, biological relevance and on-line learning.

Within the area of RL, there are wide ranges of learning techniques. These learning techniques can be classified into *model-free* (learn a controller without learning a model) or *model-based* (learn a model and use it to derive a controller) methods. Among these two methods, model-free method is the most commonly used method in behaviour engineering.

Zhang and Dietterich (1995) were the first to apply RL technique to solve NASA space shuttle payload processing problem. They used the TD(λ) method with iterative repair to this static scheduling problem. From then, researchers have suggested and addressed the field learning using RL for different RAPs (Csáji, Monostori and Kádár, 2003, 2004, 2006). Schneider (1998) proposed a reactive closed-loop solution using ADP algorithms to scheduling problems. Multilayer perceptron (MLP) based neural RL approach to learn local heuristics was briefly described by Riedmiller (1999). Aydin and Öztemel (2000) applied a modified version of Q-learning to learn dispatching rules for production scheduling. RL technique was used for solving dynamic scheduling problems in multiagent-based environment (Csáji and Monostori, 2005a, 2005b, 2006a, 2006b).

2.6.4 Evolutionary Learning

This method includes genetic algorithms (Gen and Cheng, 2000) and genetic programming (Koza and Bennett, 1999; Langdon, 1998). A key weakness of the evolutionary learning is that it does not easily allow for online learning. Most of the training must be done on a simulator, and then tested on a real-time data. However, designing a good simulator for a real-time problem operating in unstructured environments is an enormously difficult task.

Genetic algorithms (GA) are generally considered biologically inspired methods. They are inspired by Darwinian evolutionary mechanisms. The basic concept is that individuals within a population which are better adapted to their environment can reproduce more than individuals which are maladapted. A population of agents can thus adapt to its environment in order to survive and reproduce. The fitness rule (i.e., the reinforcement

function), measuring the adaptation of the agent to its environment (i.e., the desired behavior), is carefully written by the experimenter.

Learning classifier system principle: (Figure 2.9) an exploration function creates new classifiers according to a genetic algorithm's recombination of the most useful. The synthesis of the desired behaviour involves a population of agents and not a single agent. The evaluation function implements a behavior as a set of condition-action rules, or classifiers. Symbols in the condition string belong to {0, 1, #}, symbols in the action string belong to {0, 1}. # is the 'don't care' identifier, of tremendous importance for generalization. It allows the agent to generalize a certain action policy over a class of environmental situations with an important gain in learning speed by data compression. The update function is responsible for the redistribution of the incoming reinforcements to the classifiers. Classically, the algorithm used is the Bucket Brigade algorithm (Holland, 1985). Every classifier maintains a value that is representative of the degree of utility of classifiers. In this sense, genetic algorithms resemble a computer simulation of natural selection.

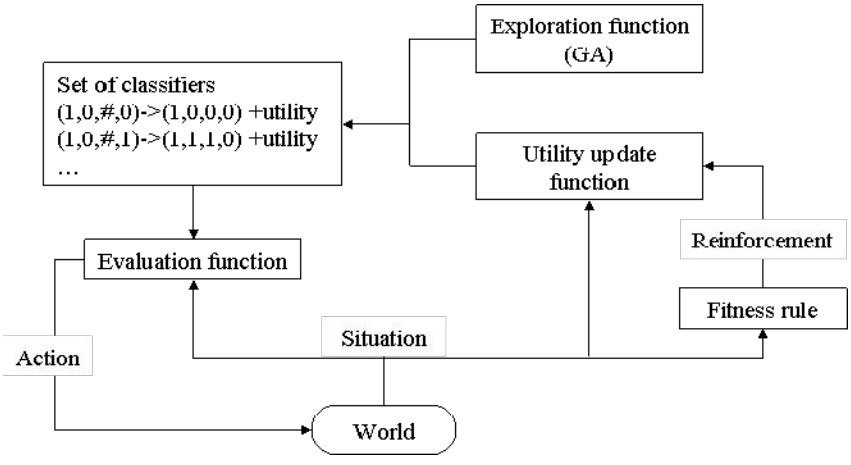


Figure 2.9: Learning classifier system

Table 2.2: Summary of four learning methods

Learning Methods	Learning process types[1]	Action decision-maker[2]	Input values determinate[3]	Pros	Cons
Neural Network	Supervised Batch Numeric Inductive	Back-propagation to minimize error	Function approximator	1. Insensitive to inaccurate inputs.	1. Requires large training data before learning starts.
					2. Slow learning in dynamic environment.
Decision Tree	Supervised Batch	Teacher	Function approximator	1. Able to incorporate with previous example data.	1. Only apply in simulation.
					2. Only outputs yes or no decision.
Reinforcement Learning	Un-supervised Continuous Numeric Inductive	Q-learning (commonly used)	Function approximator	1. Unsupervised learning and can perform model-free techniques.	1. Learning always starts from scratch with random action.
				2. Able to perform on-line learning in dynamic environment to deal with numeric quantities.	
				3. Is an automatic method and less time-consuming in learning.	
Evolutionary Learning	GA	Un-supervised Continuous Numeric Inductive Symbolic	Bucket Brigade	1. Useful optimization methods in simulations and can be tested on real-time data.	1. Requires a model of environment.
				2. Attractive in designing functional and adaptive system.	2. Can be ineffective in real environment and sensor model through simulation is not realistic.

[1]The learning processes as described in Table 2.2; [2]The algorithm(s) to define the output actions; [3]The function(s) to receive the input values from environment

2.7 Review on Reinforcement Learning

This section will introduce the selected learning method, reinforcement learning (RL). In any learning tasks, RL (Kaelbling, Littman and Moore, 1996; Sutton and Barto, 1998) system must find out by trial-and-error which actions are most valuable in particular states. In reinforcement learning system, the state represents the current situation of the

environment. The action is an output from the learning system that can influence its environment. The learning system's selection of executable actions in response to perceived states is called its policy.

In fact, RL lies between supervised learning, where teaching is from a supervisor, and unsupervised learning, where there is a no feedback. In RL, the feedback is provided in the form of a scalar reward that may be delayed. The reward is defined in relation to the task to be achieved; reward is given when the system is successful in achieving the task.

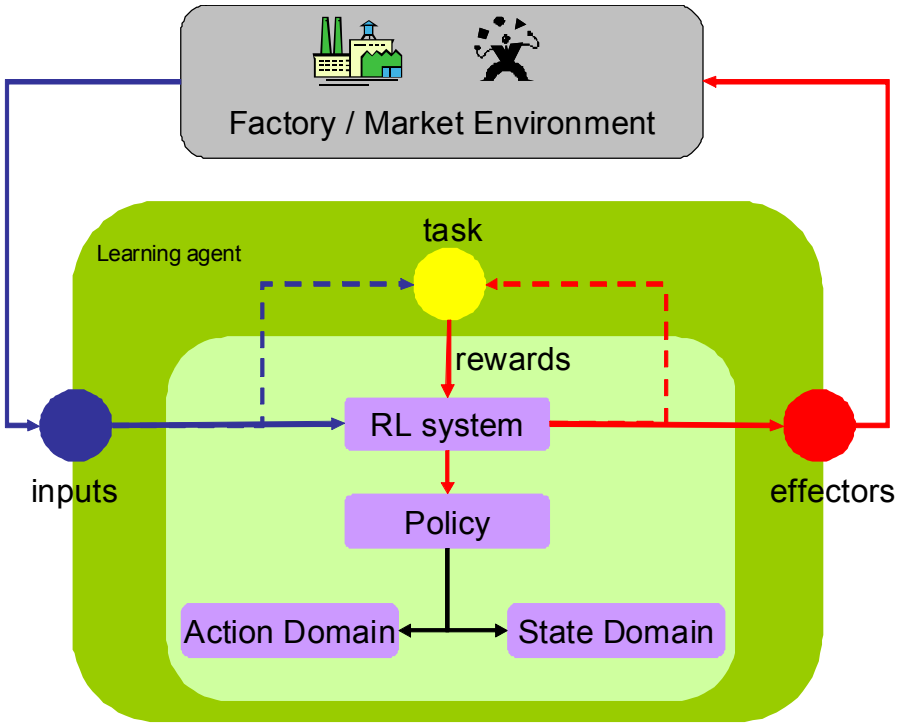


Figure 2.10: A basic architecture for RL

Figure 2.10 shows the interaction between the environment and the learning agent. The interaction between states and actions is usually regarded as occurring in a discrete time Markov environment: the probability distribution of the next-state is affected only by the execution of the action in the current state. The specified task block indicates the

calculation of reward for a particular combination of state, action, and next-state. The designer must design the reward function to suit the task initially. The reward signal can be made more positive to encourage a behaviour or more negative to discourage the behaviour. The RL system's purpose is to find a policy that maximizes the discounted sum of expected future rewards.

2.8 Classes of Reinforcement Learning Methods

The reinforcement learning methods are classified into three classes: dynamic programming, Monte Carlo methods and temporal-difference learning. Dynamic programming methods are well developed mathematically, but require a complete and accurate model of the environment. Monte Carlo methods are conceptually simple, but not suited for step-by-step incremental computation. Finally, temporal-difference methods require no model and are fully incremental.

2.8.1 Dynamic Programming

Dynamic programming (DP) is a model-based approach to solving RL problems. One form of DP stores the expected value of each action in each state. The action-value, $\boldsymbol{Q}$ (or Q-value), in a state is the sum of the reward for taking that action in that state plus the expected future reward. The action values are denoted $\boldsymbol{Q^*}$ if they satisfy the Bellman optimality equation:

$$Q^*(s_t, a_t) = E\left[r_t(s_t, a_t, S_{t+1}) + \gamma \max_{a_{t+1}} Q^*(S_{t+1}, a_{t+1}) \right] \forall s_t, a_t \qquad (2.2)$$

The DP approach finds $\boldsymbol{Q^*}$ iteratively through a forward dynamic model. There exist two usual methods to calculate the optimal policy: Policy iteration which consists of starting

from any initial policy and improving it iteratively; and Value iteration which looks for action of optimal value for every state. However, due to the "curse of dimensionality", computing an exact optimal solution by these methods is practically infeasible. In order to handle the "curse", we should apply approximate dynamic programming (ADP) techniques to achieve a good approximation of an optimal policy.

Approximate dynamic programming (ADP) (Powell, 2007) is a well-known dynamic programming approach to deal with large, discrete, continuous, states. All approaches are either learning or computing on a value function or directly a policy. To face continuous or large discrete states, these approaches rely on a function approximator to represent these functions. Today most works have dealt with parametric function approximators, though interesting non parametric approaches have also been published. The introductory discussion of function approximators will be presented in this section.

Control Policy

A control policy is defined such that the behavior of the learning agent at a given time in stationary. A deterministic policy is a function from states to control actions whereas a randomized policy is a function from states to probability distributions over actions.

Value Functions

Value functions (cost-to-go functions) are state-action pair functions that estimate how good a particular action will be in a given state, or what the return for that action is expected to be. The notation is used: $V^{\pi}(s_t)$ is the value of a state s under policy π. The expected return when s starts and following π thereafter over time. Estimating these value

functions so that they can be used to accurately choose an action will provide the best total possible reward after being in that given state.

$$V^{\pi}(s_t) = \mathrm{E}\left[\sum_{t=0}^{N} \alpha^t g(S_t, A_t^{\pi}) | S_0 = s_t\right] \quad (2.3)$$

where $\boldsymbol{S}_t, \boldsymbol{A}_t^{\pi}$ are random variables and g function is the cost of taking variable action $\boldsymbol{A}$ in state $\boldsymbol{S}$. $\boldsymbol{A}_t^{\pi}$ is chosen according to the control policy π and the distribution of $\boldsymbol{S}_{t+1}$ is $p(\boldsymbol{S}_t, \boldsymbol{A}_t^{\pi})$ where the horizon of the problem is assumed infinite. Similarly to the definition of $\boldsymbol{V}^{\pi}$, the action-value function of control policy in a learning problem can be defined below:

$$Q^{\pi}(s_t, \alpha\alpha) = \mathrm{E}\left[\sum_{t=0}^{N} \alpha^t g(S_t, A_t^{\pi}) | S_0 = s_t, A_0^{\pi} = \alpha\right] \quad (2.4)$$

where all the notations are the same as in equation (2.3). Action-value functions are important for model-free approaches, such as Q-learning algorithm (to be discussed in later section).

Bellman Equations

As the agent aims at finding an optimal policy which minimizes the expected costs, there is a need to have a concept for comparing policies. The agent receives information at each state of the environment, x, the agent is allowed to choose an action $a \in A(x)$. A policy $\pi_1 \leq \pi_2$ if and only if $\forall_x \in \boldsymbol{S} : \boldsymbol{V}_{\pi 1}(s_t) \leq \boldsymbol{V}_{\pi 2}(s_t)$. A policy is uniformly optimal if it is better than or equal to all other control policies. There always exists at least one optimal policy (Sutton and Barto, 1998). Although there may be many optimal policies, they all share the same unique optimized value function, $\boldsymbol{V}^*$. This function must satisfy the Hamilton-

Jacobi-Bellman equation (Van Roy, 2001) where ***T*** is the Bellman operator (Van Roy, 2001):

$$(TV)(s_t) = \min_{\alpha \in A(x)} \left[g(s_t, \alpha) + \alpha \sum_{y \in X} p(s_{t+1} \mid s_t, \alpha) V(s_{t+1}) \right] \tag{2.5}$$

2.8.2 Monte Carlo Methods

Monte Carlo (MC) methods are different in that they base the state value exclusively on experience, not on other state value estimates. Although a model is required, the model only needs to generate sample transitions, not the complete one as required by DP. MC records all the rewards following a state,

$$V^{\pi}(s_t) = (1-\alpha) V^{\pi}(s_t) + \alpha \sum_{k=0}^{n} r_{k+t} \tag{2.6}$$

where *n* is the terminal state, before updating the state value. This only makes sense in episodic tasks; otherwise MC would have to wait for a long time before updating. The obvious practical disadvantage of MC method is that the agent does not use its experience right away - it is in episode-by-episode sense before getting smarter.

2.8.3 Temporal Difference

A combination of the ideas from DP and MC methods yields Temporal Difference (TD) learning. Similarly to the MC method, this approach allows learning directly from the on-line or simulated experience without any prior knowledge of the system's model. The feature shared by TD and DP methods is that they both use bootstrapping for estimating the value functions.

TD algorithms make updates of the estimated values based on each observed state transition and on the immediate reward received from the environment on this transition. The simplest algorithm, One-step TD, performs the following update on every time step:

$$V^{\pi}(s_t) = (1-\alpha)V^{\pi}(s_t) + \alpha(r_{t+1} + \gamma V^{\pi}(s_{t+1})) \quad (2.7)$$

This method uses sample updates instead of full updates as in the case of DP. Only one successful state, observed during the interaction with the environment, is used to update $V^{\pi}(s_t)$ when receiving r_{t+1} (part of future reward). For any fixed policy π, the one-step TD algorithm converges in the limit to V^{π} in the mean for a constant step-size parameter α, and with probability 1 if the step-size satisfies between 0 and 1.

The one-step TD method can be used for the policy evaluation step of the policy iteration algorithm. As with MC methods, sufficient exploration in the generated experience must be ensured in order to find the optimal policy. One can use either on-policy or off-policy approaches to ensure an adequate exploration. Since we have identified the environment type to be model-free, we will focus on TD methods.

2.9 On-Policy and Off-Policy Learning

A policy in the definition of behaviour is a mapping from states to actions, while the estimation policy is to perform the action with the highest estimated action-value (Sutton and Barto, 1998).

On-Policy TD methods learn the value of the policy that is used to make decisions. The value functions are updated using results from executing actions determined by some policy. These policies are usually "soft" and non-deterministic. The meaning of "soft"

here is to ensure there is always an element of exploration to the policy. The policy is not so strict that it always chooses the action that gives the most reward.

Off-Policy TD methods can learn different policies for behaviour and estimation. Again, the behaviour policy is usually "soft" so there is sufficient exploration going on. Off-policy algorithms can update the estimated value functions using hypothetical actions, those which have not actually been tried. This is in contrast to on-policy methods which update value functions based strictly on experience; i.e. off-policy algorithms can separate exploration from control, and on-policy algorithms cannot. In other words, an agent trained using an off-policy method may end up learning tactics that it did not necessarily exhibit during the learning phase.

Two dimensions of variation can be examined in off-policy methods identification. The first variation is whether the policy is known or unknown. Methods capable of learning when the policy is unknown also can learn when the policy is known. The second variation is whether the learning system learns from a sequence of states, actions and rewards through time, or if it learns from non-sequential examples of state, action, next-state, and reward. By following the non-sequential definition of off-policy learning, to some researchers ‘off-policy’ implies ‘without following any policy’, or any sequence of actions.

Figure 2.11 classifies various model-free reinforcement learning methods or algorithms. Algorithms that can only learn on-policy include Actor-Critic, Sarsa and direct gradient ascent approaches. These systems allow occasional non-estimation policy actions in order

to explore and improve the policy; however, they cannot learn from an extended sequence of non-estimation policy actions.

One-step Q-learning is a non-sequential learning method. Since each update passes rewards only one step backwards through time, one-step Q-learning cannot exploit the information present in sequences of experiences. Rather than learn directly from the sequence of rewards, one-step Q-learning depends on bootstrapping, relying heavily on the accuracy of its internally represented estimates of value. Use of bootstrapping increases learning performance; however, over-reliance on bootstrapping causes instability (Sutton and Barto, 1998). Sequential learning methods that rely less heavily on bootstrapping can be expected to be more stable, and learn from experience more efficiently, than one-step Q-learning.

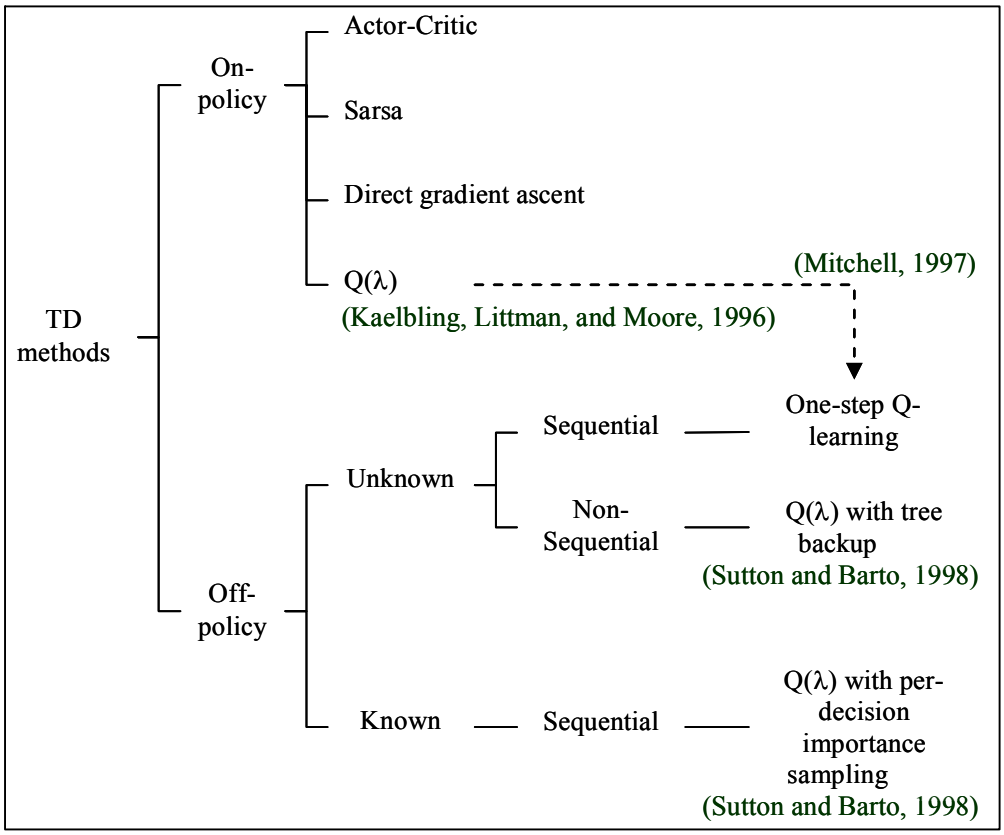


Figure 2.11: Categorization of off-policy and on-policy learning algorithms

TD(λ) is a well-known sequential learning algorithm that can be combined well with off-policy learning. The λ parameter controls the combination between bootstrapping and measuring rewards over time. Thus, reward is passed back over more than one step when $\lambda > 0$, which is also regarded as n-step TD learning. One-step Q-learning is equivalent to *TD(0)*. There are several Q(λ) algorithms that combine Q-learning and TD(λ). The degree of off-policy learning capabilities of the Q(λ) algorithms has been adjusted. Peng and William's (1996) method assumes that the number of non-estimation policy actions is small. Watkins's (1989) Q(λ) method requires that λ be set to zero for non-estimation policy actions (Sutton and Barto, 1998). Watkins's Q(λ) algorithm becomes one-step Q-learning if all actions are non-estimation policy actions. Accordingly, it is just as capable of off-policy learning as one-step Q-learning, and has additional sequential learning capabilities when following the estimation policy. However, the algorithm requires the detection of non-estimation policy actions. When actions are continuously variable, rather than discrete, it is questionable which actions follow the estimation policy. Therefore, a decision needs to be made whether the switch should be made between using actual rewards or estimated value.

Recently the statistical framework of importance sampling has led to a variation of Q(λ) that avoids the decision of whether an action has to follow the estimation-policy (Precup, Sutton and Singh, 2000). The tree backup algorithm multiplies λ by a factor that represents the probability that the estimation policy would have chosen that action. The probability replaces the problematic decision of whether an action exactly follows the estimation policy; as a result, the approach is applicable for continuous action problems.

2.10 RL Q-Learning

Q-learning is a variant of RL which is based on model-free method of learning from delayed reinforcement proposed by Watkins (1989), and Even-Dar and Mansour (2003) for solving Markovian decision problems with incomplete information. Some researchers have compared the RL methods empirically and found that Q-learning is the best. From then, Q-learning currently is the most widely used method within the family of RL in real world or simulation. This is due to the following reasons:

- The algorithm is simple.
- Simple programming is required because the system figures out the algorithm itself.
- If the environment changes, it does not need to be reprogrammed (off-policy).
- It is exploration insensitive: that is the Q-values will converge to the optimal values, independent of how the scenario behaves while the data is being collected.

Hence, we adopt Q-learning algorithm in this thesis.

2.11 Summary

Most of the practical applications in RAPs are difficult to solve, even for deterministic cases. It is known, for example, that both JSP and TSP are strongly NP-hard. Moreover, they do not have any good polynomial time approximation algorithm. In “real world” problems we often have to face uncertainties, for example the processing times of the tasks or the durations of the trips are unknown exactly in advance, and only estimations are available to work with. Unfortunately, it is not trivial to explore classical approaches, such as branch and cut or constraint satisfaction algorithms, to handle stochastic RAPs. This does not lead to efficient solutions by simply replacing the random variables with their expected values in the standard deterministic algorithms. For example, the RAP

formulation of TSP can be done as follows in 4-tuple. The set of resources represents one element, the "salesman", where $\boldsymbol{R} = \{r\}$. The states of resource r (the salesman) are $\boldsymbol{S} = \{s_1, \ldots, s_n\}$. If the state is s_i, it indicates that the salesman is in city i. The allowed operations are the same as the allowed tasks, $\boldsymbol{O} = \boldsymbol{T} = \{t_1, \ldots, t_n\}$, where the execution of task t_i indicates that the salesman travels to city i from his current location. The constraints $\boldsymbol{C} = \{\langle t_2, t_1\rangle, \langle t_3, t_1\rangle \ldots, \langle t_n, t_1\rangle\}$ are used for forcing the system to end the whole traveling in city 1, which is also the starting city. The performance measure is the latest arrival time.

On the basis of discussion on efficiently solving RAPs in the presence of uncertainties and the interest of using an unsupervised learning method to manage changes in dynamic environment, the decision was to go with the reinforcement learning method. In the context of resource allocation, reinforcement learning can be performed online in real-time and model-free environment as compared to the other learning methods. The method is automatic when the initial setup is done. It is different from evolutionary learning method as evolutionary learning first performs offline learning under simulation before testing on real-time data, which can result in ineffective learning. This fact makes reinforcement learning a better candidate for performing tasks in realistic changing environment with just feedback signal from the action the IMS performs. The design of the proposal of applying Q-learning to stochastic RAPs will be presented in Chapter 3.

Chapter 3

System Architecture and Algorithms for Resource Allocation with Q-learning

Resource allocation problems have many practical applications that are difficult to solve, even in deterministic cases. It is known, for example, that both JSP and TSP are strongly NP-hard, moreover, they do not have any good polynomial time approximation algorithm. Additionally, uncertainties are faced in "real world" problems, e.g., the processing times of the tasks or tardiness of the order are not known exactly in advance. Stochastic RAP is a challenging area for reinforcement learning algorithms that only cope with discrete data are inappropriate. In this chapter, Section 3.1 provides the information about the manufacturing system used in this thesis. Section 3.2 provides the overall software architecture to make learning work on real manufacturing system. Section 3.3 states the problems faced when incomplete data is not reliable and difficult to compute in large-scale problem. Hence Section 3.4 discusses the reformulation framework for RAP to generalized MDP. Section 3.5 develops the basic behaviour of capacity allocation in the context of manufacturing application. Section 3.6 describes the fuzzy logic approach to Q-learning algorithm. Section 3.7 presents the development of fuzzy logic on a behaviour. Section 3.8 illustrates the integration of fuzzy logic into Q-learning algorithm. Lastly, the additional development of the hybrid coordination of behaviours is presented in Section 3.9.

3.1 The Manufacturing System

Semiconductor Testing Accelerated Processing (STAP) is the manufacturing system software used in a semiconductor wafer testing plant that connects to real-time inputs and outputs. This software models the plant to make manufacturing decisions and allows the below activities to be performed:

- **Capacity analysis** – determining the impact that system changes (such as adding people, changing processes, and so on) have on the plant. The model determines how changes affect throughput, cycle time, and other measures of factory performance.
- **Capacity planning** - determining when orders will be completed and when to start new lots into the plant.
- **Shop floor control** - producing either dispatch designed rules or recommended next tasks based on the current work in process and status of the plant.

STAP is a generic software library written in C code that facilitates the simulation of manufacturing system and the real time control strategies. A Pentium process running on a UNIX operating system platform provides computing power. It employs the three-phase approach and provides facilities for modeling the physical structure, the route and the system and machine loading policy. STAP is running with the following inputs every seconds:

- product list
- machines
- machine efficiency
- limited capacity buffers
- automated material handler

- work in process
- cycle time
- routes
- due date
- start plan or orders

STAP advocates the separate development of the conceptually different views of the simulated system. This approach facilitates the modular program development, program readability and maintainability and the evaluation of different strategies (system-level control policy, dispatching control, etc) on the same system.

3.2 The Software Architecture

Owing to the complexity of the addressed problem, it is hard for optimization-based tools such as ILOG OPL to find the optimal solution in finite time. Figure 3.1 shows the overall architecture in the implementation of a single site testing plant. Each block represents a set of programming codes in order to accomplish its goal. Source data from factory and customer environments are captured from STAP which several universe databases are integrated in the practical wafer testing plant. The problem is reformulated to generate reactive solutions (states) that are applied in the behaviours. This proposed architecture also has the advantage to incorporate human interference in learning. A Q-learning with a fuzzy inference module is developed to realize the capacity allocation. While the Q-learning is responsible for arranging and selecting the sequence of orders, the fuzzy inference module conveys how resources are allocated to each order by processing time. This creates the opportunity in enhancing the system by introducing the coordination of different reactive behaviours. The later sections will describe the individual blocks in detailed.

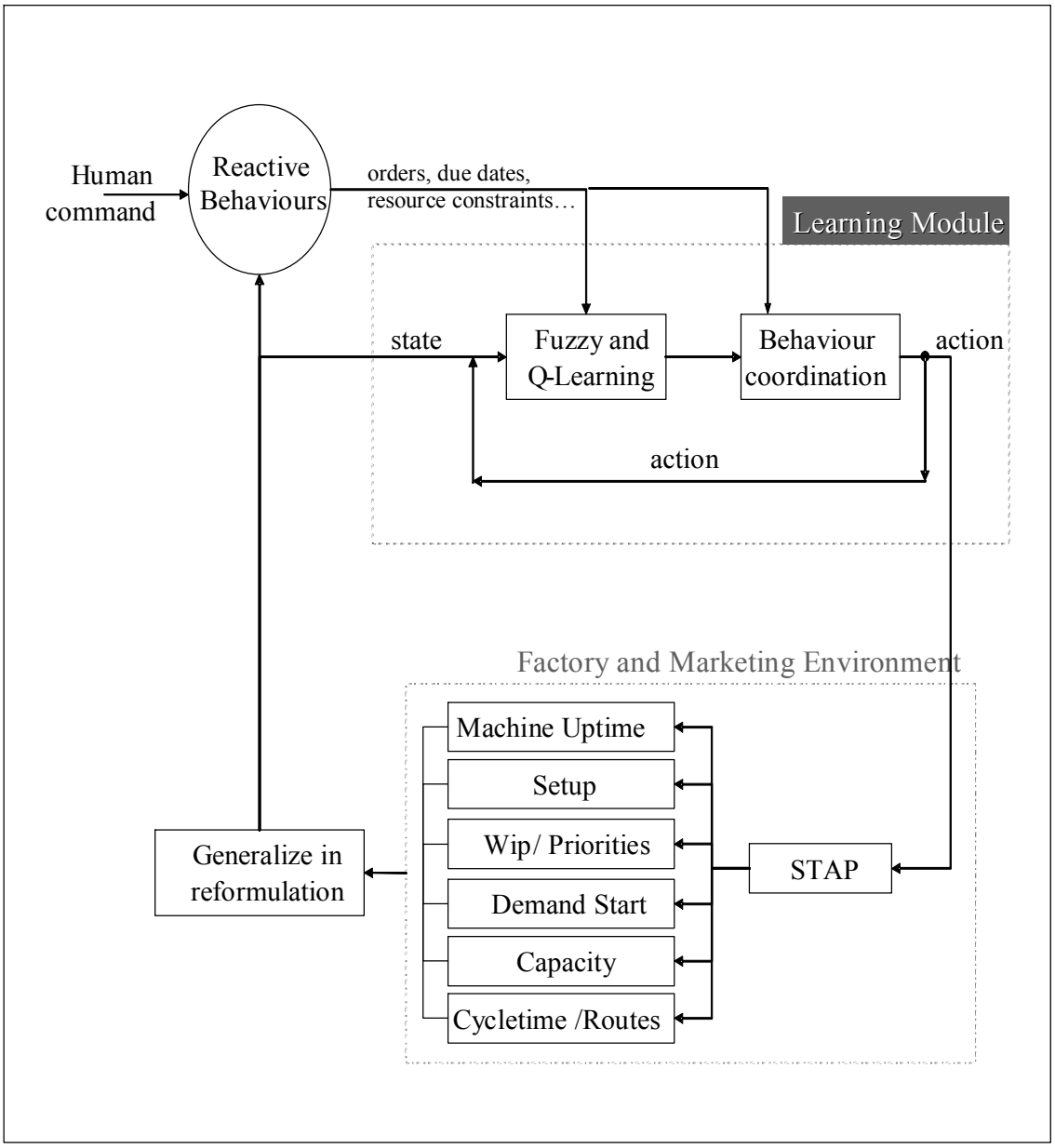


Figure 3.1: Overall software architecture with incorporation of learning module

3.3 Problems in Real World

3.3.1 Complex Computation

The idea of divide-and-conquer is widely used in artificial intelligence and recently it has appeared in the theory of dealing with large-scale MDPs. Hence partitioning a problem into several smaller sub problems is also often applied to decrease computational complexity in combinatorial optimization problems, for example, in scheduling theory. A simple and efficient partitioning method is proposed. In real world industry, the orders have release dates and due dates, and the performance measure, which involves total lateness and number of tardy orders. Assuming these measures are regular, the randomized functions defining the release and due dates are denoted as $\boldsymbol{A} : \boldsymbol{O} \rightarrow \boldsymbol{N}$ and $\boldsymbol{B} : \boldsymbol{O} \rightarrow \boldsymbol{N}$ respectively. In order to cluster the orders, we need the definition of weighted expected slack time which is given as follows:

$$S_W(v) = \sum w(s)\left[B(v) - A(v) - D(s,v)\right] \tag{3.1}$$

where $\boldsymbol{D}$ denotes the stochastic duration function with the set of resource states s in which order v can be processed and $w(s)$ are the corresponding weights.

The aim is to handle the most constrained orders first. In ideal case, the orders $\boldsymbol{O}_i$ are expected to have smaller slack times than orders in $\boldsymbol{O}_j$ if there are two clusters of $i < j$. During learning, orders in $\boldsymbol{O}_i$ are allocated to resources only. After some steps, the allocation policy relating to orders in $\boldsymbol{O}_i$ is fixed and starts sampling to achieve a good policy for orders in $\boldsymbol{O}_j$.

By clustering the orders, it will make too small clusters seriously decrease the performance of the best achievable policy and make too large clusters considerably slow down the system. This technique, however, has several advantages as below:

- it effectively decreases the search space,
- it further reduces the number of available actions in the states,
- it speeds up the learning as only a small part of the orders is allocated in a trial and which the variance of the total costs is decreased as a result.

3.3.2 Real-time Constraints

Since the STAP is required to interact with the real environment, the input data must be processed in real-time to acquire the suitable system's actions. Therefore multithreaded programming is required to process input data so as to avoid the severely degraded overall performance if the sequential program performs a blocking call. In a sequential program, the code is executed one instruction after the next in a monolithic fashion, with no regard to the many possible resources available to the program. For example, 'while loops' or 'for loops' and 'if-else' statements are used in the main ( ) function program extensively. Here, all input data will be threaded by linking the thread program with the pthread library through the header file pthread.h. The pseudo-codes below illustrate the simultaneous extraction of different sources of input data.

```
typedef struct data
{
      double splan, sdd;
}S_DATA;
```

```
void PLAN( ) //get 12 months plan
{
      s= getPLAN( );
      S_DATA.splan = s;
}

void DUEDATE( ) //get due dates requirement from customers
{
      s= getDUEDATE( );
      S_DATA.sdd = s;
}

int main(void)
{
pthread_t hThread1, hThread2;
pthread_create(&hThread1, NULL, (void *)PLAN, NULL);
pthread_create(&hThread2, NULL, (void *)DUEDATE, NULL);

      pthread_join(hThread1, NULL);
      pthread_join(hThread2, NULL);
      pthread_exit(0);
      ……
      return 0;
}
```

3.4 Reactive RAP Reformulation

Before learning is incorporated into a typical RAP to compute an efficient control policy, it is required to reformulate reactive RAPs as special MDPs. In order to reformulate the problem in stochastic framework as presented in Section 2.3, the

following terms are defined: the state space (including both initial and terminal states), the action space, the action constraint function and the effects of actions.

3.4.1 State Space, x

$x \in \boldsymbol{X}$ is defined as a 4-tuple (τ, μ, ξ, φ), where:

$\tau \in \boldsymbol{N}$ *is the current time.*

$\mu : \boldsymbol{R} \rightarrow \boldsymbol{S}$ *determines the current states of the resources.*

$\xi : \boldsymbol{R} \times \boldsymbol{N}_{\tau\text{-}1} \mapsto \boldsymbol{O}$ *stores the resources and the start times in which a task was begun.*

$\varphi : \boldsymbol{R} \times \boldsymbol{N}_{\tau\text{-}1} \mapsto \boldsymbol{N}_{\tau}$ *(where* $\mathrm{N}\tau = \{0,...,\tau\}$*) describes the stop times of the completed tasks.*

The resource allocation process starts from an initial state $(0, \mu, 0, 0)$ which corresponds to the situation at time zero and no task has been started. The initial probability distribution of the problem, x_0, is calculated as follows:

$$x_0(0, \mu) = P[\mu(r_1) = I(r_1), ..., \mu(r_n) = I(r_n)] \tag{3.2}$$

where r is the initial state of resource and n is the number of resources. Hence, x_0 provides initial states to resources according to probability distribution, $\boldsymbol{I}$.

A state x is considered as a terminal state if and only if $\boldsymbol{T}_F(x) = \boldsymbol{T}$ and it can be reached from a state x_1, where $\boldsymbol{T}_F(x_1) \neq \boldsymbol{T}$. If the system reaches a terminal state, i.e. all tasks are finished, and then the control process is treated to be completed. All terminal states can aggregate all terminal states to a global unique terminal state and introduce a new unique initial state, x_0, that has only one available action which takes randomly to the

real initial states. Then, the problem becomes a stochastic to find a routing having minimal expected cost from the new initial state to the goal state.

3.4.2 Action Space and Constraint Function

When the system is informed on the finished tasks every time, it can decide on the tasks to apply by which resources. The action space contains operation-resource assignments $a_{vr} \in \boldsymbol{A}$, where $v \in \boldsymbol{O}$ and $r \in \boldsymbol{R}$, and a special a_{wait} action when the system does not start a new task at the current time. Action a_{wait} is available in every state with an incomplete operation. An action available in states where resource is idle, it needs to process operation which is not executed earlier and its precedence constraints are satisfied. The effect of a_{wait} action is an incomplete operation will start at t on resource finishes with probability that the duration of the operation is executed by a resource which has state. This effect is named completion rate in stochastic scheduling theory or hazard rate in reliability theory.

3.4.3 Features of Reactive RAP Reformulation

From the reformulation, these MDPs have finite actions as $|\boldsymbol{A}| \leq |\boldsymbol{R}||\boldsymbol{O}| + 1$ always holds. It was observed that the MDPs are aperiodic by means of none of the states can appear more than one time. This is because during the resource allocation process τ, the quantity τ increases each time as the state changes. Therefore, the system cannot reach the same state twice. The state space of the reformulated RAP becomes finite when the allowed number of non-task operations is confined and the random variables of the duration are finite. In order to achieve the effective computation of a good control policy, it is important to try to reduce the number of states. This is done by

recognizing the performance measure reward is increasing in completing the order in time when the states in which no order has been completed can be present, except the initial states. Therefore, each a_{wait} action may result to a state where an order has been completed. By this way, the state space can be reduced and, therefore, a good control policy can be calculated more effectively.

3.5 Resource Allocation Task

The identified manufacturing production control task in this research is simple resource allocation. A few basic aspects are considered in production planning decisions. The first aspect is to allocate the machine capacity efficiently that results in high manufacturing performance, especially in complicated operations of the semiconductor wafer test industry. The second aspect is to handle and develop products in the shortest time to win the market share and compete with the competitors. The last aspect is to allocate and manage the headcount to machines in a single factory site and outputs to the quantity and quality from customers. Hence, the behaviours were: the "capacity-planning" behaviour, the "product development" behaviour and the "headcount assignment" behaviour. It is much simpler to design and program each behaviour independently. This style of programming is the essence of coordinating task-achieving behaviours in behaviour-based learning. The main feature of the "capacity-planning" behaviour and the development were described in the thesis.

"Capacity-Planning" Behaviour

As all the implementations are done in a multi-resource, multi-period, and soft-due-date problem with high complexity, this program is designed to arrange, select the

sequence of orders and obtain the capacity to be allocated to each order. Assume that a make-to-order manufacturing firm has customer orders. For each order, there is an order value (revenue) and the customer's desired due date. There are m manufacturing resources in the system. A horizon of periods is used for planning and usage of machines and independent of manpower without overtime. Here, the lead time for each testing is fixed and known and independent of the saturation. Thus, given a completion time for each order, the requirement of resource for the order in a time period can be calculated from the bill of materials (BOM) and the production standards. Moreover, given the customer-order due dates, the vector of manufacturer's preferred due dates can be calculated. This seeks to minimize the total weighted earliness and tardiness penalties for all orders, subject to resource constraints.

3.6 Q-learning Algorithm

In one-step Q-learning, real world experience takes the place of the dynamic model - the expected values of actions in states are updated as actions are executed and the responses can be measured. Let ***Q*(s,a)*** be the expected value of the discounted reinforcement of taking action *a* in state *s*. The value of this quantity can be estimated recursively with the following formula:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \max_{a'} Q^*(s',a') \tag{3.3}$$

In equation (3.3), the first term of ***R(s,a)*** is the immediate reward of taking action a in state s and the second term is the future discounted rewards. The transition function ***T(s,a,s')*** denotes the probability of making the transition from *s* to *s'* by action *a*. The γ is the discount factor *(0 ≤ γ < 1)*.

The policy follows directly from ***Q**** by selecting the optimal (or greedy) action in each state:

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{3.4}$$

where arg $\max_a$ denotes the value of a which maximizes the expression. In other words, the best policy is, in each state, to take the action with the largest Q-value. As the Q-function makes the actions explicit, this allows the computation of the Q-values on-line using the Q-learning update rule in the Q-table:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{3.5}$$

In equation (3.5), α is the learning rate (or step size) between 0 and 1 that controls convergence; and r is the reward value. High values of α will result in drastic updates of Q-values on each step whereas low values mean more conservative changes. It has been shown (Watkins, 1989) that if each action is executed in each state an infinite number of times and α is decreased appropriately, the Q-values will converge to ***Q**** with probability 1, and the optimal policy now follows from equation (3.4).

True convergence to optimal action-values is rarely achievable. In practical use the goal is that the action-values will describe an acceptable controller in reasonable time. Q-Learning represents the values of all actions in all states, rather than only representing the policy. The knowledge is expensive in terms of the amount of information that has to be stored; however, it increases the learning system's flexibility by supporting off-policy learning.

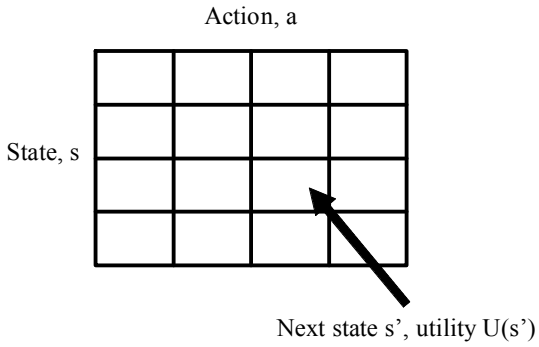


Figure 3.2: Q-table updating Q-value

Summarizing (in Figure 3.2), Q-learning basically involves constructing a Q-function: Q(state, action) $\rightarrow$ utility, that predicts the utility for each state-action pair. Table 3.1 illustrates the key characteristics of Q-learning algorithm. The factors in the table can also be applied to other RL algorithms for comparison.

Table 3.1: Key characteristics of Q-learning algorithm

Characteristics	Q-learning
Parameters	α and γ
Updates per step	1
Model-based	No
Iteration	Similar to value iteration
Policy type	Off-policy
Policy class	Unknown, Sequential
Model consists of value function	**Q**(s,a)
Other data-structures	-

3.7 Limitations of Q-Learning

This section discusses some of the difficulties that must be overcome in order to implement a useful learning system on a real problem. Proposed approaches or techniques will be introduced and implemented in a later section.

3.7.1 Continuous States and Actions

In the implementation of Q-learning, the Q-values are stored in a look-up table; continuous variables are discretised to become an index into the table. Discretising the state of the world is a common approach in artificial intelligence research. However, many quantities are inherently continuous with a fine resolution that leads to many discrete states. Even if the state can be discretized meaningfully, it might not be readily apparent how best to do it for a given task. Incorrect discretizations can limit the final formation of the learned policy, making it impossible to learn the optimal policy. If the state is coarsely discretised, there is a risk of having total states that do not belong together. If the state is finely discretised, an unmanageable huge state or action space will result. The problem with large numbers of states occurs when we are dealing with several continuous sensors. For example, consider a product that results in 24 final yields, each returning a value in the range of 70% to 90%. If we discretize each yield into two states, low and high, threshold it at some appropriate value (e.g. historical data 61.1%) that still ends up with 224 states. Even with such a coarse discretization, this is an unmanageable number of states.

Similarly with the interaction with the real world requires smoothly variable actions as well as continuous state representations. Learning systems that discretise actions suffer in similar ways to those that discretise states: high memory use and slow learning due

to inability to generalise. Continuous actions give more accurate and predictable movement, reduce wear, and improve energy consumption.

The learning method should be able to cope with continuous state and action spaces without needing to discretize them. Therefore there is a need for an approach to extend the Q-learning framework to continuous state and action spaces. We will discuss a Fuzzy approach in the Section 3.8.

3.7.2 Slow to Propagate Values

We are interested in learning on-line, while the system is operating in the real world. With increase in computing power, the amount of computation that we can apply to learning is limited. This is especially important when we are using the learned control policy as controller. We must be able to make action choices at an appropriate rate to allow us to function at mild risk in the world. Although this time constraint does not seem to be a great restriction, it prevents the use of learning algorithms that take a long time to learn. A related issue is the total amount of time that it takes to learn. Therefore the computing power of the system must be sufficient for the learning algorithm to speed up the optimization process in large-scale problems. Distributed optimization techniques to speed up the computation that result in more robust and flexible solutions will be discussed in the next chapter.

3.7.3 Lack of Initial Knowledge

Many learning systems are trying to learn with no initial knowledge. However it introduces problems when working with the real world. Initially, if the learning system

knows nothing about the environment, it is forced to act in an arbitrarily way. This is not a real problem in simulation, where the arbitrary restarts are possible, and many simulated experiences can be gathered cheaply. However, when controlling a "life" system it can prove to be false. Since each action is taken in decision, a wrong choice can cause it to incur losses at a huge expense. This can speed up the low productivity of the company from poor planning. In order for the learning system to be effective, there is a need to provide some form of bias, to give it some idea of how to act initially and how to begin to make progress towards the goal. Adding insufficient or incorrect bias might cause the learning system to fail. Hence the programmers should have some sense of judgment and appropriate assumptions about the system and its environment in supplying bias based on parametric historical data.

3.8 Fuzzy Approach to Continuous States and Actions

Fuzzy controllers map continuous state into membership of discrete classes (fuzzification), then pass the membership through rules producing actions, and finally combine the actions based on the strength of their membership (defuzzification). Fuzzy logic is a tool for creating a continuous state and action (Andrecut and Ali, 2002) to compute a new good approximation of the estimated value function.

Fuzzy Logic

Fuzzy logic is one of the most fundamental and influential computational intelligence tools (Chung and Lin, 2000) in some aspects of learning. Fuzzy logic allows variables to take on values determined by how much they belong to a particular fuzzy set defined by a membership function. A fuzzy logic control system (Figure 3.3) consists of four major parts:

- Fuzzifier: This module converts the received crisp sensor readings into fuzzy input sets, in order to be processed by the inference module.
- Fuzzy rule base: This module contains a collection of IF-THEN rules, providing the necessary connection between input and output fuzzy sets.
- Fuzzy inference engine: This module generates a fuzzy output by evaluating the fuzzy rules with the fuzzy input membership functions. This is accomplished by applying fuzzy operators to the fuzzy sets and fuzzy inference methods to the rule base.
- Defuzzifier: This module converts the generated set of fuzzy output sets into a set of crisp executor outputs.

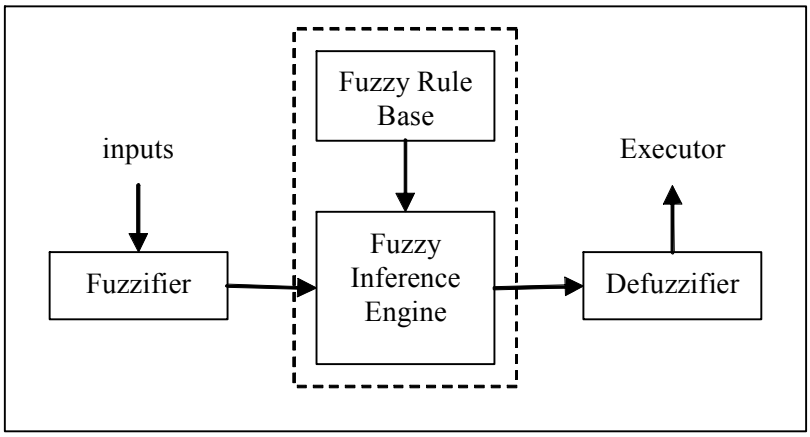


Figure 3.3: Fuzzy logic control system architecture

In the real-world, the derivation of rule base is a step of major significance to the performance of fuzzy logic control. By learning rules, fuzzy logic will not be rigid as compared to the rules constructed from human. After the learning phase, the created fuzzy rules will map input set to output command set in the following form:

IF *(a set of conditions is fulfilled)* THEN *(a set of outputs can be determined)*

Finally, learning in fuzzy logic control system is implemented such that it has the capability of rule creation, adaptation (parameter modification of THEN part of rule), and generalization (modification of IF part).

Fuzzy behaviour coordination is a way to select behaviours and decide what the motion actuators shall receive. The combination of the control actions of the behaviours allows smooth transitions between behaviours by taking into account the effector and the relative importance of each behaviour. As there is no information about the outcome of each action, some internal state of the agent is also considered in the decision process.

Here defuzzification plays an important process in behaviour coordination. This process is used in two different ways, as formulated below:

- By Weight

$$M^j = \frac{\sum_{i=1}^{N} B_i \times A_i^j}{\sum_{i=1}^{N} B_i} \tag{3.6}$$

- By Max

$$\begin{aligned} M^j &= A_i^j \\ B_I &= \max_{i=1\ldots N} B_i \end{aligned} \tag{3.7}$$

where N is the number of behaviours, M^j is the j-th order cluster, B_i is the value of the i-th behaviour state variable and A_i^j is the i-th behaviour action for M^j. Equation (3.6) considers that every behaviour is important and must be considered in the effector final

value. Equation (3.7) gives a total importance to the behaviour showing the maximum activity.

3.9 Fuzzy Logic and Q-learning

Figure 3.4 shows how fuzzy logic facilitates Q-learning by providing continuous states and actions on "capacity-planning" behaviour, with respect to STAP. This fuzzy module is to resolve the capacity conflicts that arise when two or more orders require the same kind of machine in the same time period. In addition, the Q-learning algorithm as discussed in Section 3.7 only handles discrete variables. However, it is difficult to implement Q-learning algorithm in real-time environment which works with continuous states and actions in large-scale RAP. Fuzzy logic is proposed to solve the problem. The following subsections present the step-by-step development of fuzzy logic controller for "capacity-planning" behaviour.

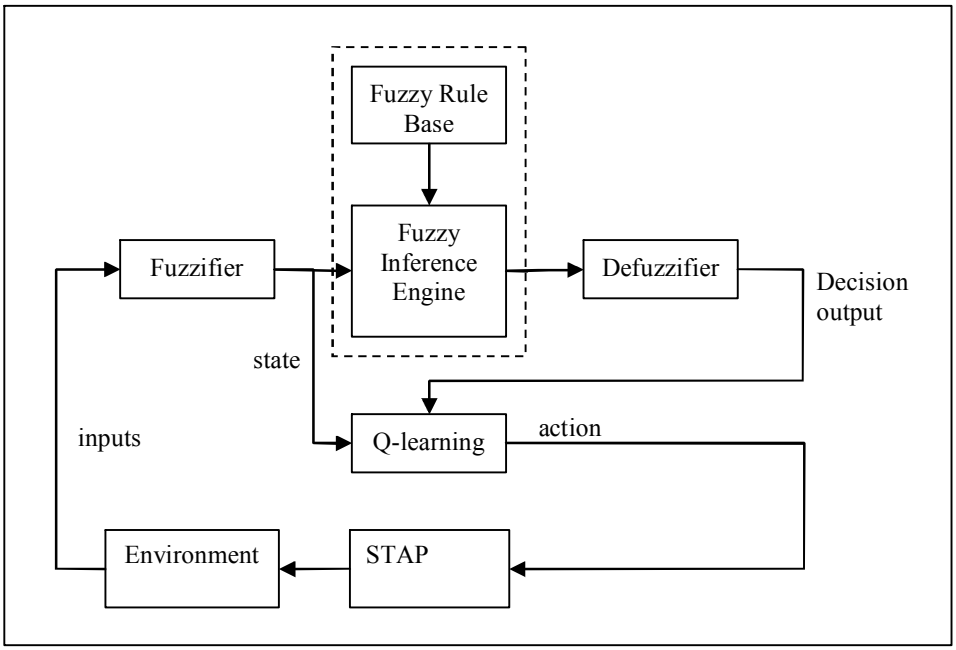


Figure 3.4: Fuzzy logic integrated to Q-learning

3.9.1 Input Linguistic Variables

Once the priority sequence of orders is known, the next step is to assign their completion dates according to the priority sequence and subject to the availability of capacity. Assume the known priority sequence are $\boldsymbol{X} = [x_1, x_2, \ldots, x_n]$. Here, the higher priority orders is arranged to be completed by their due dates. Due to the limited capacity availability, the lower priority orders have to be arranged for completion later or earlier than their due dates. According to the input of fuzzy factors, one can decide capacity allocation by computing $fF_1, fF_2, fF_3, fF_4, fF_5, fF_6$ of all unallocated orders as illustrated below.

fF_1 : The resource availability just in-time to due date.

$fF_1 = \min\{fF_1^1, fF_1^2 < 1, \ldots, fF_1^n\}$ is a fuzzy number with membership degree $f_{F_1^n}$ that defined as:

$$f_{F_1^n} = \begin{cases} 1 & \text{if } x > 1+\varphi \\ (x-1)/\varphi & \text{if } 1 \le x \le 1+\varphi, \text{ where } \varphi \text{ is set to 0.15} \\ 0 & \text{if } x < 1 \end{cases} \quad (3.8)$$

where x is the factor of availability of machine for order to the machine capacity in completing the required order's production cycle time just in time. φ is the relaxation coefficient which 15% of the machine capacity is not loaded with orders based on 6 months historical data.

fF_2 : The resource availability later than due date

fF_3 : The resource availability earlier than due date

Fuzzy factors 2 and 3 are represented by fuzzy numbers with membership degree $f_{F_2^n}$ and $f_{F_3^n}$ respectively, which is the same membership function as fF_1 but the replacement of machine availability amount either later or earlier than due date respectively.

fF_4 : The relative resource availability earlier than to later than due date

Fuzzy factor 4 is the ratio of Fuzzy factors 3 and 2 with membership function $f_{F_4^n}$ that defined as:

$$f_{F_4^n} = \begin{cases} 1 & \text{if } x > 1+\varphi \\ (x-1+\varphi)/2\varphi & \text{if } 1-\varphi \le x \le 1+\varphi, \text{ where } \varphi \text{ is set to } 0.15 \\ 0 & \text{if } x < 1-\varphi \end{cases} \quad (3.9)$$

where $x = \dfrac{fF_3}{fF_2}$.

fF_5 : The ratio of minimum possible earliness to minimum possible order tardiness

The membership function for Fuzzy factor 5 is defined as:

$$f_{F_5^n} = \begin{cases} 1 & \text{if } x > 1+\varphi \\ (x-1+\varphi)/2\varphi & \text{if } 1-\varphi \le x \le 1+\varphi, \text{ where } \varphi \text{ is set to } 0.15 \\ 0 & \text{if } x < 1-\varphi \end{cases} \quad (3.10)$$

where

$x = \dfrac{t_e}{t_l}$,

$\text{resource of earliness}, t_e(i) = \min\{t_e(1), t_e(2), .. t_e(\text{n})\} \text{ for } i = 1$

$\text{resource of lateness}, t_l(i) = \min\{t_l(1), t_l(2), .. t_l(\text{n})\} \text{ for } i = 1$

fF_6 : The relative weight of order to the total weight of all previously allocated orders

The membership function for Fuzzy factor 6 is defined as:

$$f_{F_6^n} = \begin{cases} 1 & \text{if } x > 1+\varphi \\ (x-1+\varphi)/2\varphi & \text{if } 1-\varphi \le x \le 1+\varphi, \text{ where } \varphi \text{ is set to } 0.15 \\ 0 & \text{if } x < 1-\varphi \end{cases} \quad (3.11)$$

where $x = \dfrac{w_n}{\sum_{i=1}^{m-1} w_{x_i}}$, w is the weight of order cluster and where m is the length of the production cycle of order n.

3.9.2 Fuzzy Logic Inference

This study develops three decisions, $\boldsymbol{D_1}$, $\boldsymbol{D_2}$ and $\boldsymbol{D_3}$. $\boldsymbol{D_1}$ refers to the decision of arranging an order to be completed exactly at its customer due date. The capacity it requires is allocated backward (from the due date) first and forward if necessary. $\boldsymbol{D_2}$ arranges an order to be completed one day earlier than the due date. The way to allocate the required capacity is the same as that for $\boldsymbol{D_1}$. $\boldsymbol{D_3}$ arranges an order to be completed one day later than the due date. The way to allocate the required capacity is the same as that for $\boldsymbol{D_1}$ as well. Here we define six fuzzy degrees of decisions, fD_1, fD_2, fD_3.

Although the above six input factors can be evaluated exactly with heavy computation, the factors with fuzzy numbers can be approximated to choose among decisions. The traditional rule-based decision method is widely used for problems with only a few fuzzy factors. However, for this problem, six fuzzy input factors are introduced. Even if each fuzzy factor assumes only three fuzzy values, e.g., small, medium, and large, the total number of possible combinations will be $3^6 = 729$. Thus, a traditional rule-

based decision approach will be inefficient. Hence, the following fuzzy logic-based decision method is proposed. Let fD_{n1} be the membership degree of fuzzy decision $\boldsymbol{D}_n$ where n = 1,2,3.

$$fD_1 = fF_1,$$
$$fD_2 = f_{F_1}^c \otimes f_{F_2} \otimes f_{F_3}^c \otimes f_{F_4} \otimes f_{F_5}^c \otimes f_{F_6},$$
$$fD_3 = f_{F_1}^c \otimes f_{F_2}^c \otimes f_{F_3}^c \otimes f_{F_4}^c \otimes f_{F_5}^c \otimes f_{F_6},$$

where $f^c = 1 - f$ and "⊗" denotes the product of f_1 and f_2. The best decision n is selected if n = $\arg\max[fD_1, fD_2, fD_3]$ with the given fD_1, fD_2, fD_3. The above fuzzy logic operations can be interpreted as follows. The choice of $\boldsymbol{D}_1$ is made only according to the resource availability at order due date. $\boldsymbol{D}_2$ is chosen if there are not enough resources at due date, but resources are available earlier than due date where the minimum earliness is less than the minimum order tardiness.

3.9.3 Incorporating Q-learning

In this section, we will describe in more details, and include actions, state variables and reward function of Q-learning. The RL agent has to decide between two actions: entering a product or doing nothing. The decision is based on the output of the fuzzy decision, which is illustrated in the previous section. The activation function for the 4 output units is a sigmoid function translated by -1 to fit in the reward region. One of the most important decisions when designing an RL agent is the representation of the state. In the system this is one of the major concerns since a complete representation is not possible due to the complexity of the problem. Therefore we choose to include the following information in the state representation:

- state of machines. Each machine may be found in one of the four distinct states: idle, running, blocked for engineering qualification, setup.

- state of input buffers on limited capacity.
- backlogs for each products of order.

The estimated total cumulative lateness and number of orders (O) in the buffer are adopted as the state determination criterion in the policy table (Table 3.2). This value was chosen over order tardiness penalty since it is able to distinguish between orders that are completed earlier. . For a given number of states, the range for each state is defined as a ratio fF_5 of the expected mean processing time (EMPtime). As fF_5 decreases, the system is able to distinguish the differences between orders at the lower end of the lateness spectrum with orders that are very late and being grouped together in the last interval. 1.8 times is a world-wide standard defined by semiconductor industry on wafer testing cycle time.

Table 3.2: State Variables

State	State criteria
1	O>1 and cumulative lateness < 0
2	O>1 and 0 =< cumulative lateness < $fF_5 \otimes$ EMPtime
3	O>1 and EMPtime =< cumulative lateness < $1.8 \times fF_5 \otimes$ EMPtime
4	O>1 and $1.8 \times fF_5 \otimes$ EMPtime =< cumulative lateness
5	O = = 1
6	O = = 0

So there are 3^6 combinations of states the system can identify at 3 different types of resources. For programming purposes, each state will be encoded with a value: 0 or 1 for order availability. With 729 states and four actions, the Q-table will be very large, with a grand total of 2916 Q-values. The actual Q-Learning algorithm does not need to know the details of the state or action; it just needs an identification number that

represents a state or action. For example, if it knows it is in state 4, it might decide to try action 1 to launch the product and judge how good it was allocated to the resources. Q-Learning values are built on a reward scheme. There is a need to design a reward function that will motivate the system to perform the specific behavior in which higher rewards should correspond to better allocating of resources to orders. The reward assignment for "capacity-planning" behaviour is stated in the following Table 3.3. The reward function was developed based on the principle that the later an order is completed, the larger its negative reward; likewise the earlier a job is completed the larger its positive reward. As an example, when an order is finished, the lateness of this order (L) is compared to the maximum lateness (ML) of previously completed orders in terms of days. If the lateness of the order completed is greater than the maximum lateness value, the learning agent receives a reward of -10. Otherwise, the learning agent receives a reward of +10.

Table 3.3: Reward Function

Reward function	Reward
L > ML	-10
L is flagged	-10
L >= 7	-10
L < -7	10
otherwise	10

In order to implement this Q-learning algorithm, all allocations must be segregated into steps in each episode. Each step consists of reading the percepts, choosing an action, and evaluating how well the action performs. All Q-values will presumably be equal to zero for the first step, but during the next step (when the algorithm is invoked), it will set a Q-value for $\boldsymbol{Q}(s,a)$ that will be a product of the reward it received for the last action. So as backlogs come in, the Q-values are calculated repeatedly,

gradually becoming more refined (and more accurate). The pseudo-code for Q-learning algorithm in programming implementation is given as follows:

```
Initialise Q(s,a)
Repeat (for each episode):
      Initialise s
      Repeat (for each step of episode):
            s = getInputs ( )
            a = chooseAction (s, Q) // using greedy policy
            apply (a)               // take action
            observe r, s'     // observe reward, r, and new state, s'
            Q(s,a) = Q(s,a) + α[r + γmax_α'Q(s',a') − Q(s,a)]
// update Q-value for the state using the observed reward and // the
maximum reward possible for the next state
            s = s'
      until s is terminal
```

The parameters used in the Q-value update process are:

- γ, discount factor, set between 0 and 1. This models the fact that future rewards are worth less than immediate rewards. As γ approaches zero, the agent is more biased because it takes immediate reward into account more strongly. When γ approaches 1, the agent will be more intuitive reducing the impact that the recent results have on the learned policy.
- α, learning rate, set between 0 and 1. Setting it to 0 means that the Q-values are never updated, hence nothing is learned. Setting a high value such as 0.9 means that learning can occur quickly.
- $max_{\alpha'}$, the maximum reward that is attainable in the state following the current one, i.e. the reward for taking the optimal action thereafter.

3.10 Behaviour Coordination System

This is an additional work done for this research. In order to facilitate semi-autonomous control of the system, the human user is given as much control as possible within a safety stock limit. The command from the user is considered as any other behaviour in the system and is incorporated with the overall behaviour as shown in Figure 3.5.

With the incorporation of different behaviours, a key issue is to efficiently coordinate among these behaviours to achieve a good allocation performance as well as semi-autonomous control. For behaviours' coordination, this work adopted a hybrid approach based on subsumption and motor-schema. In addition, fuzzy logic is used in conjunction with this hybrid approach to tune the coordination process. In this approach, individual behaviour is weighted appropriately based on inputs. Then, according to the response of environmental stimuli, fuzzy rules are applied appropriately to decide which behaviours to trigger. Finally, the defuzzification module converts the fuzzy output to generate the decision command based on the following formula.

$$R_{pt} = \frac{\sum_{i=1}^{n} bw_i \times o_i^t}{\sum_{i=1}^{n} bw_i} \tag{3.12}$$

In equation (3.12), *bw* is the behaviour weight and *n* represents the behaviours (see Figure 3.5) triggered by the fuzzy rules with i-th behaviour state variable. *o* is the

output decision fused to produce a single vector R_{pt} so as to be applied to the product to be launched.

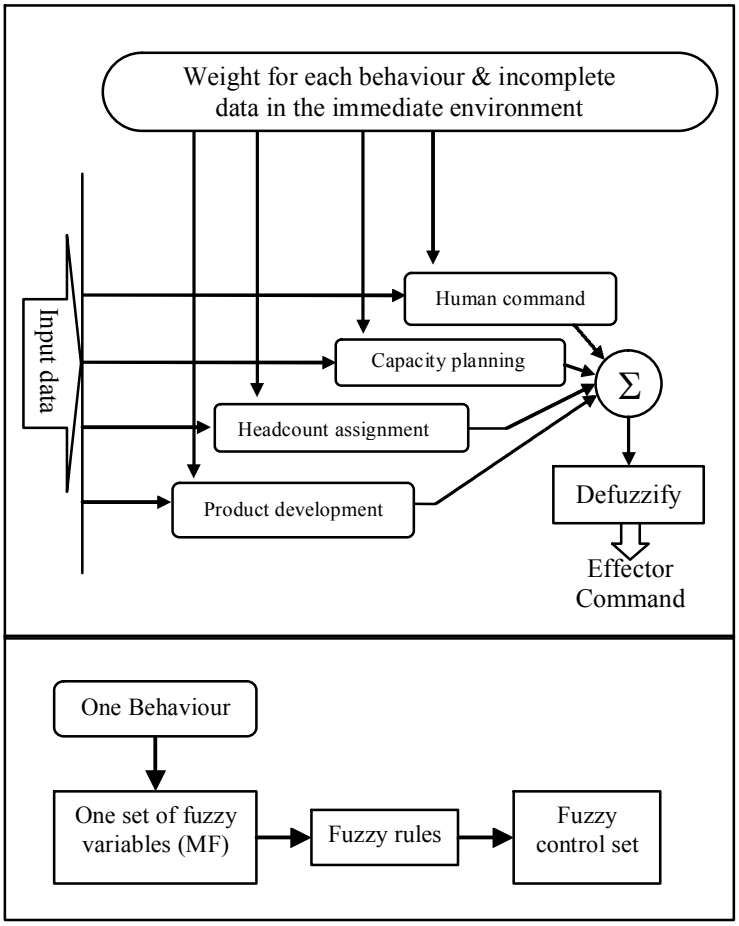


Figure 3.5: Behavioural Fuzzy Logic Controller

3.11 Summary

Reinforcement learning involves finding a policy that will maximize rewards over time. Q-Learning algorithm is a model-free approach to solve reinforcement learning problems that is capable of learning off-policy. Since Q-learning algorithm is applied in real environment and is a discrete formulation, fuzzy logic is proposed to provide continuous states and actions to Q-learning algorithm. This chapter has identified Q-learning algorithm to be suitable for learning purpose. Implementation of such an

algorithm (Wang and Usher, 2005) for reformulated stochastic RAP will be presented in the next chapter. Q-learning algorithm requires strict timing for real world operation. Therefore, synchronization with real-time information must be maintained and that is done by multithreaded programming. The architecture proposed was developed incrementally; initially familiarization of each input's working principles is necessary for implementation. All implementations are done on real factory manufacturing system that is ready for testing.

Chapter 4

Experiments and Results

This chapter presents the experiments conducted to test the learning of "capacity-planning" behaviour. Section 4.1 describes the overall experimental setup and the experiments required to be carried out for the evaluation of learning. Sections 4.2 compares the different parameter settings in Q-learning algorithm and the optimal parameters were found. Section 4.3 shows the final results of learning behaviour and test concerning convergence of the algorithm. Section 4.4 provides a simple test on the feasibility of the adaptation to changing environment.

4.1 Experiments

The experiments are divided into three main aspects and discussed in the next sections:

- Experiment A – to find optimal Q-learning parameters,
- Experiment B – to learning of "capacity-planning" behaviour,
- Experiment C – to test in changing environment.

4.1.1 Testing Environment

All experiments were conducted in a semiconductor wafer testing factory. A semiconductor wafer testing plant commonly has a variety of testers (example in Figure 4.1). A tester consists of a computer system and at least one test head. The computer

system is loaded with an automatic testing program, and the test head is loaded with the wafer that needs to be tested. A prober is used to load or unload a wafer on a test head. A loadboard which composes of probe pins for testing the wafer circuits, is installed in the test head in order to perform the tests on the wafer. For each wafer product, a different loadboard and a prober are needed. The setup of test head involves changes in hardware, temperature testing, or testing recipe.

The experiment contains three testers, T1, T2 and T3, sharing a common buffer of wafers to be tested, and two types of probers, P1 and P2, each with four sets. There are 2000 active products to be tested. Each order requires one tester and one prober to process in a certain period of time, measured in seconds. Orders, the processing requirements, and the tardiness of the products are illustrated in Appendix A (400 examples).

Figure 4.1: Example of a Tester

4.1.2 Measure of Performance

There are two classes of solutions in analyzing the behaviour – qualitative or quantitative. In qualitative analysis, the human observer is required to see and judge the manufacturing system all the time and it can be very subjective. This will be discussed in the later section. We can also make full use of the data logged when the experiments are running to perform a quantitative measurement, which is more practical. In reinforcement learning, the reward is a best tangible measure especially in comparing the effect of different parameters or learning techniques. It can be used to tell if the learning converged to a high reward-giving behaviour and show the speed of convergence in time or step domain. In this context, we will use reward parameter for comparison.

4.2 Experiment A – Comparing Q-learning Parameters

The "capacity-planning" behaviour was learnt using Q-learning algorithm, maintaining the goal of minimizing the number of late jobs and total cumulative lateness; i.e. minimizing the processing resources in testers of product-type dependent setup times. The algorithm determined the manufacturing system to optimal slack ratio using randomly generated orders (jobs) with random due dates on the tasks and the process-plans of the jobs that covered real products; as well as, the resources covered real machine types. The next subsections will describe the state variables, the reward function and the parameters of the Q-learning algorithm.

4.2.1 Experiment A1: Reward Function

The objective of this experiment is to find a deterministic reward function which relates the state and action to a reward value.

Procedure

a) Set the following parameters:

- Learning rate: 0.2
- Discount factor: 0.95
- Exploration probability: 0.8
- Actions: start products, do nothing
- States: All combinations of inputs

b) Define the reward function of rewarding more to the system

c) Ensure that the order is at the original starting time. Compile and run the program.

d) Repeat step (b) if the present reward function does not achieve the goal of the behaviour.

Results

After the some trials, it was observed that the STAP indeed avoided lateness by punishing the machine for order in late process. Without this punishment, the STAP took several times for it to learn to manage the correct order. A penalty can be given to the system whenever it chooses the late due date action so that it can try another insertion. This was the final modification of the reward function in the following:

Table 4.1: Final reward function

Reward function	Reward
L(lateness of order) > ML(maximum lateness)	-10
L is flagged	-10
L >= 7	-10
4 =< L < 7	-5
0 =< L < 4	-1
-4 =< L < 0	1
-7 =< L < -4	5
L < -7	10
Otherwise	10

4.2.2 Experiment A2: State Variables

This experiment defines the state variables to be used in Q-learning by intuition and observation. From the configuration settings of the inputs as described in Chapter 4, the test is carried out to identify which states are activated in the testing environment.

Procedure

a) Set the following parameters:

- Learning rate: 0.2
- Discount factor: 0.95
- Exploration probability: 0.8
- Actions: start products, do nothing
- Reward Function: final reward function in Experiment A1
- States: All combinations of inputs

b) Ensure that the job is at the original starting time. Compile and run the program.

Results

Figure 4.2 shows the activation of the order when order exists or not. Y1 is the presence of order. Y2 and Y3 represent the activation of state criteria. Since some of the orders of presence and non-existence were not experienced in this environment, the states can become dummy states and the number of states is reduced from 729 to 81. This implies effective processing time in lesser states.

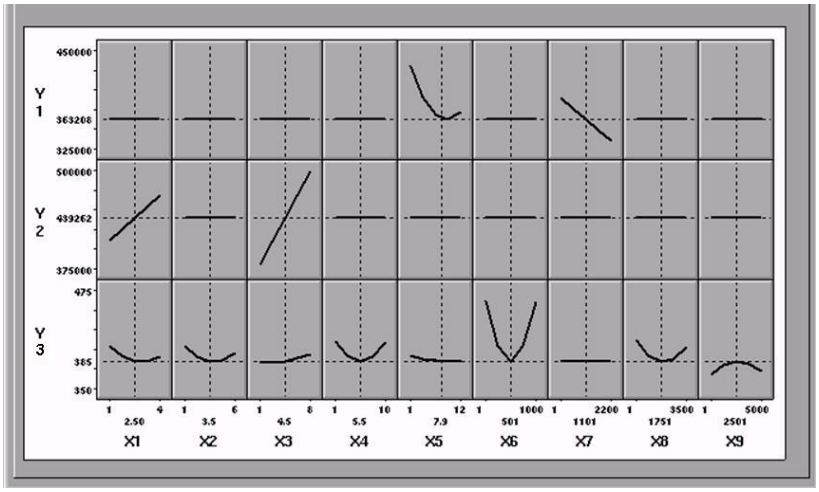


Figure 4.2: Orders activate different states

4.2.3 Experiment A3: Discount Factor

The objective of this experiment is to determine the discount factor. The effect of different settings of the discount factor, γ, will also be investigated in this experiment. Based on the Q-learning update rule, the discount factor describes future reward values. The discount factor is set to zero only if immediate reward is taken into account. Since the learning of a behaviour is a continuous task without a final state, a discount factor smaller than 1 was required. Mathematically in the case of γ=1.0, the algorithm may have

difficulties in converging. In Experiment 1, the discount factor was set to 0.95, a value commonly used. This value was chosen with exploring another setting (γ=0.5) in this experiment.

Procedure

a) Set the following parameters:
   - Learning rate: 0.2
   - Discount factor: 0.95, 0.5
   - Exploration probability: 0.8
   - Actions: start products, do nothing
   - Reward Function: final reward function in Experiment A1
   - States: final states in Experiment A2

b) Set the step number to 4000 as the learning behaviour was observed to be stabilized.

c) Set the discount factor to be 0.95.

d) Ensure that the job is at the original starting time. Compile and run the program

e) Repeat step (d) for 10 times.

f) Repeat steps (a) to (d) for discount factor of 0.5.

Results

The experimental data is shown in Figure 4.3. As seen from the graph, the behaviour was learnt equally well in the two cases. Of course, with γ=0.5 the numerical values in the Q-table were smaller than in the case of γ=0.95. As it was shown that reformulated RAPs are aperiodic and have ADP property, the discount

factor in this experimental setup is not crucial and both settings worked fine, the discount factor will be kept at 0.95 in following experiments.

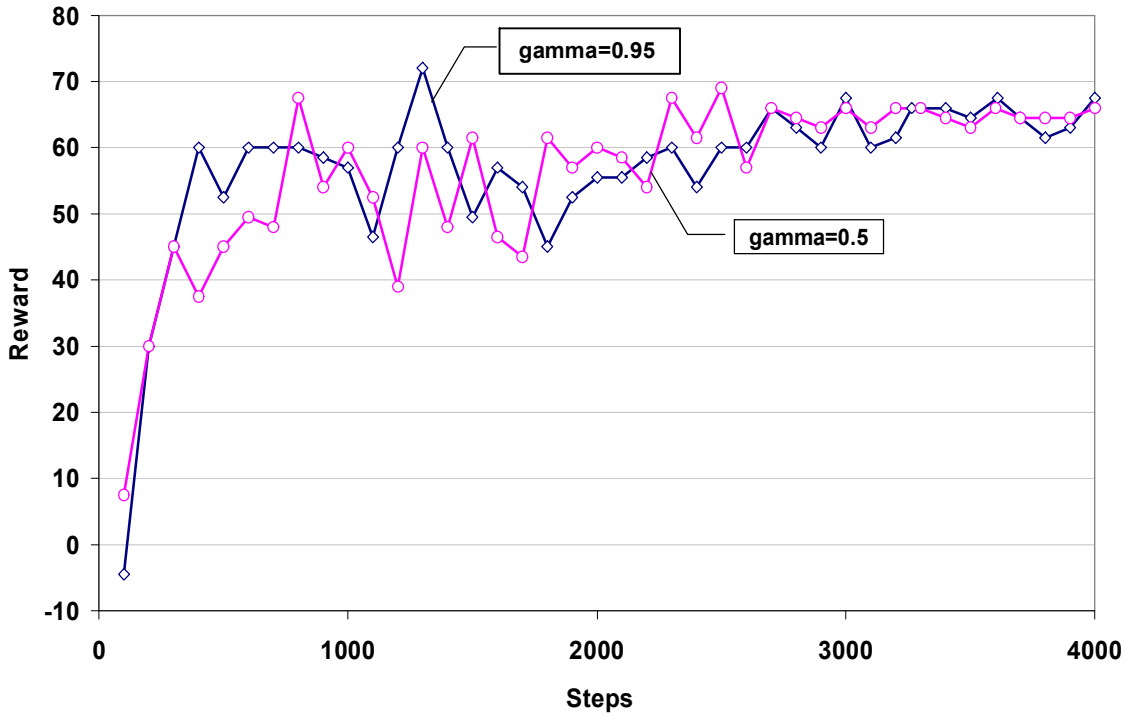


Figure 4.3: Different discount factors

4.2.4 Experiment A4: Exploration Probability

The objective of this experiment is to determine optimal exploration probability to speed up the convergence. The learning was performed with an ε-greedy policy. Most of the time, the action with the highest estimated reward is chosen; this is called the greediest action. An action is selected at random with a small probability (ε). This method ensures that if enough trials are done, each action will be tried an infinite number of times, thus ensuring optimal actions are discovered.

Procedure

a) Set the following parameters:
    - Learning rate: 0.2
    - Discount factor: final discount factor in Experiment A3
    - Exploration probability: 0.8, 0.5, 0.3, 0.1
    - Actions: start products, do nothing
    - Reward Function: final reward function in Experiment A1
    - States: final states in Experiment A2

b) Set the step number to 4000 as the learning behaviour was observed to be stabilized.

c) Set the exploration probability to be 0.8.

d) Ensure that the job is at the original starting time. Compile and run the program

e) Repeat step (d) for 10 times.

f) Repeat steps (a) to (d) for the rest of the exploration probability.

Results

Figure 4.4 shows the various settings experimentally of the exploration probability. As can be seen from the figure, smaller exploration probability increased the number of steps required to converge (i.e. time increased), and a higher probability caused the manufacturing system to act too randomly. From this experiment, 0.3 is the selected setting that resulted in the fastest stable progress in learning with large reward increment. So in the following Q-learning experiments, an exploration probability of 0.3 will be used.

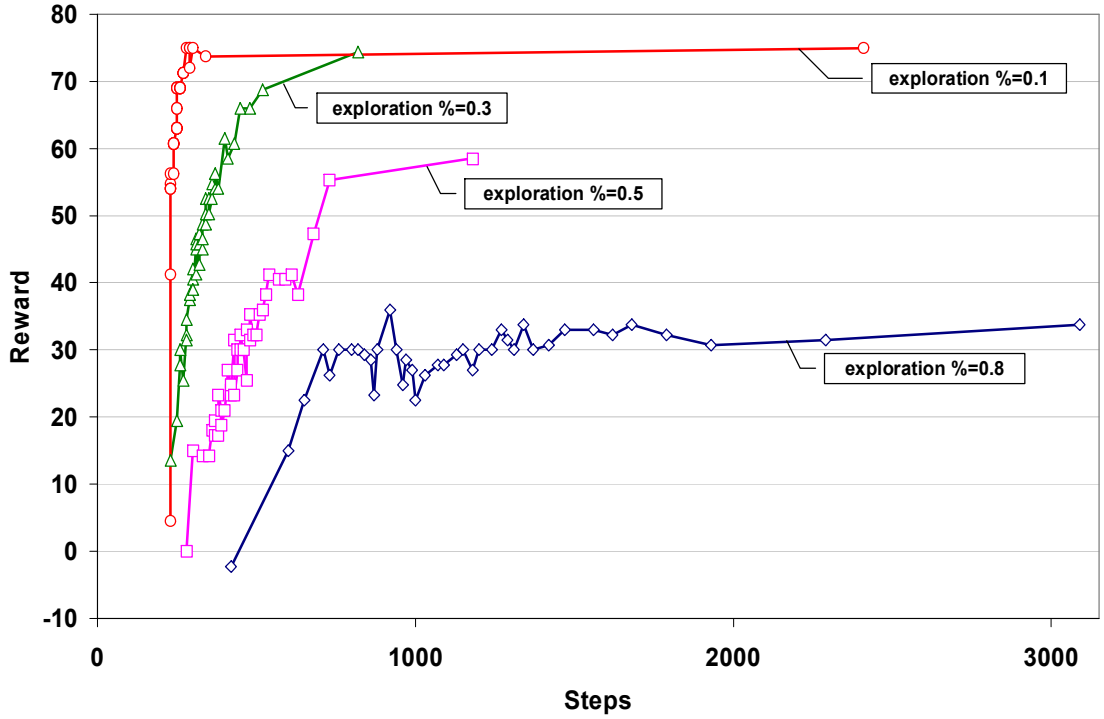


Figure 4.4: Different exploration probabilities

4.2.5 Experiment A5: Learning Rate

The objective of this experiment is to determine the learning rate that demonstrates a fast and stable learning process. The value of this factor can be constant or varied from step to step. Each test was repeated 10 times. The total test time for one experiment was 10 minutes.

Procedure

a) Set the following parameters:

- Three different learning rates: 0.1, 0.2, 0.5
- Discount factor: final discount factor in Experiment A3
- Exploration probability: final percentage in Experiment A4

- Actions: start products, do nothing
- Reward Function: final reward function in Experiment A1
- States: final states in Experiment A2

b) Set the step number to 4000 as the learning behaviour was observed to be stabilized.

c) Set the learning rate to be 0.1.

d) Ensure that the job is at the original starting time. Compile and run the program.

e) Repeat step (d) for 3 times.

f) Repeat steps (a) to (d) for the rest of the learning rates.

Results

The observed behaviour in the experiments was that the steps become smaller and smaller as learning progresses to assure convergence of Q(s, a) values. With a constant step-size parameter, the Q(s, a) values never completely converge but continue to vary in response to the most recently received rewards.

The results of the comparison of the different learning rates can be seen in the graphs in Figure 4.5. Each curve shows an average of the 3 runs and the rewards depicted are the average value of 100 consecutive steps. A general observation from these graphs is that gained reward increases throughout the 4000 learning steps indicating improvements in the behaviour. As can be seen from the curves, no clear conclusion for the best value of the learning rate can be drawn. It is difficult to find the "best" learning rate. From this experiment, 0.2 is the setting

that resulted in the fastest stable progress in learning with large reward increment. So in the following Q-learning experiments, a learning rate of 0.2 will be used.

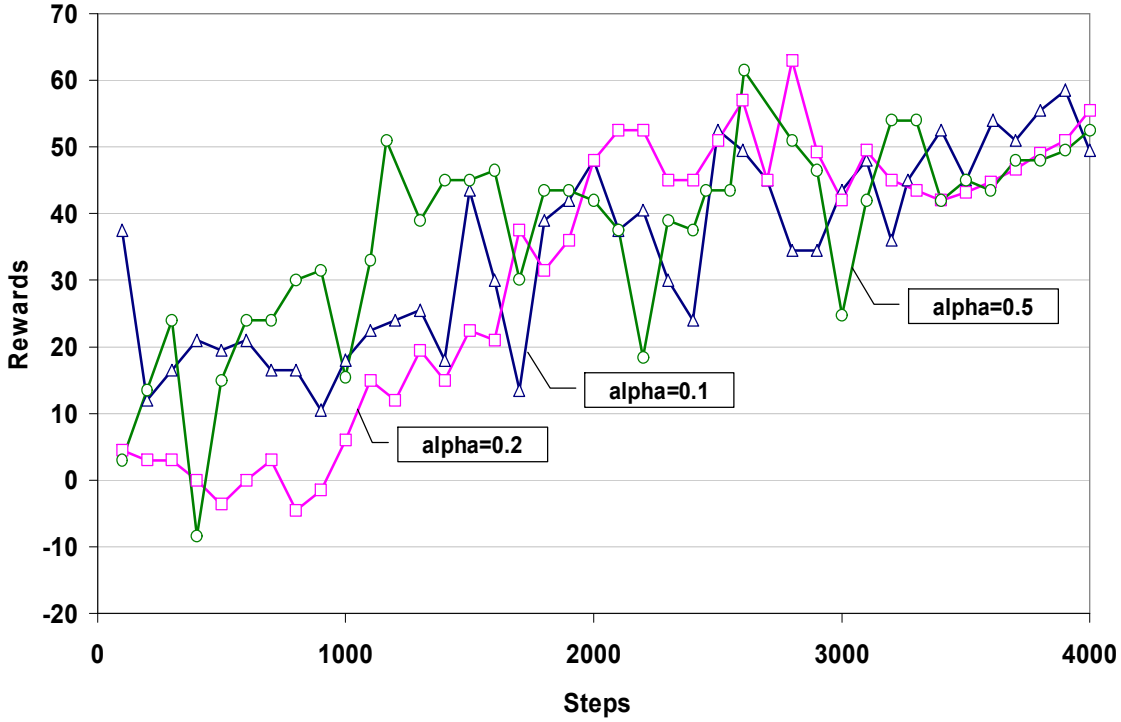


Figure 4.5: Different learning rates

4.3 Experiment B – Learning Results

After the optimal parameters were found (in Table 4.2), the learning of "capacity-planning" behaviour was carried out.

Table 4.2: Optimal parameters affecting capacity allocation learning

Q-learning Parameters	Range	Values used
State	>2	81 states with varying values
Action	>2	>14 actions with varying values
Reward function	complex	Final reward function in Experiment A1
Learning rate	0 - 1	0.2
Discount rate	0 - 1	0.95
Exploration probability	0 - 1	0.3
Steps	varied	4000

4.3.1 Convergence

One way to evaluate Q-learning algorithm is to examine the convergence of the optimal policy. Figure 4.6 shows the three learning runs of the behaviour and the average of the three runs. As the averaged reward increased, it demonstrated that the behaviour was being learnt gradually.

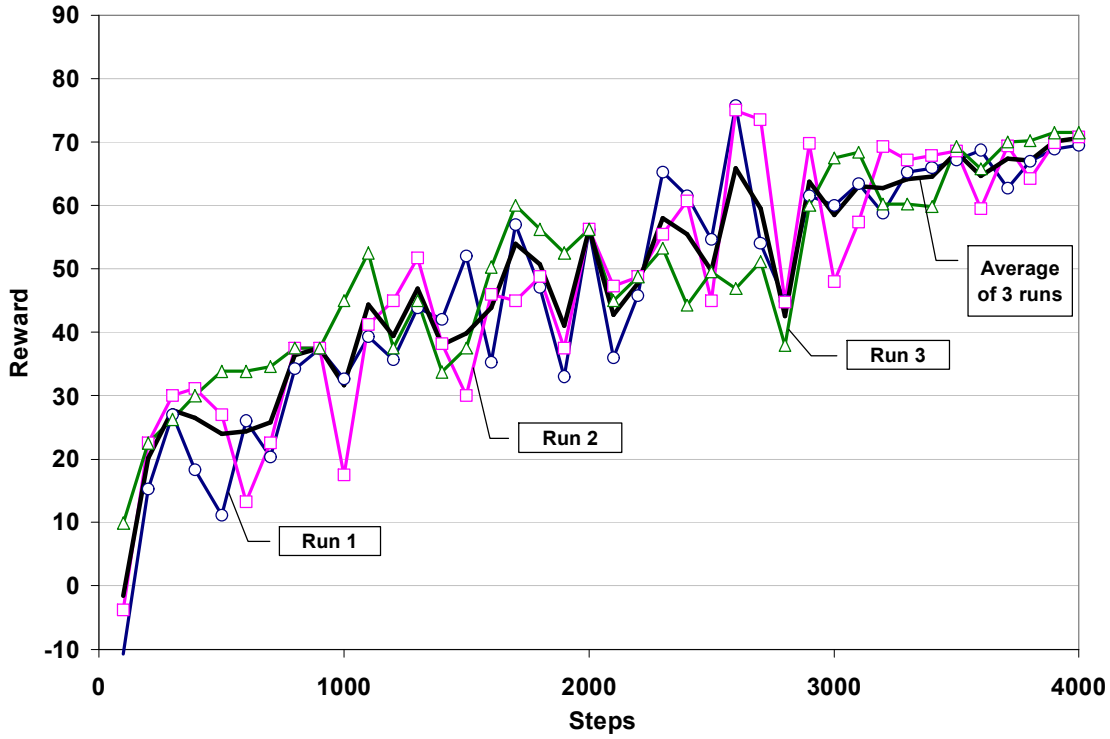


Figure 4.6: Behaviour converging

4.3.2 Optimal Actions and Optimal Q-values

The policy learnt after 4000 steps can be seen in Figure 4.7, which involved the optimal action and state. For example, for state, s=0 and average cycle time, ave CT=0, which means that the order is at any start and decision 1 is made, the optimal action is a=0, that is purely start of any product. Figure 4.8 shows the maximum Q value for each state. It can be observed that the maximum values are smaller total cumulative lateness where the

reward function was maximum. If the total cumulative lateness is large where the reward is minimum but the number of tardy orders increases, the value of this state is lower than the states which the future rewards will also be maximum.

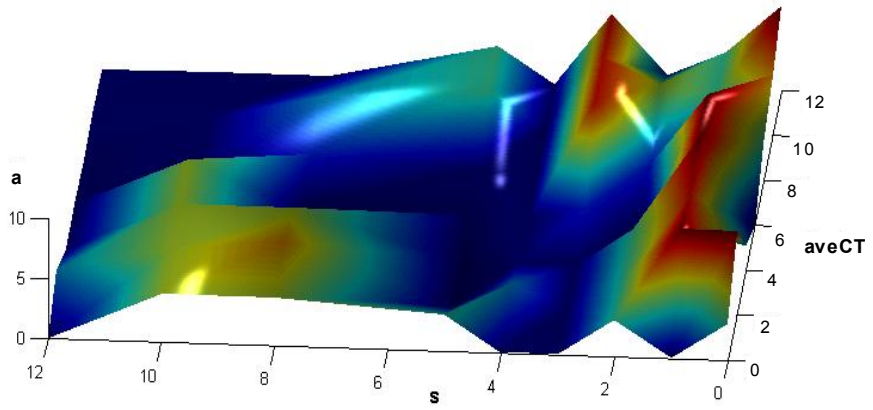


Figure 4.7: State/action policy learnt

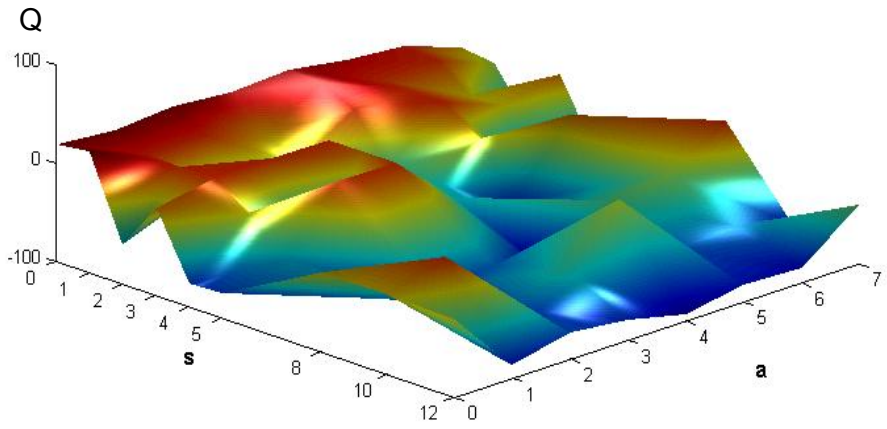


Figure 4.8: Optimal Q-values in given state and action

268 CPU seconds are consumed to generate the optimal actions and Q-values. The initial allocation plan is realized by greedy-searching the requirement of each resource on the basis of a regular lead time of manufacturing an order. For instance, O1 is allocated in six periods of time. The resultant resource allocation plan is illustrated in Figure 4.9.

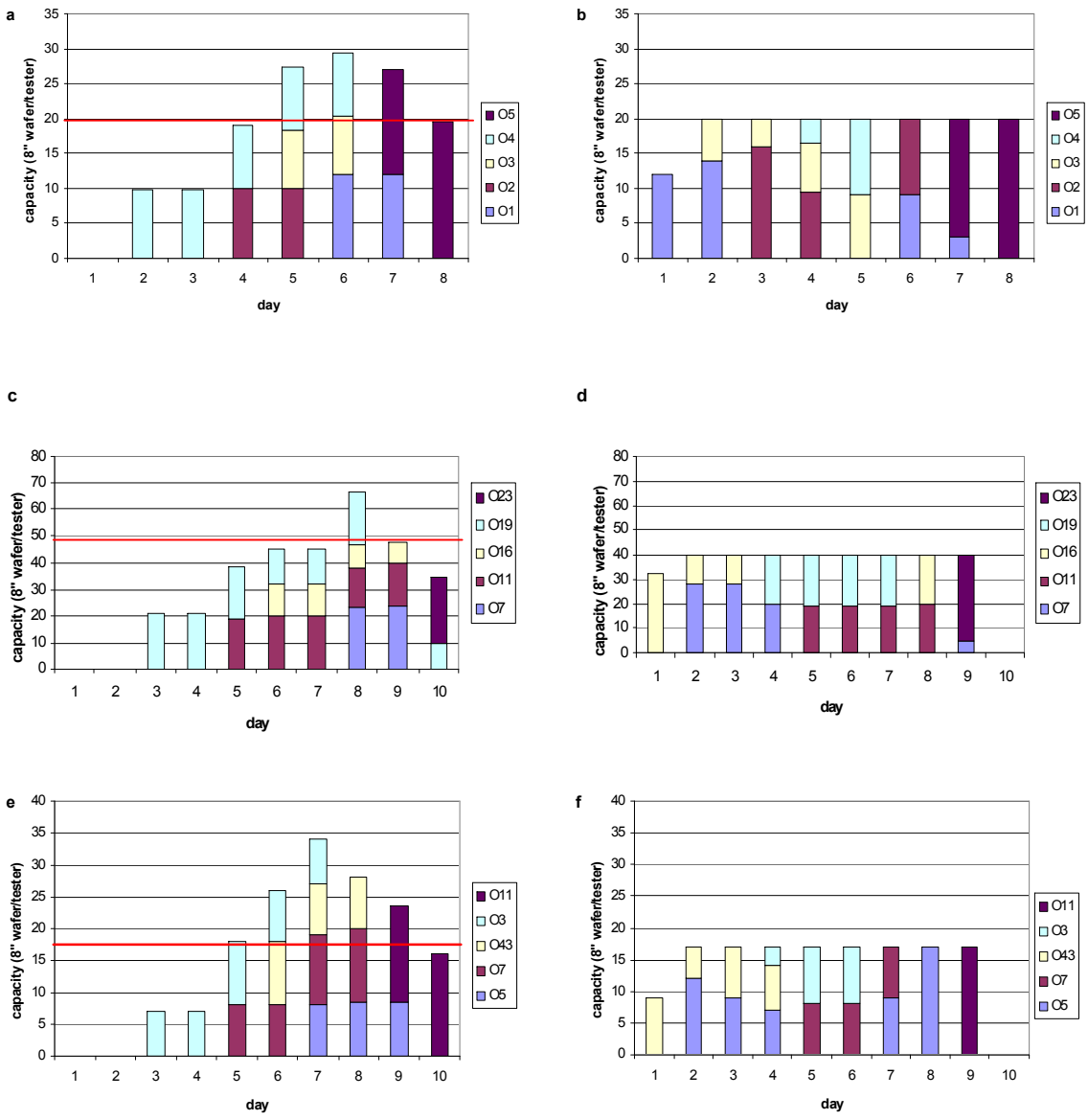


Figure 4.9: Resource allocation of initial and resultant states. (a) Initial resource allocation of tester 1. (b) Resultant resource allocation of tester 1. (c) Initial resource allocation of tester 2. (d) Resultant resource allocation of tester 2. (e) Initial resource allocation of tester 3. (f) Resultant resource allocation of tester 3.

4.3.3 Slack Ratio

The optimal slack ratio of a solution measures the tightness of the solution at the convergence of the optimal policy. The slack ratio is defined as:

$$d = \frac{1}{n}\sum_{i=1}^{n}\frac{D(O_i) - F(O_i)}{D(O_i) - R(O_i)} \tag{4.1}$$

where n is the number of orders, R(O) and D(O) denote the release and due date of order O respectively; F(O) is the finish time of order O. Figure 4.10 shows that when d > 0, the orders were on the average finished before their due dates. It was seen that the slack ratio is closer to zero after 3500 steps till 4000 steps, which means that all the orders meet their customer due dates at the and there were no spare times.

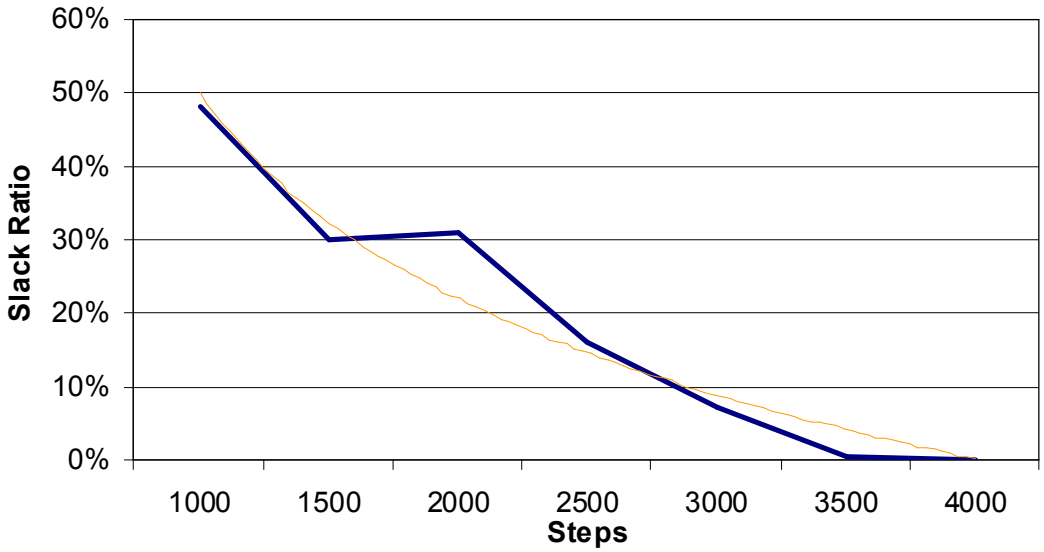


Figure 4.10: The impact of slack ratio

4.4 Experiment C – Changing Environments

This experiment also includes the additional work (behaviour coordinating system) in Chapter 3 that was tested with one event in one run. The behaviour coordination system

consists of three behaviours: “Capacity-planning” behaviour (learning behaviour), “Product development” behaviour and “Teleoperation” behaviour. During the testing phase, the “Teleoperation” behaviour was used to interfere with the “capacity-planning” behaviour, through human interruption to STAP on priority allocation or human operator does maintenance or repair of tester.

In order to demonstrate the algorithm learns in changing environments, experiments were carried out with one tester in one run on the aim of minimizing the makespan. The reward is the average reward in 100 steps. The adaptive features of the system were tested by confronting it with unexpected events, such as: resource breakdowns, new resource availability, new job arrivals or job cancellations. In each case there was an unexpected event at 4000 steps. Two scenarios were considered: (a) restart the learning process, (b) continue the learning process. Figure 4.11 shows that approach (b) is much more efficient. Hence, the rest of the experiments were applying the approach (b). It was also noticed that even if the problem became “easier” after the change in the environment at 4000 steps, for example, a new resource was available or an order was cancelled, the performance started to slightly decrease, i.e. rewards started to decrease, after the event. These experiments illustrated that the system had to “explore” the new configuration by the unexpected events.

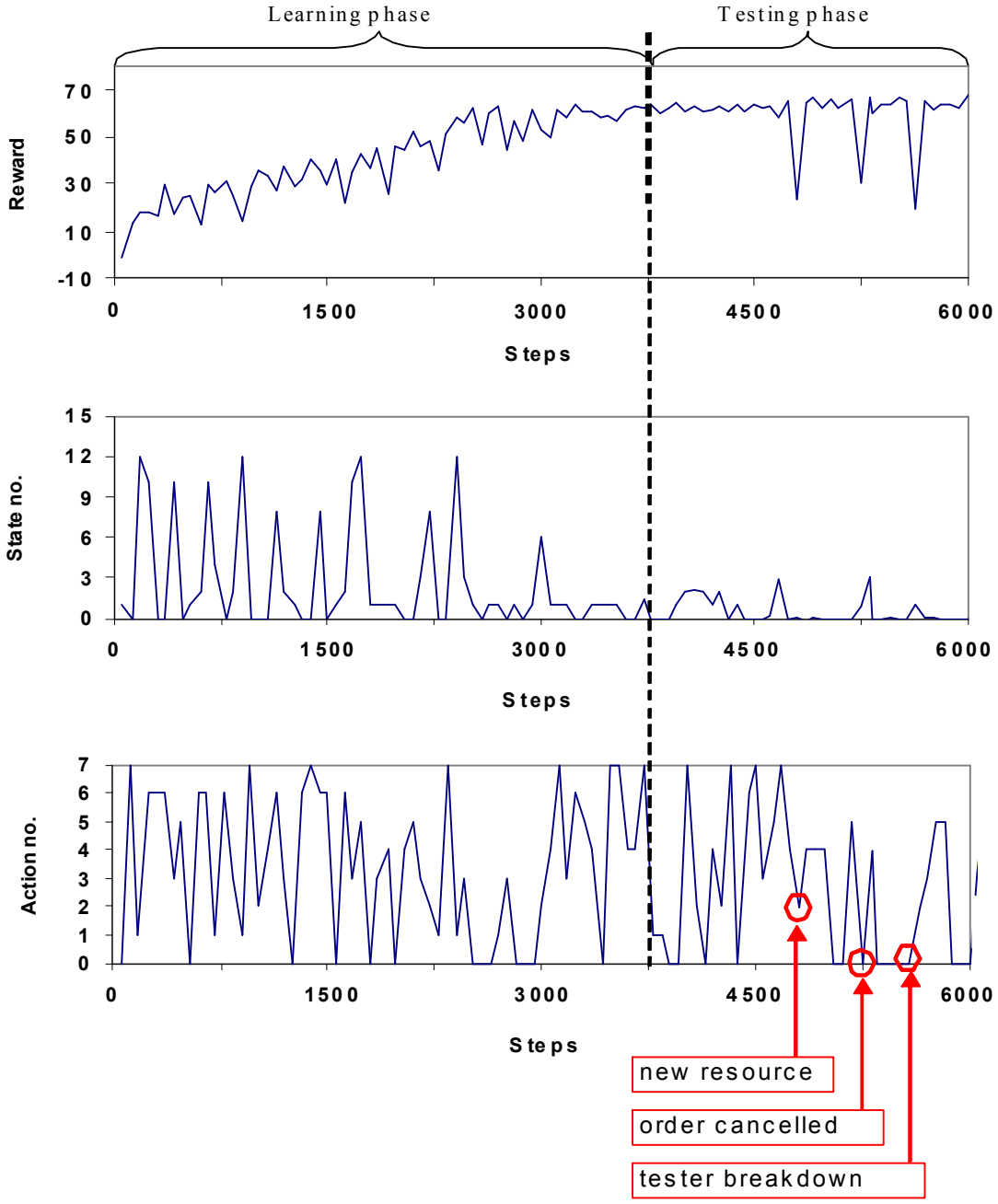


Figure 4.11: Learning and behaviour testing

4.4.1 Unexpected Events Test

The objective of this experiment is to investigate how the numbers of unexpected events affect learning, while allocating the capacity optimally.

Procedure

a) Set the parameters in Table 4.2.

b) Program to record the number of late orders, steps taken to make decision and the total cumulative lateness. Save and compile the program. These data are for later analysis.

c) Repeat step (b) for 3 times.

d) Repeat steps (b) to (c) for inserting 3 events anytime in the testing environment.

Results

The average reward from the runs for each number of events plotted against steps was shown in Figure 4.12. The experiment shows that the learning tends to converge quicker in the environment with 3 events than the one with 1 event. This can be explained since if there are more unexpected events, there are fewer choices for the system to choose, and hence, it takes STAP a shorter time to allocate the capacity with due date completion.

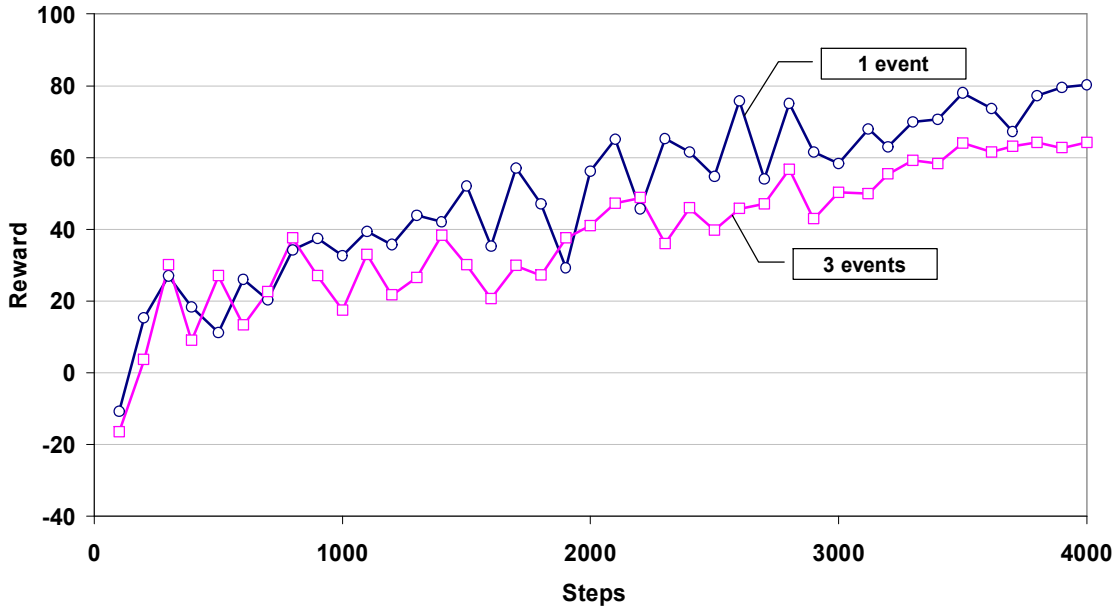


Figure 4.12: Performance in environment with different number of events inserted

4.5 Summary

The Q-learning algorithm was capable of learning behaviour for a manufacturing system in real-time, and facilitated by the continuous states and actions produced by fuzzy logic. "Capacity-planning" behaviour was learnt and results from the experiments can be discussed. The state of the world and reward function can never be measured or modeled exactly. Hence, intuition, experience and experiments are required from the programmer. Besides the definition of the state variables and reward function, it is also necessary to find out the optimal values for the Q-learning parameters. Poor settings of parameters lead to poor learning that does not converge or is slow in converging. The managing of uncertainties was tested in varying environments. This did not influence the outcome of the learning behaviour, but was used to test the performance of the system. In addition, more unexpected events in the environment help to speed up learning as lesser choices are required to be explored.

Chapter 5

Discussions

After the presentation of the experiments, some issues have arisen and can be discussed in the following sections. Section 5.1 presents learning on manufacturing system performing better than without learning. This is done by applying Analysis of Variance (ANOVA) technique, which is a hypothesis testing. Section 5.2 discusses the problems of the overall implemented system. Lastly, Section 5.3 discusses the difficulties faced when implementing Q-learning algorithm.

5.1 Analysis of Variance (ANOVA) on Learning

ANOVA (Montgomery, 1991) is a technique by which variations associated with factors can be used for testing of significance. A common question will arise after all these experiments: with respect to task performance, how significant is the influence of learning on a robot compared to one without learning. To provide the answer, two one-way ANOVAs are performed in two factors variation of one replicate. Table 5.1 first shows the two factors of variation, namely: Agent type (A) and varying environment (B) where the data is related to the number of late orders extracted from Table 5.2. Here, the data characteristic is “the smaller the better”.

Table 5.1: (event) Two factors- Agent type and Varying Environment

Varying Environment (B)	Agent Type (A)	
	Learning	Non-learning
1 event	0	3
3 events	2	7

Table 5.2 shows the effects of learning and non-learning agents under two different environments, namely an environment with 1 event and an environment with 3 events. The total number of late orders and the total number of steps taken are the data for learning agent, which are obtained from Experiment C. By replacing the learning behaviour to the non-learning behaviour, Experiment C is conducted again.

Table 5.2: Raw data from experiments

Agent	Varying Environment	No. of Steps	No. of late orders
Learning	1 event	3635	0
	3 events	2743	2
Non-Learning	1 event	4972	3
	3 events	5729	7

Two questions arise in this case. Firstly, whether the number of late orders influences the learning and secondly, whether the learning agent is behaving similarly to the non-learning one. The ANOVA table is analyzed using SPSS statistic software as shown in Table 5.3. The sources of variation are also known as factors in the experiment. The "Sum of Squares" column represents the measurement of variation due to the parameters. The "df" column is the degree of freedom that is defined as the number of independent observations for measurement. The "Mean Square" is the estimates of variance in "Sum of Squares" per "df". Lastly, the "F" value is the variance ratio of the factors to the error

source of variation. The hypothesis is tested at 5% significant level. The last column (Sig.) in Table 5.3 represents the significant ratio found in the F-distribution table. For example, the significant ratio of agent type, $F_{0.05,1,3}$ is 10.13 in the table where $\upsilon 1=1$ and $\upsilon 2=12$.

The analysis indicates that agent type differs significantly in number of late orders as the calculated F values are higher than the Sig. values. Based on the data characteristics in Table 5.3, the learning agent produces lesser late orders than the non-learning one in any changing environment. On the other hand, under the Varying Environment (B) column, the calculated F tends to be near to the Sig. value. This might be due to the sensitivity of data size. Perhaps, it would be better to increase one replication to have more conclusive results. For example, when increasing one replication and Sig. becomes 5.12, the factor of varying environment will be significant.

Table 5.3: ANOVA Table (Late orders)

Sources of Variation	Sum of Squares	df	Mean Square	F	Sig.
Agent Type (A)	16	1	16	16	10.13
Varying Environment (B)	9	1	9	9	10.13
Error	1	1	1		
Total	26	3			

Next, Table 5.4 again shows the two factors of variation, namely: Agent type (A) and varying environment (B) where the data is related to the number of steps taken to make decision. Similarly, the data is extracted from Table 5.2.

Table 5.4: (Steps taken) Two factors- Agent type and Varying Environments

Varying Environment (B)	**Agent Type (A)**	
	Learning	**Non-learning**
1 event	3635	4972
3 events	2743	5729

Again, the ANOVA table is analyzed as shown in Table 5.5. The analysis also indicates that agent type differs significantly in makespan as the calculated F values are higher than the Sig. values. The learning agent explores in smaller number of steps under the environment with 3 events than the non-learning one. In this case, the calculated F under the Varying Environment (B) column has a very small value of less than 0.1. Hence, the effect of the factor, Varying Environment (B), is negligible.

Table 5.5: ANOVA Table (Steps taken)

Sources of Variation	Sum of Squares	df	Mean Square	F	Sig.
Agent Type (A)	4251844	1	4251844	10.3159	10.13
Varying Environment (B)	1024	1	1024	0.002484	
Error	412164	1	412164		
Total	4665032	3			

In conclusion, the learning agent performs better than the non-learning one as it produces fewer or no late orders even in varying environment. It also takes lesser time to make decision while minimizing late jobs.

5.2 Problems of Implemented System

An implementation concerning the action used in the learning process is discussed. In the RL update rule, action a_t is the action at time t and contributes to the achievement of state s_{t+1}. However, when this original procedure is used, learning had problems in converging. After analyzing the executed actions and state transitions, a delay in transition was found. So it was more logical to consider action a_{t-1} for the change from s_t to s_{t+1}.

Fuzzy logic appears to be an ideal tool for creating a continuous state and action in Q-learning. The advantage of using fuzzy logic is that it models different types of uncertainty, imprecision and provides continuous actions to learning. Consequently, there were an enormous number of actions to be looked up and increasing the exploration time led to inefficient process. The main weakness of fuzzy logic is the dependency of the well-defined fixed rules. This is a tedious step and does not allow much error tolerance in operation that affects robustness of the implemented system. Hence, it may be interesting to explore and test other tools such as Artificial Neural Network in producing continuous states and actions.

In all the experiments, all the value settings have to be changed in the program. This is very troublesome and time-consuming as saving, debugging and compiling of the program have to be done for every changes made. This can be resolved by developing a user interface to adjust the value of the parameters. An example of the graphical user interface template is shown in Figure 5.1.

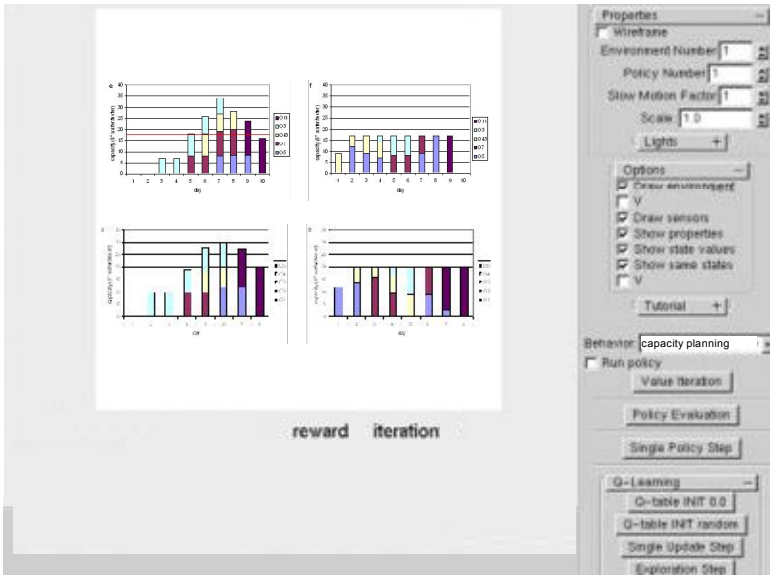


Figure 5.1: The Q-learning graphical user interface template

5.3 Q-learning Implementation Difficulties

It is an important design issue to map the inputs into a set of states to Q-learning algorithm. This is designed based on programmer intuition and experience. However, the main focus is trying to minimize the number of states. From experiment A2, it was observed that some states will never be experienced by the system. Hence the reduced number of states results in faster convergence to the desired behaviour. This is because the size of the Q-table is smaller. In addition, the measurement of the state will depend on the sample time of the algorithm. If the sample time is very small, the state measurement must have a higher precision. The use of TD(λ) algorithm described in Chapter 3 could probably minimize this problem as they update the values of a set of past states and cause less sensitivity to poor accuracy in measurement of input.

The reward function is the only information that the programmer must introduce in order to achieve the mission of the behaviour. This is designed based on programmer intuition,

experience and observation on the manufacturing system in real-time. A poorly chosen reward function can cause converging difficulties in learning. Some trials had been tested in experiment A1 to choose the optimal reward function for the "capacity-planning" behaviour.

The definitions of the reward function and the states are the most important parameters for the Q-learning algorithm. Next, different settings of the Q-learning parameters affect learning as shown from experiment A3 to experiment A5. In general, poor designs of the reward function and states, and poor settings of parameters can lead to poor learning.

The Q-learning algorithm cannot function with inaccurate or incomplete state data as it cannot differentiate the false backlog from the true backlog insertions, and will be misled. This deficiency is solved by improving the sensing and processing as discussed in Chapter 3, where order grouping management and multithreading concept were introduced. Despite the above problem, the Q-learning algorithm is able to resolve the behaviour workout principles by itself – it simplifies the programming of behaviour. As an off-policy algorithm has the advantage of allowing the interaction of the other behaviours while learning.

Chapter 6

Conclusion

In the real world domain, it is important to allocate scarce and reusable resources over time to interconnected tasks efficiently. In the past decades, programmers spent much effort on designing resource allocation algorithms especially for deterministic and static problems. However, in practice there is a significant amount of uncertainty concerning resource allocation, for example, the execution times of the tasks are usually incomplete. Moreover, there could be disturbances and changes in the environment, as well. These issues seriously limit the applicability of classical solution methods. Unfortunately, it is not trivial in extending these algorithms to handle stochastic problems. Instead of this direction, an investigation of the learning concept in this thesis was carried out in resource allocation to deal with stochastic problems. This chapter concludes the work presented throughout this thesis. Section 6.1 summarizes what was achieved and the contributions of this work. Finally, Section 6.2 suggests some interesting directions for further research.

6.1 Contributions

This research has investigated whether learning performs better than non-learning in the context of semiconductor wafer testing application. The agent behaviours were identified and implemented in the resource allocation task. “Capacity-planning” behaviour was selected for learning so that the agent could learn to improve its performance in the task.

Reinforcement learning was selected to be a suitable learning method for resource allocation learning. Its unsupervised online applicability and the non-requirement of any previous information about the environment are the most important advantages. There are various reinforcement learning algorithms and the Q-learning algorithm, which is capable of off-policy learning, was selected. This ability is important for allocation problems as it reduces the experimentation time.

It was demonstrated that the Q-learning algorithm simplifies the program design. The programmer only needs to specify the environment states and outputs with definition of reinforcement function; a deep understanding of the system's behaviour and environmental interactions is not required. However, it was illustrated in experiments that defining of the required parameters is crucial as it influences overall learning greatly.

The Q-learning algorithm is based on discrete representations of state and action. However, large-scale stochastic problems usually include continuous state and action variables; the discretization of the variables causes an enormous number of states and long learning time. Fuzzy logic was attempted to solve this problem in facilitating the Q-learning algorithm as shown in the experiments. However, it does not imply that fuzzy logic is the ideal tool for creating continuous states and actions in learning.

A framework of RAP is defined with a generalization of some standard resource allocation problems. This general RAP is then reformulated as a stochastic MDP. It has been shown that this reformulated problem has favorable properties, such as, it is aperiodic, its state and action spaces are finite, all policies are proper and the space of control policies can be safely restricted. The control polices of the reformulated problem

are defined by reactive solutions of stochastic RAPs. As demonstrated by the experiments, the RAP can be reformulated by the represented framework to model several resource management problems.

All data were extracted in the programs by sequential steps that caused STAP unable to sense the environment in real-time and delay the decision-made. It was observed that the performance improved when multithreading of extraction of data from various databases was proposed. However, too many threads performing in the program could cause ineffective performance. The orders were grouped into five different groupings to solve this problem. A threshold limit function from the real factory's world-wide standard was proposed to let STAP know when the maximum lateness can be considered in terms of times cycle time according based on the different situations faced in the environment.

Apart from developing a learning agent, additional work in developing coordination architecture was done by fuzzy logic technique. It was simple and reduced the confusion of behaviour switching as there was no specific tuning parameter involved. Besides pure reactive behaviours being incorporated into the architecture, human user command was also involved. Importantly, one should analyze the coordination strategy among behaviours first and their priorities must be designed. It was observed from the experiment that using off-policy Q-learning, in which the agent was able to learn behaviour and interact with other non-learning behaviours while performing learning is beneficial.

This resource allocation learning research with the presented RAP framework can be used to modeling several resource management problems that appear in practice, such as

security surveillance and intrusion detecting, sales and marketing, inventory management or maintenance and repair problems. Moreover, it forms the first step towards more complex real world learning tasks in those applications.

6.2 Future Work

The development of this research always provokes the discovery of new problems as well as other new interesting areas. Different aspects are considered to be worthy of investigation and were discussed as follows.

- Further Q-learning testing on other industries or multiple factories: Building communication channels between each factory's situations which are characterized by supply chain networks allows us to investigate how multiple factories affect learning.
- Improving continuous states and actions: Q-Learning is conventionally applied to discrete state and action problems. However, large-scale RAPs usually include continuous state and action variables. There are a number of possible approaches for extending the Q-learning framework directly to continuous state and action spaces. For example, ANN can also be used in Q-learning to produce continuity and smoothness in actions.
- Speeding up of optimization process: The suggested improvements, such as clustering and distributed sampling, should be further investigated, both theoretically and experimentally, since they resulted in considerable speedup.
- Investigating of proactive (open-loop) solutions: Explore the ways in extending the reactive approach to proactive solutions to handle partially observable DP. The theoretical analysis of the average effects of environmental changes on the value functions of a policy could result in new approaches to handle disturbances.

- Development of user interface for Q-learning algorithm: This interface needs to be connected to the system. Users can easily set or find the optimal parameters of the Q-learning algorithm in the interface. The graphical outputs of selected parameters can also be viewed in the user-friendly interface.
- Learning of parameters in Q-learning algorithm: There are several machine learning algorithms as discussed in Chapter 2, such as ANN. A system involving various learning algorithms can be designed, implemented and tested to investigate the best learning algorithm. Instead of using expert knowledge in finetuning the Q-learning algorithm, it can be explored to apply the learning module to search for the optimal parameter settings with results measured by the speed of processing, i.e. learning within learning concept.

Bibliography

Aberdeen, D. (2003). *A (revised) survey of approximate methods for solving partially observable Markov decision processes*. Technical report, National ICT Australia, Canberra, 2003.

Annual Report (2005). Edited by Judith Paulus. Semiconductor Industry Association, 181 Metro Drive, Suite 450, San Jose, California, 2005.

Arkin, R. C. (1998). *Behaviour-Based Robotics*. MIT Press 1998.

Åström, K. (1965). *Optimal control of Markov processes with incomplete state information*. Journal of Mathematical Analysis and Applications (JMAA), vol. 10, pp. 174–205, 1965.

Aydin, M. E. and Öztemel, E. (2000). *Dynamic job-shop scheduling using reinforcement learning agents*, Robotics and Autonomous Systems, vol. 33, pp. 169–178, 2000.

B. Van Roy. (2001). *Neuro-dynamic programming: Overview and recent trends*. In E. Feinberg and A. Shwartz, editors, Handbook of Markov Decision Processes: Methods and Applications, Kluwer, Boston, MA, 2001.

Baldwin, K.E. (1989). *Autonomous manufacturing systems*. Intelligent Control, Proceedings of IEEE International Symposium, vol. 25-26, pp. 214–220, 1989.

Baldwin, K.E. (1989). *Autonomous manufacturing systems*. Intelligent Control, Proceedings of IEEE International Symposium, vol. 25-26, pp. 214–220, 1989.

Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour*, Princeton University Press, 1961.

Brooks, R.A. (1986). *A Robust Layered Control System for a Mobile Robot*. IEEE Journal of Robotics and Automation, vol. RA, issue. 2, pp. 14-23, 1986.

Bulitko, V. and Lee, G. (2006). *Learning in real-time search: A unifying framework*. Journal of Artificial Intelligence Research, vol. 25, pp. 119–157, 2006.

Chung IF, Lin CJ, Lin CT. (2000). *A GA-based fuzzy adaptive learning control network*. Fuzzy Sets Syst vol. 112, pp. 65–84, 2000.

Csáji, B. Cs. and Monostori, L. (2005a). *Stochastic approximate scheduling by neurodynamic learning*. In Proceedings of the 16th International Federation of Automatic Control World Congress, July 3–8, 2005a.

Csáji, B. Cs. and Monostori, L. (2005b). *Stochastic reactive production scheduling by multi-agent based asynchronous approximate dynamic programming*. In Proceedings of the 4th International Central and Eastern European Conference on Multi-Agent Systems, vol. 3690, pp. 388–397, 2005b.

Csáji, B. Cs. and Monostori, L. (2006a). *Adaptive algorithms in distributed resource allocation*. In Proceedings of the 6th International Workshop on Emergent Synthesis, August 18–19, The University of Tokyo, Japan, pp. 69–75, 2006a.

Csáji, B. Cs. and Monostori, L. (2006b). *Adaptive sampling based large-scale stochastic resource control*. In Proceedings of the 21st National Conference on Artificial Intelligence, July 16–20, Boston, USA, pp. 815–820, 2006b.

Csáji, B. Cs., Kádár, B., and Monostori, L. (2003). *Improving multi-agent based scheduling by neurodynamic programming*. In Proceedings of the 1st International Conference on Holonic and Mult-Agent Systems for Manufacturing, vol. 2744, pp. 110–123, 2003.

Csáji, B. Cs., Monostori, L., and Kádár, B. (2004). *Learning and cooperation in a distributed market-based production control system*. In Proceedings of the 5th International Workshop on Emergent Synthesis, May 24–25, Budapest, Hungary, pp. 109–116, 2004.

Csáji, B. Cs., Monostori, L., and Kádár, B. (2006). *Reinforcement learning in a distributed market-based production control system*. Advanced Engineering Informatics, vol. 20, pp. 279–288, 2006.

D. L. Benavides, J. R. Duley and B. E. Johnson (1999). *As good as it gets: optimal fab design and deployment*. IEEE Trans. Semiconductor Manufacturing, vol. 12, issue 3, pp. 281-287, 1999.

D.C. Dracopoulos (1998). *Neural Robot Path Planning: The Maze Problem, Neural Computing & Applications*. Springer-Verlag London Limited, vol. 7, issue 2, pp. 115-120, 1998.

Davies, A. and Bischoff, E. (1999). *Weight distribution considerations in container loading*. European Journal of Operational Research, vol. 114, pp. 509–527, 1999.

Even-Dar, E. and Mansour, Y. (2003). *Learning rates for Q-learning*. Journal of Machine Learning Research, vol. 5, pp. 1–25, 2003.

Feinberg, E. A. and Shwartz, A. (2003). *Handbook of Markov Decision Processes: Methods and Applications*. Kluwer Academic Publishers, 2003.

Gen, M. and Cheng, R. (2000). *Genetic Algorithms and Engineering Optimization*. John Wiley, New York, 2000.

Hatvany, J. and Nemes, L. (1978). *Intelligent manufacturing systems - a tentative forecast*,. In Proceedings of the 7th IFAC World Congress A., a link between science and applications of automatic control, vol. 2, pp. 895–899, 1978.

Holland, J.H. (1985). *Properties of the bucket brigade*,. In Proceedings of the 1st International Conference on Genetic Algorithms. Hillsdale, NJ, vol. 1, pp. 1 - 7, 1985.

Kaelbling, L. P., Littman, M.L., and Moore, A. W. (1996). *Reinforcement Learning: A Survey*. Journal of Artificial Intelligent Research, vol. 4, 1996.

Kalmár, Zs., Szepesvári, Cs., and Lorincz, A. (1998). *Module-based reinforcement learning: Experiments with a real robot*. Machine Learning, vol. 31, pp. 55–85, 1998.

Kovács, A. and Beck, J. C. (2007). *A global constraint for total weighted completion time*. In Proceedings of the 4th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, vol. 4510, pp. 112–126, 2007.

Koza, J. R., Bennett, F. H., Andre, D. and Keane, M. A. (1999). *Genetic Programming III: Darwinian Invention & Problem Solving*. Morgan Kaufmann Publishers, San Francisco, 1999.

Langdon, W. B. (1998). *Genetic Programming and Data Structures: genetic programming + data structures = automatic programming*. Kluwer Academic Publishers, Boston, 1998.

Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., and Shmoys, D. B. (1993). *Sequencing and scheduling: algorithms and complexity*. Handbooks in Operations Research and Management Science, 1993.

Lovász, L. and Gács, P. (1999). *Complexity of Algorithms*. Lecture Notes, Boston University, Yale University, 1999.

Luger, G.F. (2002). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Fourth edition. New York: Addison-Wesley, 2002.

M. Andrecut and M. K. Ali (2002). *Fuzzy Reinforcement Learning*. International Journal of Modern Physics C, vol. 13, issue 5, pp. 659-674, 2002.

M. Pinedo. (1995). *Scheduling Theory, Algorithms, and Systems*. Prentice Hall, 1995.

Michael J. Riezenman (1996). *Test and Measurement*. IEEE Spectrum, pp. 65-69, January, 1996.

Mitchell, T. M. (1997). *Machine Learning*. The McGraw-Hill Companies, Inc, 1997.

Montgomery, D.C. (2001). *Design and Analysis of Experiments*. Third Edition, John Wiley and Sons, Inc., New York, 2001.

Narendra, K.S. & Thathachar (1989). M.A.L., *Learning Automata: An Introduction*. Prentice Hall International, 1989.

Neumann. J (1987). *The general and logical theory of automata*. Cambridge: The MIT Press, pp. 391-431, 1987.

Nikos Vlassis, Geoff Gordon and Joelle Pineau (2006). *Planning under uncertainty in robotics*. Robotics and Autonomous Systems, vol. 54, issue 11, pp. 885-886, 2006.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley, 1994.

Peng, J. and Williams, R. J. (1996). *Incremental multi-step Q-learning*. Machine Learning vol. 22, pp. 283–290, 1996.

Pinedo, M. (2002). *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 2002.

Powell, W. B. and Van Roy, B. (2004). *chapter Approximate Dynamic Programming for High-Dimensional Resource Allocation Problems*. Handbook of Learning and Approximate Dynamic Programming, IEEE Press, Wiley-Interscience, pp. 261–283, 2004.

Precup, D., Sutton, R. S., and Singh, S. (2000). *Eligibility traces for off-policy policy evaluation*. In Proceedings of the 17th Conference on Machine Learning, Morgan Kaufman, 2000.

Riedmiller, S. and Riedmiller, M. (1999). *A neural reinforcement learning approach to learn local dispatching policies in production scheduling*. In Proceedings of the 16th International Joint Conference on Artificial Intelligence, Stockholm, Sweden, pp. 764U–771, 1999.

Rita L.Atkinson, Richard C. Atkinson, Edward E. Smith, Daryl J. Bem, Susan Nolen-Hoeksema (1996). *Introduction to Psychology*. 12*th* Edition, Harcourt Brace College Publishers, 1996.

Russel, S.J. & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Upper Saddle River: Prentice-Hall, 1995.

S. Geary, S. M. Disney, D. R. Towill (2006). *On bullwhip in supply chains - Historical review, present practice and expected future impact*. Int. J. Production Economics, vol. 101, issue 1, pp. 2-18, 2006.

S. Girgin, M. Loth, R. Munos, P. Preux, and D. Ryabko (2008). *Basis expansion in natural actor critic methods in Recent Advances in Reinforcement Learning*. Lecture Notes in Artificial Intelligence, Springer, vol. 5323. pp. 111–124, 2008.

S. Vijayakumar, S. Schaal (2000). *Fast and Efficient incremental Learning for High-dimensional Movement Systems*. In Proceedings of the 2000 IEEE International Conference on Robotics and Automation, April, San Francisco-USA, pp. 1894-1899,2000.

S.E. Ante (2003). *Global chip sales up 1.3% in recovery year*. BusinessWeek. www.businessweek.org, March 7, 2003.

S.J. Hood, S. Bermon, F. Barahona (2003). *Capacity planning under demand uncertainty for semiconductor manufacturing*. IEEE Trans. Semiconductor Manufacturing, vol. 16, no. 2, pp. 273-280, 2003.

Schneider, J. G., Boyan, J. A., and Moore, A. W. (1998). *Value function based production scheduling*. In Proceedings of the 15th International Conference on Machine Learning, Morgan Kaufmann, San Francisco, California, pp. 522–530, 1998.

Sheridan, T. B. (1992). *Telerobotics, Automation, and Human Supervisory Control*. Cambridge: The MIT Press, 1992.

Sim Siang Kok, Ong Kai Wei and Gerald Seet (2003). *A Foundation for Robot Learning*. IEEE 4th IEEE International Conference on Control and Automation, June 10-12, pp. 75-79, 2003.

Simon, H. A. (1983). *Why Should Machines Learn*? In: Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (eds), Machine Learning: An Artificial Intelligence Approach Vol 1. Morgan Kaufmann, San Mateo, CA, 1983.

Singh, S. and Bertsekas, D. (1997). *Reinforcement learning for dynamic channel allocation in cellular telephone systems*. In Advances in Neural Information Processing Systems, vol. 9, pp. 974–980. The MIT Press, 1997.

Sontag, E. D. (1998). *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. Springer, New York.

STATS (2007). *SICAS Capacity and Utilization Rates Q3 2007*. Industry Statistics, Semiconductor Industry Association, 13 November, 2007.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Bradford Books, MIT, 1998.

Tanimoto, S. (1994). *The Elements of Artificial Intelligence*. Second Edition, W. H. Freeman and Company, New York, 1994.

Touretzky, D. S. and Saksida, L. M. (1997). *Operant Conditioning in Skinnerbots*. Adaptive Behaviour, vol.5 (3/4), pp. 219-247, 1997.

Turing. A. (1950). *Computing machinery and intelligence*. Mind, vol. 59, issue 236, pp. 4–30, 1950.

Van Roy, B., Bertsekas, D., Lee, Y., and Tsitsiklis, J. (1996). *A neuro dynamic programming approach to retailer inventory management*. Technical report, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, 1996.

W. Powell (2007). *Approximate Dynamic Programming*. Wiley, 2007.

Watkins (1989). C. J. C. H., *Learning from Delayed Rewards*. Ph.D. thesis, University of Cambridge, 1989.

Y.C. Wang, J.M. Usher (2005). *Application of reinforcement learning for agent-based production scheduling*. Engineering Applications of Artificial Intelligence, vol. 18, pp. 73-78, 2005.

Zhang, W. and Dietterich, T. (1995). *A reinforcement learning approach to job-shop scheduling*. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, Morgan Kauffman, pp. 1114–1120, 1995.

Appendix A : Sample data (system input)

Item	Start	Wafe	Production	WC	Min. Qty	Additional	Chosen Qt	Plan Qty	Demand Ty	Demand St	Wk 17 09	Wk 18 09	Wk 19 09	Wk 20 09	Wk 21 09	Wk 22 09	Wk 23 09
F883CSR6	X883CSR6	8	APEE	AGHP93K0	0	5	0	0	BKL1	PKE	0	0	3	0	0	0	0
F883CSR6	X883CSR6	8	APEE	AGHP93K0	0	0	0	0	BKL1	LATE	0	0	2	0	2	0	0
F883CSR6	X883CSR6	8	APEE	AGHP93K0	0	0	0	0	BKL2	JIT	0	0	0	0	0	3	4
F883CUK6	X883CUK6	8	APEE	AGHP93K0	0	0	0	0	BKL1	JIT	0	0	0	0	0	4	4
F883CUK6	X883CUK6	8	APEE	AGHP93K0	0	0	0	0	BKL1	LATE	0	0	0	0	4	0	0
F883CUK6	X883CUK6	8	APEE	AGHP93K0	0	0	0	0	BKL1	SAT	0	0	0	0	0	3	0
F883CUK6	X883CUK6	8	APEE	AGHP93K0	0	0	0	0	BKL2	JIT	0	0	0	0	0	4	4
F883CUKA	X883CUKA	8	APEE	AGHP93K0	0	4	0	0	BKL1	PKE	0	0	1	0	0	0	0
F883CUKA	X883CUKA	8	APEE	AGHP93K0	0	0	0	0	BKL2	JIT	0	0	0	0	0	0	0
F883CUKA	X883CUKA	8	APEE	AGHP93K0	0	0	0	0	BKL2	PKE	0	0	3	0	0	0	0
F883CXB6	X883CXB6	8	APEE	AGHP93K0	0	16	0	0	BKL1	JIT	0	0	1	0	10	0	0
F883CXB6	X883CXB6	8	APEE	AGHP93K0	0	0	0	0	BKL1	PKE	0	0	12	0	0	0	0
F883CXB6	X883CXB6	8	APEE	AGHP93K0	0	0	0	0	BKL1	LATE	0	0	3	0	6	0	0
F883CXI6	X883CXI6	8	APEE	AGHP93K0	0	0	0	0	BKL1	DEL	0	0	0	0	1	0	0
F95641P5	X95641P5	8	APEE	AGJ75_E0	0	24	0	0	BKL1	PKE	0	0	24	0	0	0	0
F95641P6	X95641P6	8	APEE	AGJ75_E0	0	25	0	0	BKL1	PKE	0	0	0	25	0	0	0
F95641PM	X95641PM	8	APEE	AGGII_E0	0	98	0	0	BKL1	PKE	98	0	0	0	0	0	48
F95641PM	X95641PM	8	APEE	AGGII_E0	0	0	0	0	BKL2	PKE	0	0	0	0	0	0	1
F95641PU	X95641PU	8	APEE	AGGII_E0	0	50	0	0	BKL1	PKE	50	0	0	0	0	0	0
F962KBJZ	X962KBJZ	8	APEE	AGITSCV0	0	24	0	0	BKL2	PKE	0	0	24	0	0	0	0
C96VA3PP	X96VA1SC	6	APEE	AGJ750A1	0	20	0	0	DES STK	DEL	10	0	10	0	0	0	0
C96VA3PP	X96VA1TC	6	APEE	AGJ750A1	0	28	0	0	BKL1	PKE	14	0	0	0	0	0	0
C96VA3PP	X96VA1TC	6	APEE	AGJ750A1	0	0	0	0	DES STK	JIT	0	0	12	0	0	0	0
C96VA3PP	X96VA1TC	6	APEE	AGJ750A1	0	0	0	0	DES STK	LATE	0	0	2	0	0	0	8
C96VA5OD	X96VA6MD	6	APEE	AGJ750A1	0	22	0	0	BKL1	PKE	0	2	0	0	0	0	0
C96VA5OD	X96VA6MD	6	APEE	AGJ750A1	0	0	0	0	DES STK	DEL	0	20	0	0	0	1	0
C96VA5OD	X96VA6MD	6	APEE	AGJ750A1	0	0	0	0	DES STK	JIT	0	0	0	0	0	0	0
C96VA3OD	X96VA6RD	6	APEE	AGJ750A1	8	50	0	8	BKL1	PKE	4	0	0	0	0	0	0
C96VA3OD	X96VA6RD	6	APEE	AGJ750A1	0	0	0	0	BKL2	JIT	8	0	0	0	0	0	0
C96VA3OD	X96VA6RD	6	APEE	AGJ750A1	0	0	0	0	BKL2	PKE	12	22	0	0	0	0	0
C96VA3OD	X96VA6RD	6	APEE	AGJ750A1	0	0	0	0	DES STK	DEL	0	0	12	26	0	10	0
E984SABQ	X984SABQ	12	APEE	AGSAP__B	0	48	0	0	BKL1	PKE	0	0	48	0	0	215	214
C9HVA3PP	X9HVA1TC	6	APEE	AGJ750A1	0	2	0	0	BKL1	PKE	2	0	0	0	0	0	0
C9HVA5PP	X9HVA3MC	6	APEE	AGJ750A1	0	2	0	0	BKL1	PKE	2	0	0	0	0	0	0
F9L281PB	X9L281PB	8	APEE	AGE2ROM0	0	22	0	0	BKL1	JIT	0	0	0	0	0	0	0
F9L281PB	X9L281PB	8	APEE	AGE2ROM0	0	0	0	0	BKL1	PKE	22	0	0	0	0	0	0
F9L281PB	X9L281PB	8	APEE	AGE2ROM0	0	0	0	0	BKL1	LATE	0	0	0	0	0	0	0
F9L281PB	X9L281PB	8	APEE	AGE2ROM0	0	0	0	0	BKL1	SAT	0	0	0	0	0	0	0
F9L281PB	X9L281PB	8	APEE	AGE2ROM0	0	0	0	0	BKL2	PKE	0	0	0	0	0	0	0
F9L281PM	X9L281PM	8	APEE	AGE2ROM0	0	49	0	0	BKL1	JIT	0	0	0	0	0	0	0
F9L281PM	X9L281PM	8	APEE	AGE2ROM0	0	0	0	0	BKL1	PKE	47	0	0	0	0	0	0
F9L281PM	X9L281PM	8	APEE	AGE2ROM0	0	0	0	0	BKL2	PKE	2	0	0	0	0	0	0
F9L281PN	X9L281PN	8	APEE	AGE2ROM0	0	0	0	0	BKL1	JIT	0	0	0	0	0	0	0
F9L281PU	X9L281PU	8	APEE	AGE2ROM0	0	24	0	0	BKL1	PKE	20	0	0	0	0	0	0
F9L281PU	X9L281PU	8	APEE	AGE2ROM0	0	0	0	0	BKL2	PKE	4	0	0	0	0	0	0
F9L321PB	X9L321PB	8	APEE	AGE2ROM0	0	24	0	0	BKL1	JIT	0	0	0	0	0	0	0
F9L321PB	X9L321PB	8	APEE	AGE2ROM0	0	0	0	0	BKL1	PKE	7	0	0	0	10	0	0
F9L321PB	X9L321PB	8	APEE	AGE2ROM0	0	0	0	0	BKL1	SAT	0	0	0	0	0	0	0
F9L321PB	X9L321PB	8	APEE	AGE2ROM0	0	0	0	0	BKL2	PKE	17	0	0	0	11	0	0
F9L321PB	X9L321PB	8	APEE	AGE2ROM0	0	0	0	0	BKL2	SAT	0	0	0	0	0	0	0
F9L641PB	X9L641PB	8	APEE	AGE2ROM0	0	0	0	0	BKL1	JIT	0	0	0	0	0	0	0
F9L641PB	X9L641PB	8	APEE	AGE2ROM0	0	0	0	0	BKL1	PKE	0	0	0	0	0	0	46
F9L641PB	X9L641PB	8	APEE	AGE2ROM0	0	0	0	0	BKL2	JIT	0	0	0	0	0	0	0
F9L641PB	X9L641PB	8	APEE	AGE2ROM0	0	0	0	0	BKL2	PKE	0	0	0	0	0	0	3
CA081DA6	XA081DA6	6	APEE	AGA530_3	0	0	0	0	SFTY STK	DEL	0	0	0	0	0	0	0
CA124BB6	XA124BB6	6	APEE	AGUFLEX3	0	0	0	0	SFTY STK	DEL	0	0	0	0	0	0	0
CA124BC6	XA124BC6	6	APEE	AGUFLEX3	0	0	0	0	SFTY STK	DEL	0	0	0	0	0	0	0
CA130AFH	XA130AFH	6	APEE	AGA565_3	1	47	0	1	BKL1	DEL	1	0	0	0	0	0	0
CA130AFH	XA130AFH	6	APEE	AGA565_3	0	0	0	0	BKL1	PKE	47	0	0	0	0	0	0
FA148ABP	XA148ABP	8	APEE	AGCATAL4	0	31	0	0	BKL1	PKE	0	0	24	7	0	0	0
CA184AB6	XA184AB6	6	APEE	AGCATAL3	0	0	0	0	SFTY STK	DEL	0	0	0	0	0	0	0
CA185BB6	XA185BB6	6	APEE	AGCATAL3	0	0	0	0	SFTY STK	DEL	0	0	0	0	0	0	0
CA190ACH	XA190ACH	6	APEE	AGA530_3	0	220	0	0	BKL2	PKE	220	0	0	0	0	0	0
CA205ADH	XA205ADH	6	APEE	AGA565_3	0	402	0	0	BKL1	PKE	402	0	0	0	0	0	0
CA211EA6	XA211EA6	6	APEE	AGA565_3	0	27	0	0	SFTY STK	DEL	0	27	0	0	0	0	0
CA212BB6	XA212BB6	6	APEE	AGA565_3	0	0	0	0	SFTY STK	DEL	0	0	0	0	0	0	0
CA224CD6	XA224CD6	6	APEE	AGUFLEX3	0	1	0	0	SFTY STK	DEL	0	1	0	0	0	0	0
CA344ED6	XA344ED6	6	APEE	AGA565_3	0	0	0	0	SFTY STK	DEL	0	0	0	0	0	0	0
CA517AA6	XA517AA6	6	APEE	AGNOTEST	0	223	0	0	BKL1	PKE	1	0	0	0	1	0	1
CA517AA6	XA517AA6	6	APEE	AGNOTEST	0	0	0	0	BKL2	PKE	199	0	0	23	323	322	183
CA635AAH	XA635AAH	6	APEE	AGA530_3	0	21	0	0	BKL1	PKE	5	0	0	0	0	0	0
CA635AAH	XA635AAH	6	APEE	AGA530_3	0	0	0	0	DES STK	DEL	16	0	0	0	0	0	0
CA6AAC80	XA6AAC80	6	APEE	AGJ750A1	0	74	0	0	BKL1	PKE	26	0	0	0	0	0	0
CA6AAC80	XA6AAC80	6	APEE	AGJ750A1	0	0	0	0	BKL2	PKE	48	0	0	0	0	0	0