NORTHEASTERN UNIVERSITY
Graduate School of Engineering

Thesis Title: Reinforcement Learning Approach for Disassembly
Author: Harshawardhan Mukund Kulkarni
Department: Mechanical and Industrial Engineering
Approved for Thesis Requirement of the Master of Science Degree

Thesis Advisor, Dr. Sagar Kamarthi

Thesis Advisor, Dr. Ibrahim Zeid

Department Chair, Dr. Hameed Metgalchi

Graduate School Notified of Acceptance:

Director, Graduate School of Engineering,
Dr. Sara Wadia-Fascetti

8/13/2010
Date

8/15/2010
Date

April 16, 2010
Date

Aug 30, 2010
Date
REINFORCEMENT LEARNING APPROACH FOR DISASSEMBLY

A Thesis Presented

by

Harshawardhan Mukund Kulkarni

to

The Department of Mechanical & Industrial Engineering

In partial fulfillment of the requirements

for the degree of

Master of Science

in

Computer Systems Engineering

Northeastern University,
Boston, Massachusetts

August 2010
Abstract

Disassembly line is the best way to disassemble products with similar components in large quantity. Similar to an assembly line, a disassembly line consists of a series of workstations. Factors such as multiple demand arrivals, multiple arrivals of EOL products, different state of EOL products, fluctuations in the inventory and levels make a disassembly line very complex and difficult to control. Earlier researchers have proposed a Multi-Kanban mechanism for a disassembly line, which involves dynamic routing of the Kanbans to various workstations. This research focuses on applying reinforcement learning to the disassembly environment.

We have designed the reinforcement learning agent with 19 state variables – 5 component buffers, 9 assembly buffers and 5 demand variables. The agent selects its action from 14 possible actions which are related to either pulling an assembly for disassembly or pulling a component for satisfying a demand. The agent then receives a scalar reward based on the selected action and the current state of the system. The weights of the neural network are then updated. In this way, the neural network gets trained as the simulation progresses. We observed that the number of demands satisfied by the intelligent system is greater than that satisfied by the Multi-Kanban System. In addition work-in-process inventory in case of the intelligent system is much smaller than in case of the Multi-Kanban System.

Keywords: Reinforcement Learning, Q-learning, Elman Network, backpropagation.
Dedicated to my family and friends...
Table of Contents

Abstract

Table of Contents

List of Figures

List of Tables

1. Introduction........................................................................................................1
   1.1. Overview.......................................................................................................1
   1.2. Motivation....................................................................................................3
   1.3. Research Scope and Contributions..............................................................4
   1.4. Outline.........................................................................................................4

2. Literature Review................................................................................................5

3. Background of Disassembly..............................................................................9
   3.1. Disassembly..................................................................................................9
   3.2. Push and Kanban Pull Mechanism...............................................................14

4. Background of Machine Learning.................................................................16
   4.1. Reinforcement Learning...............................................................................19
   4.2. Elman Network.............................................................................................29
   4.3. Q-Learning..................................................................................................32
   4.4. Backpropagation.........................................................................................42

5. Disassembly Line with Multiple Precedence Relationships..........................47
   5.1. Introduction..................................................................................................47
   5.2. Objectives....................................................................................................49
5.3. Model Description.................................................................49

5.4. BP Algorithm for Q-learning.........................................................45

6. Simulation/Results.................................................................63

7. Conclusions............................................................................75

References
List of Figures

Figure 1. Basic Kanban mechanism in a production line......................................................17
Figure 2. Standard RL model................................................................................................20
Figure 3. Non-recurrent network..........................................................................................29
Figure 4. Recurrent network...............................................................................................30
Figure 5. Elman network......................................................................................................31
Figure 6. A fully connected multi layer perceptron.............................................................42
Figure 7. Computations within a single node.......................................................................44
Figure 8. Different types of transfer functions....................................................................45
Figure 9. Disassembly line with products having different precedence relationships........48
Figure 10. Kanban and material flows in a disassembly line..............................................54
Figure 11. The Multi Kanban mechanism............................................................................55
Figure 12. Workflow.............................................................................................................44
Figure 13. Elman network....................................................................................................58
Figure 14. Generalized neuron............................................................................................59
Figure 15. WIP vs. Time plot for RL simulation #1.........................................................65
Figure 33. WIP vs. Time plot for MK simulation #9

Figure 34. WIP vs. Time plot for MK simulation #10
List of Tables

Table 1. States .................................................................................................................. 35
Table 2. Actions ............................................................................................................... 35
Table 3. Q-Table ............................................................................................................. 36
Table 4. Components with their nomenclature ............................................................. 50
Table 5. Appliance arrival data ..................................................................................... 51
Table 6. Component data ............................................................................................... 52
Table 7. Reinforcement Learning agent ......................................................................... 63
Table 8. Multi-kanban mechanism ................................................................................ 64
Chapter One

Introduction

1.1 Overview

In recent years, with the technology of the products reaching new heights every day, more and more products are being dumped into the environment by the consumers to buy new, more advanced products which usually are available at a lower price than the existing ones. These practices of disposing the products reaching their end of life either through dumping in landfills or by shredding and incineration are reckoned to be too polluting and unnecessary wasting valuable environmental resources, by failing to retrieve and reuse materials and functional components potentially available in the discarded product. Realizing this, several countries have passed regulations that force manufacturers not only to manufacture environmentally conscious products, but also to take back their used products from consumers so that the components and materials recovered from the products may be reused and/or recycled. Disassembly plays an important role in product recovery.

Recycling household appliances is very popular in the recycling industry due to the large amount of recyclable materials and reusable components present in them. Many components in appliances could be refurbished to last longer than the products themselves. For example, electric motor from a dishwasher, washer or dryer can be rebuilt and reused several times. In order to obtain these materials and components for further reuse and recycle, manufacturer engages in a process called product and material recovery. During the process, reusable components and
recyclable materials are recovered from End-Of-Life (EOL) products by disassembly and/or shredding and mining.

Disassembly of household appliances has unique characteristics. Even though different appliances have different functions and appearances, they typically share similar structure, similar materials and similar disassembly processes. However, sequencing of an EOL appliance in a disassembly process depends on the precedence relationships of components in that appliance causing difficulties because of the different products that arrive at a disassembly system. Others difficulties include fluctuation in demand for components and fluctuation in the supply of EOL products. These difficulties must be addressed for the disassembly system to be effective.

Disassembly line is the best way to disassemble these kinds of products in large quantity. Similar to an assembly line, a disassembly line consists of a series of workstations. However, there are many characteristics that make a disassembly line unique in its own way. Factors such as multiple demand arrivals, multiple arrivals of EOL products, different state of EOL products, fluctuations in the inventory levels etc. make a disassembly line very complex and difficult to control. A good choice of control mechanism is required to make the disassembly line efficient.

In an assembly system, two types of control systems are usually implemented viz. push system and pull system. In a push system, the system produces products continuously and the inventory manages to cope with the fluctuation in demand. In a pull system, products are manufactured as and when the demand occurs. None of these systems are of practical use for a disassembly system.
Udomsawat and Gupta have proposed a Multi Kanban mechanism for disassembly, considering four scenarios: 1. Disassembly line with single type products, 2. Disassembly line with component discriminating demands, 3. Disassembly with products having multiple precedence relationships and 4. Sudden breakdown of a workstation. The mechanism is designed in such a way its implementation helps reduce the inventory buildup in the system that is commonly found in a disassembly line with push system implementation. The mechanism is able to ensure a smooth operation of the disassembly line where multiple types of appliances arrive for processing. The authors have given a set of rules for the routing mechanism of the Kanbans to the most desirable workstation for each scenario. We intend to develop an intelligent system which will pull the appropriate subassembly/assembly for disassembly using Reinforcement Learning. The goal is to reduce the work-in-process inventory and maximize the number of demands satisfied. We consider *disassembly of products with multiple precedence relationships*.

### 1.2 Motivation

A disassembly line is far more difficult complex than a normal assembly line. The inflow of raw materials (in this case, products) can occur at any workstation, depending upon the disassembly sequence. Also, the inflow is not at a fixed rate since the products are collected from trash, dumps, landfills etc. which are not a reliable source. A few products might not even be in a condition to yield any components from them. Some products have a small shelf life. Therefore, fluctuations in demand, supply and inventory, variable disassembly time, disassembly sequence, environmental concern and federal rules and regulations make a disassembly line much more complicated to control. A neural network can therefore be used to solve such complex systems.
Following the understanding behind the reinforcement learning, our interest would be to implement it into the disassembly environment. Therefore, ‘the more difficult to control’ disassembly environment can be controlled by robots, who can adjust their action or ‘learn’ to act according to various circumstances occurring in front of them. Based on which component is to be retrieved, the robot can be able to sequence its actions according to the configuration of the EOL product arriving at the workstation. We hope that this research helps the disassembling industry to improve its efficiency, reduce the work-in-process inventory as well as satisfy the demands of its consumers.

1.3 Research Scope and Contributions

Due to the complexities of a disassembly line, we limit the scope of this research to the following factors:

- Demands for components are stochastic and follow an exponential distribution.
- The arrival of supply of EOL products is stochastic and follows an exponential distribution.
- The components mix of a product is known.
- The disassembly time is stochastic and follows an exponential distribution.
- Comparisons of performance measures are for the proposed Reinforcement learning system with the Kanban System.

The primary contribution of this research is the application of reinforcement learning algorithm to a disassembly environment, to decide the assembly/subassembly to disassembled depending upon the demands for the components, time for disassembly and the total work-in-process inventory. Prior work in machine learning/neural networks has been done in the field of
disassembly sequence, determining feasible plans for a remanufacturing system, line balancing, and disassembly line scheduling etc. This is a totally new research area that is focused on the type of assemblies for disassembly with Elman neural network trained by reinforcement learning technique and error backpropagation.

1.4 Outline

This dissertation is organized in eight chapters.

Chapter 1 gives a brief overview of the dissertation along with Motivation, Research Scope and Contributions.

Chapter 2 presents the Literature Review on all related studies. The studies involved are disassembly, reinforcement learning, and neural networks.

Chapter 3 gives a background on disassembly, push and pull system.

Chapter 4 gives a background on Machine Learning techniques, Elman Neural Network, Q-learning and backpropagation and Chapter 5 states the problem statement and research objectives.

Chapter 6 presents the Disassembly Line with Multiple Precedence Relationships. Here, the subassemblies can travel both upstream and downstream in the disassembly line. We describe a disassembly line involving household appliances.

Chapter 7 presents the Simulation Results of the neural network and finally, Chapter 8 presents the conclusion of the dissertation.
Chapter Two

Literature Review

Research related to disassembly and its economics for material recovery has expanded over the years. Johar and Gupta[1] developed a mathematical approach to balance the inventory generated from a disassembly line depending on the shape and size of the components, amount of space available in the sorting area of the disassembly line. Udomsawat and Gupta[2],[3],[4] presented a Multi-Kanban mechanism for a disassembly environment for Disassembly line with single type products, Disassembly line with component discriminating demands, Disassembly with products having multiple precedence relationships and Sudden breakdown of workstation. McGovern and Gupta[5] reviewed the complexity theory and used to prove that Disassembly Line Balancing Problem is NP-complete, unary NP complete, and NP hard, necessitating specialized solution methodologies, including those from the field of combinatorial optimization.

Imtanavanich and Gupta[6] determined the optimal number of take-back EOL products for the DTO system using Genetic Algorithms. Al-Turki and Gupta[7],[8] presented a JIT system which uses and algorithm to dynamically and systematically manipulate the number of Kanbans in order to offset the blocking and starvation caused by the said factors during a production cycle. McGovern and Gupta[9] developed a two optimal algorithm to balance the part removal sequence and reuse the total number of workstations. Kizilkaya and Gupta[10] compared the performance of Dynamic Kanban System for Disassembly Line to the Modified Kanban System and found that DKSDL is superior. Kizilkaya and Gupta[11] modeled DKSDL using simulation
for disassembly of a voice recognition client unit and showed that DKSDL is a viable and practical solution technique to address complex product problems in the industry.

Kongar and Gupta[12] introduced genetic algorithm for a disassembly process, which starts with a set of solutions, called the population. New populations are chosen according to their fitness and this is repeated until a predefined condition is satisfied. The authors found that this method is practical, easy to use, considers the precedence relationships and additional constraints in the product structure. McGovern and Gupta[13],[15] used a genetic algorithm for disassembly line scheduling. McGovern and Gupta[14] found that genetic algorithms and HK-heuristics found optimal solutions quickly when compared to a greedy algorithm. Korugan and Gupta[16] suggested a single stage pull type control mechanism with adaptive Kanbans and state independent routing of the production information.

A lot of research has been done on machine learning and neural networks and their application on disassembly, disassembly sequence generation or a re-manufacturing system. Huang, Wang and Johnson [17] describe the economic analysis method of the disassembly process and then a method of using an artificial neural network for disassembly sequence generation. Song, Guan [18] discuss resource planning for a complex remanufacturing system. The authors use a machine learning approach by using rough set theory and then a reinforcement process to enhance its confidence. The network needs a training set to get trained. Bakker [19] illustrates backpropagation through Long Short Term Memory recurrent neural network model/critic. Qiao, Hou and Ruan [20] describe a Neural Network for selecting action in a mobile robot with Q-Learning. They use Q-table to store the Q-values.
In our case, the number of Q-values is not known since the number of states is not known. Hence, we would need on-line Q-learning which does not use tables. Shiraga, Ozawa and Abe [21] [22] [23] describe a reinforcement learning algorithm for neural networks with incremental learning ability using a Resource Allocating Network and Long Term Memory. Gaskett, Wettergreen and Zelinsky [24] discuss a wire-fitted Q learning Neural Network using advantage learning. Sun, Wang and Cheng [24] discuss Reinforcement Learning method for continuous state space for an Elman Network. Here, they use a dynamic Elman network to approximate the Q-value for a state-action pair. This paper uses online Q-learning (since look-up tables cannot represent the states) which is suitable for our problem.

In the field of remanufacturing, a lot of research has been carried to apply machine learning to it to solve the complexity of the disassembly environment. Shah, Gosavi and Nagi [25] have design a reinforcement learning algorithm for switching between the source of raw materials – ‘cores’, which are obtained from recycled products and ‘non-recycled’ or unused products to reduce the manufacturing costs. Reveliotis [26] discusses the uncertainty management and modeling using neuro-dynamic programming. The agent seeks to determine the level of disassembly so that the derived units will be directed either: (i) for remanufacturing/refurbishing and reuse; or (ii) for extraction and recycling of (some of) their materials; or, (iii) for disposal through dumping or incineration.

Our problem is concerned over selecting the appropriate assembly/disassembly for performing the disassembly operation based upon the work-in-process inventory of the entire system and the demand for each individual component.
Chapter Three

Background of Disassembly

3.1 Disassembly

In the past decade, technological advances in electronic data management and communications have spurred economic growth and improved people’s lives in countless ways. However, our growing dependence on electronic products both at home and in the workplace has given rise to a new environmental challenge: electronics waste. A recent study by The United States Environment Protection Agency\(^1\) shows that electronics already make up approximately 1 percent of the municipal solid waste stream. Research completed in Europe shows that electronics waste is growing at three times the rate of other municipal waste. To the extent possible, electronics waste should be prevented, and older electronics should be reused and recycled. Electronics such as Televisions and Monitors, Computers, Computer Peripherals, Audio/Stereo Equipment, VCRs, DVD Players, Video Cameras, Telephones, Fax and Copying Machines, Cellular Phones, Wireless Devices, Video Game Consoles etc. are being dumped into the environment due to new products that are available at a cheaper price and better quality. These End-of-Life products are disposed off either through dumping in landfills or by shredding and incineration are reckoned to be too polluting since there are hazardous materials, such as lead, mercury, and hexavalent chromium, in circuit boards, batteries, and color cathode ray tubes (CRTs). Televisions and CRT monitors contain four pounds of lead, on average (the exact amount depends on size and make). Mercury from electronics has been cited as a leading source

of mercury in municipal waste. In addition, brominated flame retardants are commonly added to plastics used in electronics. If improperly handled, these toxics can be released into the environment through incinerator ash or landfill leachate.

Disposing these products also leads to unnecessary wasting valuable environmental resources, by failing to retrieve and reuse materials and functional components potentially available in the discarded product. In 1998, over 112 million pounds of materials were recovered from electronics, including steel, glass, and plastic, as well as precious metals. Reusing and recycling the raw materials from end-of-life electronics conserves natural resources and avoids the air and water pollution, as well as greenhouse gas emissions, which are caused by manufacturing new products. Realizing this, several countries have passed regulations that force manufacturers not only to manufacture environmentally conscious products, but also to take back their used products from consumers so that the components and materials recovered from the products may be reused and/or recycled. Disassembly plays an important role in recovering components and valuable materials.

Now-a-days many manufacturers design the products considering remanufacturability. Some electronics manufacturers are accepting household electronics for recycling. In some cases, these services are provided free-of-charge. Asset management and recovery programs have been available to major corporations and large purchasers of electronic equipment for quite some time. Now, electronics manufacturers are beginning to offer similar services for households and small businesses. The United States is now giving tax incentives to individuals who donate computer equipment. The 21st Century Classrooms Act for Private Technology Investment encourages
large companies to donate computer equipment to public and private schools. These organizations can receive tax benefits when donating to a non-profit organization.

What is The United States Environment Protection Agency doing To Encourage Reuse, Recycling, and Greener Purchasing of Electronics?

EPA’s goal is to promote greater product stewardship of electronics. Product stewardship means that all who make, distribute, use, and dispose of products share responsibility for reducing the environmental impact of those products. We intend to work towards this goal in three ways: 1) increase reuse and recycling of used electronics, 2) ensure that management of electronics is safe and environmentally sound, and 3) foster a life-cycle approach to product stewardship, including environmentally conscious design, manufacturing, and toxics reduction for new electronic products. EPA is currently working with stakeholders in both the public and private sectors to meet these goals. In support of these efforts, EPA will be looking to streamline regulations and policies. We aim to make it easier and more cost-effective for consumers, retailers, recyclers, manufacturers, and governments at all levels to help divert these products into environmentally sound reuse and recycling, as well as reduce the environmental footprint of electronic product use.

In addition, EPA’s Design for the Environment Program (www.epa.gov/dfe) is working with electronics manufacturers to incorporate environmental considerations into product design. EPA’s Environmentally Preferable Purchasing Program (www.epa.gov/opptintr/epp) is helping federal agencies in the purchasing of environmentally preferable products and services, including electronics. Also, the Energy Star Program (www.energystar.gov) promotes energy-efficient products through its labeling and education program. EPA’s WasteWise Program is challenging
its almost 1,100 partners to set goals for reducing electronics waste (www.epa.gov/wastewise).

Finally, EPA’s Office of Solid Waste is supporting multistakeholder dialogues, collection pilots, public education, and international cooperation to foster greater awareness and coordination of electronics reuse and recycling issues. For more information about EPA’s efforts to encourage product stewardship for electronics, visit www.epa.gov/epr.

Buying Green

Environmentally responsible electronics use involves not only proper end-of-life disposition of obsolete equipment, but also purchasing new equipment that has been designed with environmental attributes. Think about this when purchasing new equipment, and ask your retailer or electronics supplier about environmentally preferable electronics. Households, companies, and governmental organizations can encourage electronics manufacturers to design greener electronics by purchasing computers and other electronics with environmentally preferable attributes and by requesting takeback options at the time of purchase. Look for electronics that:

• Are made with fewer toxic constituents
• Use recycled content
• Are energy efficient (e.g., showing the “Energy Star” label)
• Are designed for easy upgrading or disassembly
• Utilize minimal packaging
• Offer leasing or takeback options
• Have been recognized by independent certification groups (such as the Swedish TCO or Blue Angel) as environmentally preferable.
The National Recycling Coalition has assembled information on environmentally preferable procurement of electronics on their web site, at: www.nrc-recycle.org/Programs/electronics/index.htm.

Household appliances like dishwashers, ovens, refrigerators, computers, air conditioners etc. have unique characteristics. Even though different appliances have different functions and appearances, they typically share similar structure, similar materials and similar disassembly processes. Disassembly line is the best way to disassemble these kinds of products in large quantity. Similar to an assembly line, a disassembly line consists of a series of workstations. However, disassembly line differs from an assembly line in various aspects. Factors such as multiple demand arrivals, multiple arrivals of EOL products, different state of EOL products, fluctuations in the inventory levels etc. make a disassembly line very complex and difficult to control.

A typical disassembly line consists of a series of workstations oriented according to the disassembly sequence of the end-of-life (EOL) product. It differs from an assembly line in a number of ways. In an assembly line, base or the core of the product enters at the first workstation and various operations are performed on the subsequent workstations to produce the final product. The finished product comes off the line through the last workstation. Usually, demand exists for the product coming out of the last workstation only. In contrast, EOL products can arrive at any workstation of the disassembly line, depending upon the product configuration. Often, one or more components of the arriving EOL products are missing. Likewise, demands can arrive for any component at any workstation, depending on which one is demanded by the customer. These factors can lead to fluctuations in the inventory levels in a disassembly line.
3.2 Push and Kanban Pull Mechanism

Traditional push system

Traditional push system is typically employed in assembly and disassembly environments. This section will provide a concept of basic push system, types of push systems and advantages and disadvantages of traditional push system.

3.2.1 Basic Push System

The emphasis of push system relies on forecasting information of customer demands, suppliers lead time and production capacity. Using this information, the system generates “schedule” of where, when and how many products and material must be transported to. Materials in the system are processed in order to meet the scheduled demand. Push system also relies on software or tool such as MRP and MRP-II. Push system can be implemented in many types of production systems such as job shop, manufacturing cell, and manufacturing line.

One key characteristic of a push system is it tends to generate large amounts of inventory. The inventory level must be maintained in order to meet the schedule. Because materials are pushed through the manufacturing system, they are usually processed as soon as they arrive at a workstation and worker and/or machine are available. Push system is also called “made-to-stock” system as opposed to a pull system, “made-to-order” system.

Bonney, Zhang, Head, Tien and Barson (Bonney et al., 1999) presented a discussion and comparison between push and pull system. The authors examine systems’ performance under various conditions including small and large setup time using simulation.
3.2.2 Advantages and Disadvantages of Push System

Push system operations depend on amount of inventory stocked in the system. Thus, it is common for a push system to attain high customer service level. Also, push system does not require frequent setup time due to larger batch size. Additionally, push system usually provide high worker and machine utilization. It is also easy to implement.

Increasing in inventory level by using push system may cost in extra warehousing. In addition, it does not respond well with fluctuation in demand if the forecast is not accurate.

3.2.3 Kanban Pull System

Kanban is usually employed as a major tool to implement pull system. In this section, we will present background of the system, basic rules and functions of the system, and advantages and disadvantages of the system.

3.2.4 Origin

Kanban system was first systematically used in Toyota production system in 1953 (Ohno. 1988). The first Kanban was a piece of paper that carried three groups of information, viz., pick up information, transfer information, and production information. Taichi Ohno invented the Kanban using an idea from supermarket anti first implemented it in Toyota’s machine shop. Eventually in 1962, the Kanban system was implemented widely in all Toyota plants as the main operating method. The key objective of Kanban was to limit overproduction or inventory and to allow production to begin at the right time. In 1988, Ohno reported that Kanban, as the primary control mechanism, was accounted for Toyota’s 4.8 billion dollars annual production. Success of Kanban and Toyota production system brought attention from around the world.
3.2.5 Kanban mechanism

Six basic functions of Kanban are providing pick-up or transport information, providing production information, preventing over production and excessive transport, serving as a work order attached to goods, preventing defective products by identifying process causing the defective, and revealing existing problem and maintain inventory control (Ohno, 1988). Ohno also indicate six rules of Kanban as follow.

Rule 1. The later stage picks up number of products indicated in the Kanban from the earlier stage.

Rule 2. Earlier stage produces number of products in the exact quantity indicated in the Kanban.

Rule 3. No products are produced or transported without a Kanban.

Rule 4. Kanban must always be attached to the products.

Rule 5. Defective products are not transported to the subsequent stage.

Rule 6. Reducing the number of Kanban increases their sensitivity.

First and second rule of Kanban serves as a withdrawal order between stages, transporting order, and production order. Rule 3 through 6, when practiced faithfully, help expand role of Kanban in the production line. Without them, the system will never benefit control aspect of Kanban nor cost reduction.

Basic Kanban mechanism in a production line can be depicted in Figure 1 (Matta et al., 2005)
The production stages and the buffers of the system are represented by ellipses and triangles respectively. Parts flow from the first stage to the last stage in linear fashion. Finished products wail in the last buffer, buffer N until a demand for one part from customer arrives. The Kanban system controls flow of parts by sending authorization to each stage from the corresponding buffer. WIP is limited to the amount of Kanban in the system. FP, represents the buffer containing products that have been processed by stage i. and by k. the number of Kanbans at stage i. Flow of physical parts in stage lean he described as follow. Each (i-1)-stage product can be processed by stage I when there is at least one unit of Kanban in buffer I). If both of them are presented, they proceed through stage i together. When entering stage I. (i-1)-stage product released Kanban K to the previous stage, stage i-I. This way, information transmitted from the customer is carried to the upstream stage stage-by-stage. Hence, information delay help reduce surge in inventory. Amount of WIP in the system is equivalent to number of Kanbans in the system.

**Figure 1: Basic Kanban mechanism in a production line**
3.2.6 Advantages and Disadvantages of Kanban mechanism

The first key advantage from implementing Kanban is its inventory control aspect. As mentioned earlier, amount of WIP in the system is equivalent to the number of Kanbans in the system. Production line can benefit tremendously from reducing the inventory. Example of subsequent benefits from reducing inventory are case to locate products, reduction in warehousing cost and reduction in risk of unsold products.

The second advantage when implementing Kanban is the ability to keep track of manufacturing process to maintain quality of the products. Since the circulation of the Kanban must be stopped when problem arises, it is not difficult to pinpoint where the culprit is in the system.

Kanban is also easy to understand and implement. Visual information on the Kanban helps reduce confusion among workers. This information ensures accuracy of products quantity, and transporting origin and destination.
Chapter Four

Background of Machine Learning

4.1 Reinforcement Learning

4.1.1 What is RL (Reinforcement Learning)?

One of the important branches of computer science is AI (Artificial Intelligence). Machine learning is a subcategory of AI that becomes a hot research area recently.

Machine learning in general can be classified into three categories:

1. Supervised learning (SL): These are learning in which you know the input as well as the output.
2. Unsupervised learning (USL): These are learning in which you know the input but not the output.
3. Reinforcement learning (RL): This kind of learning falls between the above two categories, in which you have the input, but not output, instead you have "critic" – whether the classifier's output is right or wrong. RL is often the learning mechanism of an agent acting in an environment that tries to achieve the best performance over time by trial and error.
Figure 2 is a standard model of RL:

![Diagram of a standard RL model](image)

In this model the agent in the environment chooses an action $a_i$, obtains reward $r_i$, and switch from state $s_i$ to state $s_{i+1}$. The goal is to maximize long term reward, where $\gamma$ is called the discounting factor.

4.1.2 Central Features and Issues of RL

RL dates back to the 1960's, originated from the research of dynamic programming and Markov decision processes. Monte Carlo method is another source of RL methods, learning from stochastic sampling of the sample space. A third group of method that is specific to RL is the temporal difference method ($\text{TD}(\lambda)$), which combines the merits of dynamic programming and Monte Carlo method, developed in the 1980's mostly by the work of Sutton and Barto.

The classical bandit problem is an example of RL.
One of the important RL algorithms is the \textit{Q-learning algorithm} introduced in the next section.

Models of optimal behavior in RL can be classified into (1) Finite-horizon model; (2) Infinite-horizon discounted model and (3) Average-reward model.

To measure learning performance, criteria include (1) eventual convergence to optimal, (2) speed of convergence to (near-)optimality and (3) regret, which is the expected reward gained after executing the RL algorithm.

The three main categories of RL algorithms are: (1) dynamic programming, (2) Monte Carlo methods and (3) temporal difference methods.

Ad-hoc strategies used in balancing RL exploration/exploitation include greedy strategies, randomized strategies (e.g. Boltzmann exploration), interval-based techniques and more.

RL algorithms can also be classified into model-free methods and model-based methods. Model-free methods include Monte Carlo methods and temporal difference methods. Model-based methods include dynamic programming, certainty equivalent methods, Dyna, Prioritized sweeping/queue-dyna, RTDP (Real-Time Dynamic Programming), the Plexus planning system etc.

The following are some of the central issues of RL:

- Exploration vs. exploitation: This is illustrated in the bandit problem, in which a bandit machine has several levers with different payoff values. The bandit machine player is given a fixed number of chances to pull the levers. He needs to balance the number of
trials used to find the lever with the best payoff, and the number of trials used to pull a particular lever only.

- **Life-long learning:** The learning is real-time and continues through the entire life of the agent. The agent learns and acts simultaneously. This kind of life-long learning is also called "online learning".

- **Immediate vs. delayed reward:** A RL agent needs to maximize the expected long-term reward. To achieve this goal the agent needs to consider both immediate reward and delayed reward, and try not to be stuck in a local minimum.

- **Generalization over input and action:** In RL where a model-free method is used to find the strategy (e.g., in Q-learning), a problem is how to apply the learned knowledge into the unknown world. Model-based methods are better in this situation, but need enough prior knowledge about the environment, which may be unrealistic, and the computation burden is cursed by the dimension of the environment. Model-free methods, on the other hand, requires no prior knowledge, but makes inefficient use of the learned knowledge, thus requires much more experience to learn, and cannot generalize well.

- **Partially observable environments:** The real world may not allow the agent to have a full and accurate perception of the environment, thus often partial information are used to guide the behavior of the agent.

- **Scalability:** So far available RL algorithms all lack a way of scale up from toy applications to real world applications.

- **Principle vs. field knowledge:** This is the general problem faced by AI: a general problem solver based on the first principle does not exist. Different algorithms are needed to solve
different problems. Moreover, field knowledge are often beneficial and necessary to be added to significantly improve the performance of the solution.

The field of AI has produced numerous algorithms for learning. There are essentially two large subdivisions of learning (which apply to animals as well as agents): supervised learning and unsupervised learning.

*Supervised learning* is often accomplished using neural networks. In a supervised learning situation, an agent is given input/output pairs and, after many examples, it develops its own function that can decide what to do with a given input. For example, a computer hooked up to a camera could be shown a series of satellite photographs of a forest. Some of the pictures could contain tanks hiding in the trees, and others could be regular unoccupied forest. One photo after another, the agent is shown a picture and told whether or not tanks are present in the scene. Once the teaching process is done, a new picture is shown to the agent and it tries to identify whether or not a tank is present. This type of problem is ideal for neural networks. The agent in this situation is learning passively; that is, after each photograph is shown, it doesn't take an action or make any statements. It just sits back and learns.

Even more interesting than supervised learning is *unsupervised learning*. This type of agent receives feedback from each action it performs, which allows it to judge how effective the action was. The feedback is extracted from the environment, either through sensors or internal states such as counting. This feedback is then classified as a reward (or reinforcement). An algorithm decides the value of the reward, which can be either positive or negative. These built-in rewards are very similar to the instincts and feelings that guide humans and other animals. Small
samplings of reinforcements that guide your typical day are hunger, pain, enjoyment of food, and sensing cold temperatures.

There are three main advantages of reinforcement learning:

- Very little programming is required because the agent figures out the algorithm itself.
- If the environment changes, it doesn't need to be reprogrammed. Even if the agent design is altered, it will relearn the optimal algorithm.
- If the learning algorithm is properly designed, the agent is guaranteed to find the most efficient policy.

Reinforcement learning shines when given a complex problem. Any problem with many different states and actions—so many that it is complicated for humans to fathom—is ideal for reinforcement learning. In robotics, if you want to program a six-legged walking agent, you need to understand which direction each of the motors turns, you need to pay attention to the sensors that indicate leg position relative to the others, and you need to pay attention to a myriad of physical conditions such as balance. This can be downright complex because a simple pair of wires that are reversed could throw everything off. With reinforcement learning, the agent can sit there experimenting with different walking gaits, measure how far a gait has caused it to move, and the best gait will reveal itself with enough reinforcement. The user could then change the length of the agent legs, change motor sizes, and reverse wires; and the agent will readapt to the new hardware. If the walking-algorithm were manually programmed everything would need to be reprogrammed. Our problem of Kanban routing in a disassembly environment also involves a number of states and actions and hence applying reinforcement learning seems feasible.
There are two types of unsupervised reinforcement learning. The first requires a model of the world so it can make proper decisions. For example, a self-learning chess program would need to know the position of all the pieces and all the available moves to both players in order to make an informed decision. This can be complex because it needs to keep many statistics. The second type uses an action-value model, which creates a function to deal with different states. This is known as Q-Learning.

Reinforced learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals. Reinforced learning is of great importance in designing the behavior strategy of mobile robots which can act according to its environment and changes occurring in it. Robots nowadays are used in almost every facet of industry. Here, we focus on the use of robots in the disassembling facilities. Unlike an assembly line, where there is a pre-defined sequence for a particular product, a disassembly line is unpredictable and complex which makes it difficult to control. Some unique characteristics of the disassembly environment include labor intensiveness, shorter self lives of End-of-Life (EOL) products and components, environmental concern, strict rules and regulations, and sever fluctuation in demand, supply and inventory. The input of the disassembly environment, i.e. EOL products or subassemblies often arrives at the workstation in different combinations and configurations. Thus, to separate the components from the subassembly, different disassembly sequence needs to be implemented.

Reinforced learning deals with how agents (robots) can learn successful control policies by experimenting in their environment. Each instance the agent performs an action based on a state of the environment, it receives a ‘reward’ which is a numerical value assigned by a reward
function. The task of the agent is to perform a sequence of actions, observe their consequences and learn a control policy. The control policy we desire is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent. The agent doesn’t know what will be the next state after performing a specific action, or it doesn’t know about how beneficial its last move was. The action performed by the agent is independent of the past state(s) or action(s).

**Reinforcement Learning agent:**

The definition of a reinforcement learning agent consists of:

- A state representation,
- A reward function,
- An action selection control policy, and
- A learning algorithm for estimating the state (or action) values.

All these decisions are very important for the success of an RL agent. The above elements of the RL agent are described in the following sections.

**State of the system:**

The state of the system is described as a combination of 9 assembly buffers, 5 component buffers and 5 demand queues. These numbers are stored as integers and normalized when they are fed to the neural network for training as well as for selecting the action once the network is trained. The state space is thus continuous.

9 assembly buffers: B<sub>1</sub> to B<sub>9</sub> containing assemblies and subassemblies.

- B<sub>1</sub> is of the type A-B-C-E (Microwave Oven)
- B<sub>2</sub> is of the type A-B-C-D-E (Washer/Dryer)
• B₃ is of the type B-C-E
• B₄ is of the type B-C-D-E
• B₅ is of the type B-A
• B₆ is of the type C-E
• B₇ is of the type C-D-E
• B₈ is of the type C-B-A (Refrigertor)
• B₉ is of the type D-E

5 component buffers: B₁₀ to B₁₄ containing components.

• B₁₀ is of the type A
• B₁₁ is of the type B
• B₁₂ is of the type C
• B₁₃ is of the type D
• B₁₄ is of the type E

5 demand queues: B₁₅ to B₁₉

• B₁₅ stores the demand queue for component A
• B₁₆ stores the demand queue for component B
• B₁₇ stores the demand queue for component C
• B₁₈ stores the demand queue for component D
• B₁₉ stores the demand queue for component E

**Action space:**

The agent can select an action from an action space of 14 different actions. Each action specifies the type of assembly to be drawn for disassembly or the type of component to be discarded.
Action $A_1$ represents pulling an entity from $B_1$, $A_2$ represents pulling an entity from $B_2$, … and $A_{14}$ represents pulling an entity from $B_{14}$.

**Rewards:**

The agent receives a reward when an action is performed. The reward system is designed according to the state of the disassembly line in which the action has been performed.

$$r(S, A_i)$$ is reward the agent would receive when it selects action $A_i$ (1 $\leq$ i $\leq$ 14) in any state $S \text{ until a demand exists}$ in any demand queue is expressed as follows.

$$r(S, A_i) = R - I - C$$

where, $R$ is the reward value received by the agent due to the action selected, $I$ is the negative reward received proportional to the total WIP and $C$ is the negative reward received for each extra finished product produced. We consider WIP as sum of all components in the subassembly buffers and the component buffers. $R$ is computed by comparing the state variables for each selected action. $I$ and $C$ are given as follows:

$$I = 2(B_5 + B_6 + B_9) + 3(B_3 + B_7) + 4(B_4)$$

$$C = |a - C_A| + |b - C_B| + |c - C_C| + |d - C_D| + |e - C_E|$$

**Learning Algorithm:**

The neural network used for computing the Q values is trained according a Backpropagation algorithm which is described in Section 10.
4.2 Elman Networks \cite{25}

Static neural networks cannot meet the requirements of application engineering, because they lack the necessary dynamical characteristics. Therefore, recurrent networks have attracted more attention in recent years.

Elman Networks are a form of recurrent Neural Networks which have connections from their hidden layer back to a special copy layer (called context layer). This means that the function learnt by the network can be based on the current inputs plus a record of the previous state(s) and outputs of the network. In other words, the Elman net is a finite state machine that learns what state to remember (i.e., what is relevant). The special copy layer is treated as just another set of inputs and so standard back-propagation learning techniques can be used (something which isn't generally possible with recurrent networks).

As an example (adapted from that in BOOK Artificial Intelligence: A New Synthesis), consider an intelligent agent that is navigating a grid-world of cells. It can sense the cells to the north, south, east, and west of it directly, and then can move in one of these directions. However, in order to know what is in the diagonally adjacent cells (i.e., north-east etc) then the agent will need to remember the values from its last position (which will give some of the missing information, but not all). An Elman network can be devised that will accomplish this. Consider the following (non-recurrent) network for deciding on an action based on the inputs from the sensors:
Figure 3: Non-recurrent network

We can convert this into an Elman network that remembers previous state by adding a new set of inputs which are fully connected by recurrent links to the hidden layer outputs (but delayed by one unit of time):

Figure 4: Recurrent network

This should then (with proper training) be able to learn a suitable function from inputs and stored state to action.
The basic structure of Elman Network is illustrated in Figure 3. The Elman Network has four layers: input layer, hidden layer, context layer and the output layer. Adjacent layers are connected by adjustable weights. Basically, Elman networks are special type of feed-forward networks with additional memory. The weights between the context layer and the hidden layer make the network sensitive the history of input data, which is very useful in dynamic modeling.

![Elman Network Diagram](image)

**Figure 5: Elman network**

The notation used in this article is as given below:

- $u_{ij}$: The weight that connects node $i$ in the input layer to the node $j$ in the hidden layer.
- $v_{jl}$: The weight that connects node $l$ in the context layer to the node $j$ in the hidden layer.
- $w_{kj}$: The weight that connects node $k$ in the output layer to the node $j$ in the hidden layer.
- $x_i, Q_k$: Inputs and Outputs of the Elman neural network, where $i = 1, 2, \ldots, n$ and $k = 1$.
- $h_j$: Output of the hidden node $j$, where $j = 1, 2, \ldots, m$.
- $y_j$: Input of the context layer, i.e. output of the hidden layer at $(t-1)^{th}$ iteration.
Each node of the hidden layer is connected to an additional layer, called the context layer, as shown in the figure above. This context layer is then fully connected back to the hidden layer in a forward manner, i.e. there exists a weight from every context node to every hidden node. The context layer behaves as it is an input layer, the only difference being, it gets its values from the hidden layer. Therefore, the network behaves like a feedforward network while computing the outputs of the hidden layer and the output layer. All the weights in the network get trained by using back-propagation. After calculating the outputs of each hidden node, the values get copied into each hidden node, which are used in the next iteration.

4.3 Q-Learning

While there are a few different types of learning, reinforcement learning normally helps adjust our physical actions and motor skills. The actions an organism perform result in feedback, which in turn is translated into a negative or positive reward for that action. As a baby learns to walk it may fall down and experience a little pain. This negative feedback will help the baby learn what not to do. If the baby has been able to stay standing for a matter of time, then it is doing something right and that achievement of the goal is a positive feedback: another positive feedback may be it being able to access something it desired. As the baby continues to try to walk, it will develop motor coordination in such a way that reward will be maximized. Pain, hunger, thirst, and pleasure are some examples of natural reinforcements. Actions can either result in immediate reward or be part of a longer chain of actions eventually leading to reward.

Q-learning [Watkins, 1989] is a recent form of Reinforcement Learning algorithm that does not need a model of its environment and can be used on-line. Therefore, it is very suited for repeated games against an unknown opponent. Q-learning algorithms work by estimating the values of
state-action pairs. The value $Q(s, a)$ is defined to be the expected discounted sum of future payoffs obtained by taking action $a$ from state $s$ and following an optimal policy thereafter. Once these values have been learned, the optimal action from any state is the one with the highest $Q$-value. After being initialized to arbitrary numbers, $Q$-values are estimated on the basis of experience as follows:

1. From the current state $s$, select an action $a$. This will cause a receipt of an immediate payoff $r$, and arrival at a next state $s'$.
2. Update $Q(s, a)$ based upon this experience as follows:

   Small changes in $Q(s, a) = x[r + y \max Q(s', b) - Q(s, a)]$

   where $x$ is the learning rate and $0 < y < 1$ is the discount factor.

3. Go to 1.

This algorithm is guaranteed to converge to the correct $Q$-values with the probability one if the environment is stationary and depends on the current state and the action taken in it called Markovian, a lookup table is used to store the $Q$-values, every state-action pair continues to be visited, and the learning rate is decreased appropriately over time. This exploration strategy does not specify which action to select at each step. In practice, a method for action, called the Boltzmann distribution strategy, is usually chosen that will ensure sufficient exploration while still favoring actions with higher value estimates. Experiments with Q-learning agent have been done in the past with favorable results. Bradke(1993) described encouraging results in applying multi-agent reinforcement learning to efficiently damp out disturbances of a flexi beam. Littman and Boyan (1994) describe a distributed RL algorithm for packet routing. Littman
(1994) also describes experiments with Q-learning agents that try to learn a mixed strategy that is optimal against the worst possible opponent in a zero-sum 2-player game. So, we can see that sufficient work has been done and even more underway to achieve our goal of "learning" artificial intelligence.

**Q-learning** is a specific kind of reinforcement learning that assigns values to action-state pairs. The state of the organism is a sum of all its sensory input, including its body position, its location in the environment, the neural activity in its head, etc. So in Q-learning, this means that because for every state there are a number of possible actions that could be taken, each action within each state has a value according to how much or little rewards the organism will get for completing that action (and reward means survival).

*The Seesaw Task*

Now let’s take this Q-learning approach to the seesaw task. (Remember, the goal is for the seesaw to keep the free wheel continually moving without it stopping in the middle or falling off the edges.)

The only sensory input (feedback) that the seesaw is getting from the environment is from the blue light sensor. In our seesaw task, we know that on average, it takes about 4000 milliseconds (ms) for the wheel to pass the light sensor once and then pass it again (roll back and forth once, called a “period”). When the light sensor detects the wheel (by sensing a decrease in the ambient light value), it decides when, from the point of detection, to flip the seesaw. It could flip immediately after the wheel passes or pause for any number of milliseconds before flipping. The decision on how long to pause is based on how long the wheel took to complete one period.
Without learning, this choice is arbitrary, but with learning, the seesaw may, for example, let the wheel roll further before flipping to allow it to gain more momentum if the period was short (meaning the wheel had little momentum and didn’t travel far along on the seesaw before returning). The closer the period length gets to the average (4000 ms), the more reward is received.

Because reward is assigned to a state-action pair (a specific action is chosen depending on the state), let’s lay out what the states and actions are: The state of the system must be determined by the input and previous state. By obtaining readings of when the wheel passes, the light sensor can measure how long a period length is by subtracting the first reading from a second. This number could be 0 or greater. Instead of having an infinite amount of different states, to make things simpler, we have divided this range into six intervals…

<table>
<thead>
<tr>
<th>State</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt; 1500</td>
</tr>
<tr>
<td>2</td>
<td>1500 – 2000</td>
</tr>
<tr>
<td>3</td>
<td>2000 – 2500</td>
</tr>
<tr>
<td>4</td>
<td>2500 – 3000</td>
</tr>
<tr>
<td>5</td>
<td>3000 – 3500</td>
</tr>
<tr>
<td>6</td>
<td>&gt; 3500</td>
</tr>
</tbody>
</table>

We have also defined four actions that can occur as a result of being in a certain state…
Table 2: Actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Pause for …</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 ms</td>
</tr>
<tr>
<td>2</td>
<td>100 ms</td>
</tr>
<tr>
<td>3</td>
<td>200 ms</td>
</tr>
<tr>
<td>4</td>
<td>300 ms</td>
</tr>
</tbody>
</table>

Because all combinations of states and actions are assigned Q-values, the Q-table will look something like this…

Table 3: Q-Table

<table>
<thead>
<tr>
<th></th>
<th>Action 1</th>
<th>Action 2</th>
<th>Action 3</th>
<th>Action 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When the seesaw finds itself in a certain state, it will choose an action based on each action’s Q-value. The higher the Q-value, the more chance that action has of being chosen. Now, let’s look at the action Q-learning equation for determining Q-values…

\[
Q(a,i) = Q(a,i) + L(R(i) + Q(a_{i+1},i) - Q(a,i))
\]
Where the following is true:

\[
\begin{align*}
Q &\quad \text{table of Q-values} \\
a &\quad \text{previous action} \\
i &\quad \text{previous state} \\
j &\quad \text{the new state resulting from the previous action} \\
a_{1} &\quad \text{the action that will produce the maximum Q-value} \\
L &\quad \text{the learning rate (between 0 and 1)} \\
R &\quad \text{the reward function}
\end{align*}
\]

This may look complicated, but once you look at each piece of the equation it makes fairly good sense how it all fits together. This equation is calculated after the robot completes an action. Unlike the real environment, but necessary for such an algorithm, everything, which occurs in the environment and the actions that a robot takes, are divided into steps. Think of it like a chessboard as opposed to a real war. There are a finite amount of actions one can take in chess and it always proceeds in a linear fashion. First, one player moves one piece, than the opponent moves one piece. In Q-learning, during one step, the robot analyzes data from the environment, chooses an action, and evaluates that action according to our equation.

Let’s turn on the seesaw and watch it learn. While it’s running, read the following description on how it implements the Q-learning equation. Turn on the RCX and press the green “run” button. Make sure the wheel is at a position where it will begin rolling.

The learning program cycles on a round of three passes of the wheel in front of the light sensor:

1- On the first pass, a start time is recorded, and new Q-values are determined from measuring the length of the 3\textsuperscript{rd} to 1\textsuperscript{st} pass.
(a lower tone sounds every time a first pass is made)

2- On the second pass, the start time is subtracted from the current time to determine how long one “period” took (the wheel rolling once back and forth). This is the state of the environment, and from this measure, the best action is determined.

(a very high tone sounds if a positive reward is given)

3- On the third pass, the seesaw implements the new action (delay before flipping) and begins a timer that will be used to determine the seesaw’s “reward” when it cycles back around to the first pass.

(a higher tone sounds every time the third pass is made)

When the wheel passes the light sensor for the very first time, no reward is calculated, but every time after, the actual period length is subtracted from the goal period length (4000 milliseconds). If the actual period length is within 200 milliseconds of the goal length, the action the seesaw took is positively reinforced.

Let the seesaw run for at least 10 minutes, observing when reinforcement takes place by listening to the tones. Can you tell if the seesaw has changed its performance from when it started? Try this demonstration a number of times, and observe if it acts differently on different runs.

Look back at the Q-learning equation. Each time a reward is given, this equation is used to figure out the new Q-value for the specific action-state pair. The first parameter signifies the old Q-value. Because, in our example, a Q-value can only get higher, we will be adding onto the old value. The learning parameter (L) is there to determine how fast a value can increase (the higher a value is, the more chance there is that it will be chosen). We don’t want the seesaw creating
the highest value immediately, because there may be a better solution within that state that just hasn’t been tried, or there may have been a fluke that caused the wheel to perform better than it usually would in that state. \( Q(a_t, i) \) is a very important parameter which looks ahead at the next set of actions that can take place if such an action is chosen. There may be a good \( Q \) value for a certain action, but if that action takes the system to a place with very low \( Q \) values, it hasn’t done well. By looking ahead, the equation can learn to determine if the action it takes will continue the system on a good or bad path. Over time the seesaw should learn which state-action strategies work best for keeping the wheel rolling on a period as close to 4000 ms as possible.

**Machine vs. Human**

Let’s take a moment to compare how you would do on such a task, versus what we’ve seen the program do…

1. Turn the RCX on, and press the green “run” button.
2. Change to manual mode by pressing the touch sensor connected to port “2”.
3. Now, watch the wheel carefully press the touch sensor connected to port “3” when you think the seesaw should flip. Remember, you don’t want the wheel to hit the ends or stop in the middle. (There will be a beep whenever the wheel crosses the light sensor’s path.)

How did you do? Pretty easy right? Humans have the skills necessary to quickly learn the system and make predictions about the behavior of the wheel. But as opposed to our rich sensory input, machines in general have much more limited information to work with. The only
input this seesaw has is the light sensor telling it when the wheel is passing. To simulate this experience, close your eyes and try the same task again, this time only listening to the beeps to make judgments about the wheel...

Not so easy, huh? Where we fail, the program in the computer is good: doing calculations and quickly computing probabilities. But regardless, it receives very little input, and more input would most likely help with the task. After all, you became adept very quickly at keeping the wheel moving when your eyes were open, much fast than the learning program.

The demonstration you just participated in should get you to think about and comprehend the gap of complexity between humans and machines such as this one. Even though we both can make use of reinforcement learning, our intrinsic characteristics affect the results. Of course, there are many machines which are much more complex than this simple seesaw, but as of yet, none are even close to reaching such biological complexity of even simpler animals.

When Q-learning, as well as other types of reinforcement learning is reproduced in a robot or a synthetic system of some sort, scientists typically use a formula that quantifies (puts numbers to) sensory input, the value of certain actions, etc. Biological systems don’t use abstract numbers like a computer program but instead use neuronal population behaviors and chemicals called neurotransmitters for forming action-state connections and reward.

Take some time to think about some ways in which the seesaw could be improved to help with the task. Would touch sensors help? More light sensors?
**Conclusion**

Learning is a necessary function for a system which can grow and adapt to its specific environment. We’ve seen how reinforcement learning uses feedback from the environment to develop patterns over time which has proven to provide reward and resources for the system. While the seesaw demonstration is a very simplified version of what truly takes place, we are still able to learn many of the basic principles of the interactions between an organism and environment using a simple example.

Structure of an ANN can be classified into 3 groups as per the by arrangement of neurons and the connection patterns of the layers: feedforward (error backpropagation networks), feedback (recurrent neural networks and adaptive resonance memories), self-organizing (Kohonen networks). Also neural networks can be roughly categorized into two types in terms of their learning features: *supervised* learning algorithms, where networks learn to fit known inputs to known outputs, and *unsupervised* learning algorithms, where no desired output to a set of input is defined. The classification is not unique and different research groups make different classifications.

The feedforward neural networks consist of three or more layers of nodes: one input layer, one output layer and one or more hidden layers. The input vector $x$ passed to the network is directly passed to the node activation output of input layer without any computation. One or more hidden layers of nodes between input and output layer provide additional computations. Then the output layer generates the mapping output vector $z$. Each of the hidden and output layer has a set of connections, with a corresponding strength-weight, between itself and each node of preceding layer. Such structure of a network is called a *Multi-Layer Perceptron* (MLP).
The feedback neural networks have loops that feedback information in the hidden layers. In Self-Organising Feature Maps (SOFM) the multidimensional input space is mapped into two or three dimensional maps by preserving the necessary features to be extracted or classified. An SOFM consists of an input layer and an output map. Some of the commonly used feedforward and feedback neural networks are briefly discussed below.

**4.4 Backpropagation**

The error backpropagation network (EBP) is one of the most commonly used types of neural networks. The EBP networks are widely used because of their robustness, which allows them to be applied in a wide range of tasks. The error backpropagation is the way of using known input-output pairs of a target function to find the coefficients that make a certain mapping function approximate the target function as closely as possible.
The task faced by a backpropagation neural network is that of learning supervised mapping: given a set of input vectors and associated target vectors, the objective is to learn a rule that captures the underlying functional relationship between the input vectors and the target vectors.

Mathematically, each target vector \( \vec{z} \) is a function, \( f \), of the input vector \( \vec{x} \):

\[
\vec{z} = f(\vec{x})
\]

The task of the backpropagation network is to learn the function \( f \). This is achieved by finding regularities in the input patterns that correspond to regularities in the output patterns. The network has a weight parameter vector, whose values are changed to modify a function \( f \) computed by the network to be as close as possible to \( f \).

The backpropagation network operates in two modes: mapping and learning. In mapping mode, each example is analysed one by one and the network estimates the outputs based on the values of the inputs. For every example, each input node passes a value of an independent variable \( x_i \) to all the nodes of the hidden layer. Each hidden node computes a weighted sum of the input values based on its weights. The weights are determined during the learning mode. Finally, from this value of the weighted sum, the hidden nodes compute a sigmoid output \( y_i \) of the hidden nodes. The sigmoid function provides a bounded output of the hidden node. Each of the output nodes receives the outputs of the hidden nodes \( y_i \), computes a weighted sum of the inputs based on the weights \( b_i \) and finally, determines the sigmoid output \( z_i \) of the node. The output of the output node, \( z_i \), is the estimated value of the \( i^{th} \) dependent variable. The output from the output node is compared with the target output and the error is propagated back to adjust the connecting weights \( a \) as well as \( b \) and this procedure is called backpropagation.
For an MLP, given the input vector $X=(x_1, x_2, ..., x_n)$, the output from the hidden node will be as follows:

$$y_j = g(u) = g(a_{0j} + \sum_{j=1}^{N_{in}} a_{ij} x_i)$$

Where $j=1..N_{in}$ and $a_{ij}$ is the weight of the $i^{th}$ node for the $j^{th}$ input. The outputs from the hidden nodes would be the input to the next hidden layer (if there is more than one hidden layer) or to the output nodes. The outputs of the output nodes should be calculated as follows:

$$z_k = g(b_{0k} + \sum_{j=1}^{N_{out}} b_{jk} y_j)$$

Where $k=1..N_{out}$ and $b_{jk}$ is the weight of the $j^{th}$ node for the $k^{th}$ output. The transfer function, mostly used a sigmoid or a logistic function (Figure 4), gives values in the range of $[0,1]$ and can be described as:

$$g(u) = \frac{1}{1 + e^{-u}}$$

\[\text{Figure 7: Computations within a single node}\]
The mean square error is the way of measuring the fit of the data and is calculated as:

\[ E = \frac{\sum_{n=1}^{N} \sum_{k=1}^{K} (z_{kn} - t_{kn})^2}{2NK} \]

where \( N \) is the number of examples in the data set, \( K \) is the number of outputs of the network, \( z_{kn} \) is the \( k \)-th actual output for the \( n \)-th example and \( t_{kn} \) is the \( k \)-th target output for the \( n \)-th example. For more details see Smith (1993).

In the learning mode, an optimization problem is solved to decrease the mean square error and it finds such a value for \( a \) and \( b \) to bring the \( E \) to minimum. By solving the optimization problem and knowing the slope of the error surface, the weights are adjusted after every iteration. As per the gradient descent rule the weights are adjusted as follows:

\[ \Delta w(\xi) = -\eta \frac{\partial E}{\partial w} + \mu \Delta w(\xi - 1) \]
where, \( \eta \) is the learning rate and \( \mu \) is the momentum value.
Chapter Five

Disassembly Line with Multiple Precedence Relationships

5.1 Introduction

In this chapter, we investigate the complexity of a disassembly line. The EOL products entering the disassembly line have multiple precedence relationships. To illustrate this phenomenon, we consider household appliances such as refrigerator, microwave ovens, washer/dryer, air conditioners and vacuum cleaners for disassembly. These household appliances are used for different purposes and have different layouts and structures. Yet, they have many components in common. Parts such as solenoid valve, motor/compressor, copper tubing, circuit boards, steel frames, LED displays and fuses can be found in all these appliances. However, the disassembly sequence of each appliance may differ from the other in some way. Therefore, different appliances enter the disassembly line at different workstations, depending upon its disassembly sequence. Figure 7 below illustrates a disassembly line with products having different precedence relationships.
The location of the EOL products where it enters the disassembly line depends upon the precedence relationships among. The input location of a product would be the workstation that disassembles its first component in the disassembly sequence. Each component has a different disassembly time. At each workstation, one component is disassembled, except when there are only two components remaining in the subassembly. Each workstation has two types of buffers: a subassembly buffer and a component buffer. Out of the available subassemblies, the subassembly which will create minimum extra inventory, need minimum time for disassembly or satisfy demand for the most wanted component at the earliest is selected. The demand arrival is assumed to follow an exponential distribution, which is described in the next section.
5.2 Objectives

The primary objective of this chapter is to convey the complexities involved in a disassembly environment. We compare the results of our simulation with the Multi Kanban System considering the work in process inventory and the number of demands satisfied as the performance measures.

5.3 Model Description

Arrival Pattern of EOL Appliances

In a disassembly line, the arriving products may contain different configurations of the components from its base number of components. If a product has \( N \) number of base components, the possible combinations \( Q(N) \) is given by

\[
Q(N) = 2^N - N - 1
\]

Here, we consider microwave over, washer/dryer, refrigerator, electric/gas range, dishwasher, water heater and vacuum cleaner. These appliances contain reusable components such as electric motors, compressors, circuit boards, thermostatic switches as well as recyclable materials such as iron, copper and aluminum. Table 1 lists the components with their nomenclature.

---

Table 4: Components with their nomenclature

<table>
<thead>
<tr>
<th>Appliance</th>
<th>Components</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Metal frame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Aluminum component /</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microwave Oven</td>
<td>Copper piping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Washer/Dryer</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>Refrigerator</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Electric / Gas</td>
<td>Range</td>
<td>●</td>
<td>●</td>
<td>-</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Dishwasher</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Water Heater</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>-</td>
</tr>
<tr>
<td>Vacuum Cleaner</td>
<td></td>
<td>-</td>
<td>-</td>
<td>●</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

According to the formula given above, the possible number of combinations for Washer/Dryer is 26. The workstation where the product enters the disassembly line depends on the combination. We assume that component A is disassembled at workstation 1, component B at workstation 2, component C at workstation 3 and components D and E at workstation 4. The disassembly sequence of the washer/dryer is A-B-C-D-E. Therefore, if a product consists of components B, D and E will enter the disassembly line at workstation 2, where component B is removed first, followed by the extraction of D and E. A product with missing component C will skip workstation 3. These situations can destabilize the disassembly line, leading to overflow of
materials (components or subassemblies) at one workstation, while starving the other. We consider three EOL products – microwave oven, washer/dryer and refrigerator as shown in Table 2.

Table 5: Appliance arrival data

<table>
<thead>
<tr>
<th>Appliance</th>
<th>Disassembly Sequence</th>
<th>Mean Arrival Rate (units/hour)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microwave Oven</td>
<td>A-B-C-E</td>
<td>15</td>
</tr>
<tr>
<td>Washer/Dryer</td>
<td>A-B-C-D-E</td>
<td>15</td>
</tr>
<tr>
<td>Refrigerator</td>
<td>C-B-A</td>
<td>15</td>
</tr>
</tbody>
</table>

**Demand Arrival**

The arrival of demand in a disassembly line is multi-level as opposed to an assembly line. Demand can arrive for any component. Demand for component A arrives at workstation 1, for B at workstation 2, for C at workstation 3 and for components D and E at workstation 4. We consider inter-arrival time of two consecutive demands for each component to follow an exponential distribution. The demand for each component is as shown in Table 3.
<table>
<thead>
<tr>
<th>Component Type</th>
<th>Mean Demand Arrival Rate (units/hour)</th>
<th>Mean Disassembly Time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>D</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

**Buffers**

There are three types of buffers in the system – component (output) buffer, internal subassembly (input) buffer and external product (input) buffer. The three products considered here are Microwave Oven (disassembly sequence: A-B-C-E), Washer/Dryer (disassembly sequence: A-B-C-D-E) and Refrigerator (disassembly sequence: C-B-A). Therefore, Microwave Oven and Washer/Dryer will only enter at workstation 1 and Refrigerator will enter only at workstation 3. Workstation 1 would therefore have only the external product buffer for its input. Workstation 2 would have only the internal subassembly buffer, which would be routed from workstation 1 or workstation 3. Workstation 3 would have both internal subassembly as well as external product buffer, whereas workstation 4 would have the internal subassembly buffer only. Each workstation would consist of a component buffer for its output (i.e. individual components A, B, C, D or E) except for workstation 4 which would have two buffers, one for component D and the other for component E.
5.3.1 Multi Kanban Routing mechanism\(^4\)

Consider workstation \(j\), where \(1 \leq j \leq N-1\). When a demand for component \(j\) arrives at the component buffer of workstation \(j\), one unit of component \(j\) is retrieved and the component Kanban \(j\) attached to it is routed to the most desirable workstation. The procedure for determining the most desirable workstation to route component Kanban \(j\) is given below. (Note that this procedure is not applicable to component Kanbons \(N-1\) and \(N\). In both cases the Kanbons are routed to the input buffer of the last workstation).

A component Kanban originating from workstation \(j\) will be routed to a workstation \(i\), where \(1 \leq i < j\), or workstation \(j\) depending on the availability and the desirability of the subassembly that contains component \(j\). Routing component Kanban \(j\) to workstation \(i\), where \(1 \leq i \leq (j-1)\), will result in an immediate separation of component \(j\) from component \(i\). Thus, the only subassembly located at the input buffer of workstation \(i\) that would be useful is a subassembly that contains only components \(i\) and \(j\). If this type of subassembly exists in the input buffer of workstation \(i\), then workstation \(i\) is qualified. Similarly, if there is at least one subassembly in the input buffer of workstation \(j\), then workstation \(j\) is qualified.

Next, we need to select the most desirable workstation to route component Kanban \(j\) to, among the qualified ones, such that, if chosen, will cause the least amount of extra inventory in the system. Choosing workstation \(i\) will increase the inventory level of component \(i\) by an additional unit. Thus, the best workstation \(i\) is the one that is most starving for its component. By checking the backorder level for demand \(i\), we could determine the most starving workstation. If there is a tie, select the most downstream workstation. Choosing workstation \(j\) will create a residual subassembly that will be further disassembled at downstream workstations. If workstation \(j\) is
chosen, then a proper subassembly must be chosen to disassemble. For example, if a backorder exists at the component buffer of workstation $k$, where $j < k \leq (N-1)$, then, if available, we might try to disassemble a subassembly that contains only components $i$ and $k$. If more than one workstations qualify as starving workstations, then the one that is most starving among them is chosen. If there is a tie, then the most downstream workstation is selected.

![Diagram of Kanban and material flows in a disassembly line](image)

**Figure 10: Kanban and material flows in a disassembly line**

We can now compare the starving levels of workstations $i$ and $j$. If the highest starving level of workstation $i$ is greater than or equal to the starving level of workstation $j$ then we will route the component Kanban $j$ to workstation $i$, otherwise, we will route it to workstation $j$. Note that whenever an external subassembly is available, it will always be chosen first. Internal subassemblies will only be used when no external subassembly of the desired kind is available.
Subassembly Kanbans are routed in a fashion similar to component Kanbans. Figure 4 shows a concept of the Multi Kanban Mechanism.

For a product whose precedence relationships do not coincide with the disassembly sequence of the line, we may need to route the Kanban to one of downstream workstations (workstation $k$, where $j < k \leq (N-1)$) depending on whether it meets the above described criteria or not. We select the best destination using the same criteria as for the regular products that are routed in only the downstream direction. In other words, there are more choices to select from when allowing mixed products with multiple precedence relationships into the system. However, the selection criteria remain the same regardless of Kanban routing direction.

![Figure 11: The Multi Kanban mechanism](image)
5.3.2 RL Agent

Whenever a demand for a component arrives in the respective demand queue, the agent should select an action, such that, the demand gets satisfied as soon as possible. For example, if a demand for component B arrives, the agent should satisfy it by pulling a component from the component buffer ‘B’ or pull a subassembly from the input buffers (buffers B₃, B₄ or B₅ in this case). If the input buffers of workstation II are all empty, the agent should pull an assembly from buffer B₁, B₂ or B₈. Since the disassembly time for component A and C is the same, pulling of a particular assembly would not make a difference. The assembly accumulated to the highest level should be selected. The disassembly of this pulled assembly will then supply a subassembly to buffers B₃, B₄ or B₅. The supplied subassembly is again disassembled to retrieve component B. the agent’s task is therefore to supply the component to the respective component buffer if it is empty in the first place. After this, the agent would perform action A₁₁ which will fetch the maximum reward. The agent will be performing the disassembly operations as long as a demand exists in any of the demand queue.
Figure 12: Workflow
5.4 BP Algorithm for Q-Learning

Step 1: Initialize weights $u_{ji}$, $v_{ji}$ and $w_{ij}$ to small random values and $y_j(0) = 0$.

Step 2: Perform steps 3–15 until $Q_1(t+1) - Q_1(t) \geq \varepsilon$, some small threshold value.

Step 3: Present input pattern $X(t) = (S(t), a_g(t))^T \in \mathbb{R}^n (n = 23)$ to the input layer, where,

$$S(t) = (s_1, s_2, ..., s_{19})^T$$

is the system state vector, $A = \{a_1, a_2, ..., a_{14}\}$ is the action space and $a_g(t) = (b_1, b_2, b_3, b_4)$ is the binary representation of 14 possible actions ($g \in A$).

Action $a_g(t)$ is selected according to the approximately greedy and continuously differentiable Boltzmann-Gibbs distribution.

Figure 13: Elman network
\[ g(t) = \arg\max_{x \in \mathcal{A}} \left( \text{prob}(x) \right) \]

Where, \( x \) is calculated by the following equation:

\[
\text{prob}(x) = \frac{\exp\left( \frac{Q(S(t), x)}{T_t} \right)}{\sum_{x=1}^{14} \exp\left( \frac{Q(S(t), x)}{T_t} \right)}
\]

where, \( T_t > 0 \) is a temperature parameter that controls the stochastic degree of action selection. The action with the highest probability is selected for the input layer.

Step 4: Equate the activation of each input node to \( x_i(t) \) where,

\[ x_1 = s_1, x_2 = s_2, \ldots, x_{19} = s_{19} \text{ and } x_{20} = b_1, x_{21} = b_2, x_{22} = b_3, x_{23} = b_4. \]

Figure 14: Generalized neuron
Step 5: Equate the inputs of the context layer to \( h_j(t-1) \) such as

\[
    z_i(t) = h_i(t-1),..., z_m(t) = h_m(t-1).
\]

Step 6: Compute the output field of each the hidden node \( j \)

\[
    p_j(t) = \sum_{i=1}^{n} u_{ji}(t)x_i(t) + \sum_{l=1}^{m} v_{jl}(t)y_l(t)
\]

where, \( u_{ji}(t) \) and \( v_{jl}(t) \) are the connecting weights between the input layer and hidden layer and, the hidden layer and context layer respectively as shown in Figure 2; \( m \) is the number of nodes in the context layer(same as the hidden layer).

Step 7: Compute the activation of each hidden node \( j \)

\[
    h_j(t) = f(p_j(t))
\]

where, \( f(\theta) = \frac{1-e^{-\theta}}{1+e^{-\theta}}, f'(\theta) = \frac{2e^{-\theta}}{(1+e^{-\theta})^2} \).

Step 8: Compute the output field of the output node

\[
    q_k(t) = \sum_{j=1}^{m} w_{kj}(t)h_j(t)
\]

where, \( w_{kj} \) is the connecting weight between the hidden layer and the output layer.

Step 9: Compute the activation of the output node

\[
    Q_k \left(S(t), a_g(t)\right) = q_k(t)
\]

Step 10: Compute error information term of the output node

\[
    \delta_k = e_k Q_k \left(S(t), a_g(t)\right)
\]

where, \( e_k = \frac{1}{2} \left( r_{t} + \gamma \max_{A(t+1)} Q_k \left(S(t+1), A(t+1)\right) - Q_k \left(S(t), a_g(t)\right) \right)^2 \)
\( r_t \) is the scalar reward received by the agent on selecting action \( a_t \) in state \( s_t \) and
\( 0 \leq \gamma \leq 1 \) is the discount factor.

Step 11: Compute the weight correction for hidden-to-output layer connections (\( j = 1, 2, \ldots, m \) and \( k = 1 \))
\[
\Delta w_{kj} = \eta \delta_k h_j(t)
\]
where, \( \eta \) is the learning rate.

Step 12: Compute the error information term of each hidden node \( j \)
\[
\delta_j = f'(p_j(t)) \sum_k \delta_k w_{kj}(t)
\]
\[
\delta_j = f'(p_j(t)) \sum_k e_k Q_k(S(t), a_g(t)) w_{kj}(t)
\]

Step 13: Compute the weight correction for input-to-hidden layer and context-to-hidden layer
\[
\Delta u_{ji} = \eta f'(p_j(t)) \sum_k e_k Q_k(S(t), a_g(t)) w_{kj}(t)x_i(t)
\]
\[
\Rightarrow \Delta u_{ji} = \eta \frac{2e^{-(p_j(t))}}{1 + e^{-(p_j(t))}} \sum_k e_k Q_k(S(t), a_g(t)) w_{kj}(t)x_i(t)
\]
\[
\Delta v_{ji} = \eta f'(p_j(t)) \sum_k e_k Q_k(S(t), a_g(t)) w_{kj}(t)y_j(t)
\]
\[
\Rightarrow \Delta v_{ji} = \eta \frac{2e^{-(p_j(t))}}{1 + e^{-(p_j(t))}} \sum_k e_k Q_k(S(t), a_g(t)) w_{kj}(t)y_j(t)
\]

Step 14: Update the hidden-to-output layer connection weights
\[
w_{kj}(t) \leftarrow w_{kj}(t) + \Delta w_{kj}(t)
\]
Step 15: Update the input-to-hidden and context-to-hidden layer connection weights

\[ u_{ji}(t) \leftarrow u_{ji}(t) + \Delta u_{ji}(t) \]

\[ v_{ji}(t) \leftarrow v_{ji}(t) + \Delta v_{ji}(t) \]
Chapter Six

Simulation Results

We modeled the reinforcement learning agent based on the algorithm described in Section 5.4 and the Multi Kanban model described in Section 5.3 in Java and performed 10 runs for a period of 10,000 minutes each. The number of demands satisfied by the Reinforcement Learning Agent and by the Multi Kanban Mechanism are shown in Table 4 and Table 5 respectively.

<table>
<thead>
<tr>
<th>Simulation No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfied demand A</td>
<td>719</td>
<td>775</td>
<td>744</td>
<td>817</td>
<td>753</td>
<td>731</td>
<td>738</td>
<td>743</td>
<td>722</td>
<td>738</td>
</tr>
<tr>
<td>satisfied demand B</td>
<td>719</td>
<td>775</td>
<td>744</td>
<td>817</td>
<td>753</td>
<td>724</td>
<td>737</td>
<td>742</td>
<td>713</td>
<td>734</td>
</tr>
<tr>
<td>satisfied demand C</td>
<td>732</td>
<td>774</td>
<td>743</td>
<td>817</td>
<td>752</td>
<td>741</td>
<td>763</td>
<td>737</td>
<td>712</td>
<td>734</td>
</tr>
<tr>
<td>satisfied demand D</td>
<td>644</td>
<td>615</td>
<td>609</td>
<td>640</td>
<td>639</td>
<td>618</td>
<td>618</td>
<td>572</td>
<td>664</td>
<td>607</td>
</tr>
<tr>
<td>satisfied demand E</td>
<td>712</td>
<td>771</td>
<td>738</td>
<td>673</td>
<td>750</td>
<td>732</td>
<td>691</td>
<td>704</td>
<td>710</td>
<td>721</td>
</tr>
</tbody>
</table>

Average WIP

<table>
<thead>
<tr>
<th>WIP at time = 10000</th>
<th>35</th>
<th>7</th>
<th>2</th>
<th>8</th>
<th>7</th>
<th>94</th>
<th>64</th>
<th>19</th>
<th>34</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIP at time = 9940</td>
<td>32</td>
<td>7</td>
<td>7</td>
<td>22</td>
<td>10</td>
<td>80</td>
<td>66</td>
<td>22</td>
<td>29</td>
<td>19</td>
</tr>
<tr>
<td>Average WIP</td>
<td>46.888</td>
<td>13.826</td>
<td>11.658</td>
<td>18.350</td>
<td>17.033</td>
<td>86.693</td>
<td>61.355</td>
<td>52.278</td>
<td>45.202</td>
<td>56.678</td>
</tr>
</tbody>
</table>
Table 8: Multi Kanban mechanism

<table>
<thead>
<tr>
<th>Simulation No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>satisfied demand A</td>
<td>817</td>
<td>757</td>
<td>832</td>
<td>809</td>
<td>767</td>
<td>828</td>
<td>794</td>
<td>816</td>
<td>806</td>
<td>775</td>
</tr>
<tr>
<td>satisfied demand B</td>
<td>484</td>
<td>488</td>
<td>496</td>
<td>496</td>
<td>494</td>
<td>494</td>
<td>489</td>
<td>494</td>
<td>494</td>
<td>477</td>
</tr>
<tr>
<td>satisfied demand C</td>
<td>560</td>
<td>556</td>
<td>579</td>
<td>580</td>
<td>572</td>
<td>574</td>
<td>552</td>
<td>530</td>
<td>563</td>
<td>560</td>
</tr>
<tr>
<td>satisfied demand D</td>
<td>159</td>
<td>161</td>
<td>164</td>
<td>167</td>
<td>165</td>
<td>164</td>
<td>159</td>
<td>163</td>
<td>163</td>
<td>162</td>
</tr>
<tr>
<td>satisfied demand E</td>
<td>319</td>
<td>322</td>
<td>327</td>
<td>333</td>
<td>328</td>
<td>329</td>
<td>317</td>
<td>326</td>
<td>328</td>
<td>324</td>
</tr>
<tr>
<td>WIP at time = 10000</td>
<td>1321</td>
<td>1091</td>
<td>1351</td>
<td>1272</td>
<td>1123</td>
<td>1335</td>
<td>1226</td>
<td>1216</td>
<td>1235</td>
<td>1209</td>
</tr>
<tr>
<td>Average WIP</td>
<td>654</td>
<td>542</td>
<td>674</td>
<td>611</td>
<td>579</td>
<td>661</td>
<td>653</td>
<td>605</td>
<td>614</td>
<td>569</td>
</tr>
</tbody>
</table>
The WIP vs. Time plots for each simulation run are as follows:

I. Reinforcement Learning agent:

Figure 15: WIP vs. Time plot for RL simulation #1

Figure 16: WIP vs. Time plot for RL simulation #2
Figure 17: WIP vs. Time plot for RL simulation #3

Figure 18: WIP vs. Time plot for RL simulation #4
Figure 19: WIP vs. Time plot for RL simulation #5

Figure 20: WIP vs. Time plot for RL simulation #6
Figure 21: WIP vs. Time plot for RL simulation #7

Figure 22: WIP vs. Time plot for RL simulation #8
Figure 23: WIP vs. Time plot for RL simulation #9

Figure 24: WIP vs. Time plot for RL simulation #10
II. Multi Kanban mechanism:

![Figure 25: WIP vs. Time plot for MK simulation #1](image1.png)

![Figure 26: WIP vs. Time plot for MK simulation #2](image2.png)
Figure 27: WIP vs. Time plot for MK simulation #3

Figure 28: WIP vs. Time plot for MK simulation #4
Figure 29: WIP vs. Time plot for MK simulation #5

Figure 30: WIP vs. Time plot for MK simulation #6
Figure 31: WIP vs. Time plot for MK simulation #7

Figure 32: WIP vs. Time plot for MK simulation #8
Figure 33: WIP vs. Time plot for MK simulation #9

Figure 34: WIP vs. Time plot for MK simulation #10
Chapter Seven

Conclusions

From the simulation results, we conclude that the WIP fluctuates in a limited range for the Reinforcement Learning agent when compared to the Multi Kanban mechanism, where WIP increased steadily against time. Table 3 and Table 4 infer that the number of demands satisfied by the Reinforcement Learning agent is greater than that by the Multi Kanban mechanism.
References


