SUPERVISORY CONTROL OF ROLLING MILLS: 
THE OPTIMISATION OF SOAKING PIT 
SCHEDULES

Thesis submitted for the degree of

Doctor of Philosophy

at the University of Leicester

by

Turhan Ozen BSc (METU) MSc (Imperial)
Department of Engineering
University of Leicester

September 2002
Abstract

The work presented in this thesis is the result of research that aims to enhance the supervisory (level II) control system used at Alcoa Europe FRP and supplied by VAI UK.

There are opportunities for improvement in three main areas:

- improved product shape, especially at low gauges;
- fault diagnosis of product quality, to improve yield;
- optimisation of soaking pits/rolling mill process, to increase throughput and improve efficient use of plant resources.

The first part of this thesis focuses on the Expert System developed to augment the existing system for improved supervisory control. The Expert System focuses on utilising human knowledge, whilst the existing system provides the infrastructure for the overall system.

The second part of the thesis explains the scheduling system developed to manage the operations of the soaking pits upstream of the rolling mill in order to gain maximum throughput from the overall process and to use the plant resources efficiently.
Acknowledgement

I would like to express my gratitude to Professor Ian Postlethwaite and Dr Da-Wei Gu for their invaluable supervision and help throughout my PhD studies. For all their help and collaboration, I would also like to thank to my research colleagues Dr Li Qun Yao from the University of Leicester and Dr William Browne who was at the University of Leicester during this project and now continues to contribute from the University of Reading. Steve Loves, Simon George, John White, Michael Mar, Chris Lane and Chris Skillen from Alcoa Europe FRP deserve a very special thank you for not only their invaluable help throughout this work but also for their friendship and the working environment they provided during the times spent at the plant. I would also like to express my gratitude to Dr Ian Robinson and Dr Philip Attack from VAI Tech UK for the support and supervision they gave throughout the project.

Finally, I wish to thank to all my colleagues in the Control and Instrumentation Group at the University of Leicester Engineering Department, and my friends and family who gave me support and inspired me throughout. Dr Emmanuel Prempain, Dr Yi Cao, Dr Fu-Wah Poon, Muhammed Khalid Khan, Dr Mathew Turner, Dr Sarah Gatley, Julie Holland and Sema Ozen, are some of the people whom I cannot express how grateful I am.

The work presented in this thesis is the result of a research project funded by the UK Engineering and Physical Sciences Research Council.
# Table of Contents

Abstract ............................................................................................................................................. I
Acknowledgement ............................................................................................................................ II
Table of Contents ............................................................................................................................. III
1 Introduction..................................................................................................................................... 1
2 The Soaking-Pits/Rolling-Mill Process ......................................................................................... 10
  2.1 Background ............................................................................................................................ 10
  2.2 Layout of the soaking pits/rolling mill process .............................................................. 10
  2.3 Soaking Pits .......................................................................................................................... 12
  2.4 Rolling mill ............................................................................................................................. 15
    2.4.1 Work and backup rolls ............................................................................................... 16
    2.4.2 Mill housing ................................................................................................................... 16
    2.4.3 Cooling Sprays .............................................................................................................. 17
    2.4.4 Instrumentation .......................................................................................................... 17
    2.4.5 Control Hierarchy ......................................................................................................... 17
  2.5 Slabs................................................................................................................................. 18
  2.6 Scalper............................................................................................................................ 18
  2.7 Pit Areas ............................................................................................................................ 19
  2.8 Transport .......................................................................................................................... 19
3 Expert Systems ......................................................................................................................... 20
  3.1 Introduction ....................................................................................................................... 20
    3.1.1 Knowledge representation ....................................................................................... 23
    3.1.2 The main structure of an expert system ................................................................... 23
    3.1.3 Selection of the expert system shell ........................................................................... 27
    3.1.4 Basic components of G2 ......................................................................................... 29
    3.1.5 Object oriented philosophy in G2 .......................................................................... 31
  3.2 System Design .................................................................................................................. 32
    3.2.1 Architecture ............................................................................................................... 33
    3.2.2 Expert system design ............................................................................................... 34
  3.3 Project development using G2 ......................................................................................... 38
    3.3.1 Designing the knowledge base ............................................................................... 38
    3.3.2 Module hierarchy ...................................................................................................... 38
    3.4 Fusion of data mining and data gathering ................................................................. 54
    3.4.1 Knowledge-elicitaton ............................................................................................. 52
    3.4.2 Data-mining .............................................................................................................. 53
    3.4.3 Schedules of set points ........................................................................................... 54
  3.5 First online trial ............................................................................................................... 57
    3.5.1 Aims and objectives ............................................................................................... 58
    3.5.2 Method ...................................................................................................................... 59
    3.5.3 Results ...................................................................................................................... 60
    3.5.4 Discussion ................................................................................................................ 64
    3.5.5 Results ...................................................................................................................... 65
  3.6 Summary ........................................................................................................................... 66
4 Artificial Neural Networks ..................................................................................................... 67
  4.1 Definition .......................................................................................................................... 67
1 Introduction

The metal industry is considered one of the primary heavy industries upon which the industrial development of a nation is based. Increasing global competition in the world metal market and the requirement from customers of higher tolerance product are pressing the producers and plant equipment manufacturers to search for more advanced and efficient control systems to improve the production quality and to enhance the productivity for higher throughput.

The work presented in this thesis is the result of a collaborative study in this direction between University of Leicester, Alcoa Europe FRP and VAI UK. The aim of the research is to enhance the supervisory (level II) control system used at Alcoa Europe FRP and supplied by VAI UK [1].

The existing supervisory control system at Alcoa Europe FRP is advanced and uses model based adaptation in order to calculate the required set points for rolling [2]. However, there are opportunities for improvement in three main areas:

- improved product shape, especially at low gauges;
- fault diagnosis of product quality, to improve yield;
- optimisation of soaking pits/rolling mill process, to increase throughput and improve efficient use of plant resources.

The shape of plate product from the rolling mill is manually controlled via the existing level II system. The human operators are able to visually quantify shape e.g. wavy edge, and can suggest possible causes e.g. lack of sprays on centre of work rolls. Such human expertise can be compiled into a knowledgebase so that an Expert System can control the shape problems wherever they originate and adopt suitable mechanisms for improvement via a control strategy selection.

Fault diagnosis/prediction is not part of the current control system for either the plant or product. Conventional plant fault prediction utilising statistical process control is via a PLC (industrial microcomputer). Any eventual product faults are not predicted in the existing system. Unfortunately, product quality faults do occur, which require time intensive human investigation in an attempt to determine the cause. An expert system can identify important faults, predict their occurrence and automatically diagnose the cause with remedial action suggested. This can save time, improve yield and reduce cost.

The soaking pits are also not controlled by the existing mill set-up system; instead a PLC-based system is used. The operation of soaking pits is heavily constrained with many temperature settings and heating practices by strict metallurgical standards.
Therefore, an opportunity for improving heating methods of the product lies in the timings for loading the pits to ensure that the rolling mill is not idle waiting for slabs to reach rolling temperature or rejecting slabs because it is busy with processing the content of another pit or the slabs have been heated for too long.

In the rest of this thesis the system developed to achieve these goals, the methods that are used and the results obtained are described in detail. Some of the results have been previously published:

[3] explains the overall design of the complete system and the interactions between its different modules and the exiting plant systems.

[4] explains the neural network models developed to estimate the heating and rolling times of pit loads in order to use this data to synthesis a schedule for the soaking pits/rolling mill process which results in higher throughput and efficient plant processing.

[5] explains the system developed to synthesise a schedule for the soaking pits/rolling mill process which increases the throughput and improves the efficient use of the resources which also results in higher product quality.

[6] explains the rule formation techniques used to build up the knowledge base for the expert system that enhances the level II control of the rolling process.

[7] shows how data-mining and knowledge-elicitation techniques were combined in the advanced supervisory control system and presents the results of successfully rolled aluminium plates of exacting specifications.

[8] explains the development of the initial expert system interface.

[9] gives detailed information on soaking pits, explains the present system used to manage the soaking pits/rolling mill process, presents the state of research for enhancing the productivity of similar processes and proposes methods to develop a scheduling system to increase the throughput of the process, efficient use of the resources and the quality of the product.

I am a co-author of the refereed publications [3, 5, 6, 7].

The control systems used for production of aluminium plates can be divided into three levels, i.e., level I dynamic control, level II supervisory control and level III plant wide control. Figure 1 shows the relations between these three levels of control.

The level I control system directly communicates with plant equipment to control individual units of the plant, working at certain set-points specified by the level II system in the presence of various plant disturbances. A mill gauge control system and a mill speed control system are two examples of such level I control systems. A level II supervisory control system is responsible for the production of correct set-points for
several level I control systems, so that these sub-systems can co-operate together to produce quality final products. Level III is the top-level control system of the whole plant for plant-wide optimisation and scheduling.

![Levels of rolling mill automation.](image)

After thirty years worth of development, level I control techniques in metal rolling processes have become mature. Hence, a common feature of modern rolling mills is that they are equipped with level I closed-loop process control. Therefore, in order to increase yields further and to consistently maintain quality products, an attention has been focused in recent years on the development of level II systems. Now, level I and II systems work hand-in-hand [10]. Several advanced level II rolling mill set-up systems have been developed [10, 11, 12, 13, 14, 15, 2]. In addition to using classical mill set-up models, some advanced model adaptation techniques have been adopted in level-II control [15, 2]. Artificial Intelligence techniques have also been used in mill set-up systems [14, 2]. But the human expertise resource available on plant has not been fully utilised, so a technology capable of capturing human knowledge in a control system is required.

The first part of this thesis focuses on the Expert System developed to augment the existing system for improved supervisory control [1, 16, 17]. The Expert System focuses on utilising human knowledge, whilst the existing system provides the infrastructure for the overall system. This work was done by a team of researchers. As a member of the team I was involved in every stage of this research and contributed especially in deciding the system architecture, data gathering, testing the system and researching recent advances in this field. The implementation of the system was mainly carried out by the Research Associates working on this EPSRC funded project.

The second part of the thesis explains the scheduling system developed to manage the operations of the soaking pits upstream of the rolling mill in order to gain maximum throughput from the overall process and to use the plant resources efficiently. This
work was carried out by myself under the direction of Prof Postlethwaite and Dr Gu and introduced the novel integration of the neural network technology and scheduling theory. This part of the thesis represents my main individual contribution to the complete supervisory control system.

A soaking pit (a furnace to heat the metal prior to hot rolling) is a key unit process in the metal industry. Energy consumption by the soaking pits absorbs a significant part of the total operational costs, and the effectiveness of soaking pit operation has a pronounced effect on the downstream mill operation as well as on the quality of the final product. Although nowadays, the metal industry has started to employ continuous casting, which does not utilise soaking pits, there are still a considerable number of hot rolling mills, where continuous casting is not suitable and soaking pits are used to produce hot slabs for subsequent rolling. In the aluminium industry, processes upstream of rolling, such as scalping, which take place at room temperature, makes the soaking pits a necessary part of the process.

Due to the importance of this process a lot of research was carried out during the 70s and 80s on the modelling, optimisation and control of the soaking-pits/rolling-mill process [18, 19, 20, 21, 22]. Some of the research concentrated on the continuous behaviour of the thermal dynamics of the soaking pits and related processes, while others focused on the discrete events of the various unit operations for the sake of scheduling and operation control. But since then, very little research has been done on soaking pits. Among the recent publications on this topic are [23, 24, 25, 26, 27, 28] which present valuable work done on the modelling, control and optimisation aspects of the soaking process. These studies present various possibilities to improve the soaking pit process and its control using computer technology. The exponential development in computer and software technology has brought a wide opportunity to improve and make industrial processes more efficient. The competitors in this industry who successfully exploit these developments will gain a significant advantage.

In [23], a soaking pit/rolling mill process model is developed for a real industrial steel mill based on extended coloured Petri nets (ECPN). Both the discrete events and the continuous dynamics involved in the process are modelled in a unified framework, allowing better investigation of the interactions between these two parts. ECPN diagrams are given for the critical components involved in the soaking pit/rolling mill process, including the arrival of hot ingots, the charging and discharging of soaking pits, the soaking process, the preheat furnace operation, the mill operation, etc. The idea used to model the heating process of the ingots is the same as the one in [29].

In [24], which complements [23], a heat scheduling and optimisation problem is addressed which consists of decisions on the optimal number of soaking pits to be
used, the optimal soaking time, and the optimal temperature set-points of individual soaking pits for a practical soaking pit/rolling mill complex. Constructed upon a hybrid system model based on extended coloured Petri nets for the soaking pit process presented in [23], the optimal temperature set points for a soaking pit for different soaking times are obtained by a simulation-based offline heuristic optimisation strategy. Algorithms to determine the optimal number of soaking pits to be used and the optimal soaking time have been developed for the desired optimal rolling rate, which can be obtained via simulation. The overall heat scheduling and optimisation problem of soaking pits is investigated in three sub-problems:

- Determine the optimal number of soaking pits,
- Determine the optimal soaking time,
- Determine the optimal temperature set points for the soaking pits.

In [25], a numerical study is conducted for heat transfer in a soaking pit. A transient three-dimensional radiation-convection-conduction model is developed and this model is used to assess the effects of various operating parameters on the heat transfer to ingots.

[26] describes the development and implementation of a hybrid SCADA (Supervisory Control and Data Acquisition) system for the air suspension alumina calciner. The system includes two additional modules, an 'expert system' and a dynamic model continuously predicting the quality of the product. The system guides the furnace operator to reduce sensible heat losses by operating with a marginal excess air blow rate. This results in higher energy efficiency at higher production rates and, at the same time, avoids potentially dangerous operating conditions leading to loss of production and consequential energy losses. The traditional SCADA functions (data acquisition, maintaining history files for important parameters and display of important operating parameters in both a mimic process diagram and a series of graphs) are performed too. In addition to these tasks a simplified dynamic model gives the operator a prediction of the current value of the main product quality parameter, which can only be measured with laboratory equipment.

The efficient and reliable control of a reheating furnace is a challenging problem. Conventional reheating furnace operation has been heavily dependent upon lookup tables that list the optimal set points. [27] describes a modified modular neural network for the supervisory control of a re-heating furnace. Based on the divide-and-conquer concept, a modular network is capable of dividing a complex task into subtasks, and modelling each subtask with an expert network. To model such activities, a gating network is used for the classification and allocation of the input data to the corresponding expert network. To overcome the correlation effects among
process variables and the problem of dimensionality, principal component analysis (PCA) has been employed to remove the correlation and reduce the problem dimension. From PCA analysis, it was possible to decide on the optimal dimension for the problem, to describe the dynamic behaviour of the furnace.

[28] describes a rule-driven mathematical model and presents an efficient rolling horizon scheduling algorithm based on tabu search as well as computational results.

The work done on optimisation of the soaking pits/rolling mill process during the research presented in this thesis focuses on scheduling.

The utilization of classical scheduling theory in most production environments is minimal. In many production environments, including the soaking pits/rolling mill process, first line management carries out scheduling and plant loading and the existing body of theory that may relate to some or perhaps all of the scheduling problems is not used. This work is aimed at researching the gap between the scheduling theory and practice and developing a real industrial application.

The scheduling of soaking pits/rolling mill operation is a reasonably sized industrial sequential scheduling and resource allocation problem. An optimal schedule would fix the execution time of controllable events and select resources, whenever there is a choice, in such a way that the sequence of events minimizes the completion time of the last batch in the sequence, while satisfying all model constraints. A desired schedule for the soaking pits/rolling mill process would increase the machine utilization and throughput. This can be achieved by setting the timings for loading the pits to ensure that the rolling mill is not waiting idle for slabs to reach rolling temperature or rejecting slabs even though they have become ready for rolling because the mill is busy rolling another pit’s load.

This thesis presents an approach to modelling the soaking pits/rolling mill process and describes powerful heuristics (SPO Algorithm) that produce near optimal schedules.

The reminder of the thesis is organized as follows.

In chapter 2, an engineering description of the Alcoa Europe FRP plant at Kitts Green is given. The existing rolling mill control system and soaking pits/rolling mill process are described to place the developed system in context. A physical description of the plant shows the scope of the project, whilst the outline of the existing control system shows where the supervisory control and scheduling system are introduced.

Chapter 3 presents the work done on supervisory control of the rolling process by a team of researchers from the University of Leicester. The team consisted of Dr William Browne, Dr Li Qun Yao and myself led by Prof. Ian Postlethwaite. Dr Da-Wei Gu, Dr Yi
Cao, Dr Fu-Wah Poon and Mr Lim Fung Siong also contributed to this work at various stages. The chapter starts with a brief summary of the background on expert systems in order to familiarise the reader with the technology and underlying principles. The method by which the Expert System is integrated into the existing control structure is then described. The chapter continues with detailed information on methods used to develop the expert system's interface and to construct the knowledge base. The chapter concludes with results from a successful on-line trial of the developed system which demonstrate enhanced product quality.

The second part of the thesis starts with chapter 4 where a concise summary of Artificial Neural networks is presented. Artificial Neural networks are used to estimate the heating time and rolling time of the pit loads. This information is then used to synthesise the schedule for the soaking pits/rolling mill process. It is important to understand the technology of Artificial Neural Networks in order to use the most appropriate techniques and develop an architecture that performs most effectively in estimating these data. The ANN models developed to estimate the heating and rolling times are explained in chapter 5. These models are referred to as "SPO ANN models" throughout the thesis. Chapter 5 starts with an explanation of the architecture of the SPO ANN models and the training algorithms used to capture the inherent information contained in the input data fed into the models. It continues with detailed information on pre and post processing applied to this data before it is used in training and regularization methods used to control the network parameters in order to maximise the generalization capability of the models. The chapter concludes with simulation results and their analysis.

Scheduling of soaking pits/rolling mill process is a difficult combinatorial optimisation problem. In order formulate this problem and to understand the challenges that are addressed, it is necessary to have an appreciation of state of the art optimisation theory and similar combinatorial optimisation problems. To evaluate the applicability of proposed solution methods and to compare performance of different algorithms, one needs to understand the computational complexity theory. For these reasons, chapter 6 is devoted to providing necessary background on optimisation in general, combinatorial optimisation and computational complexity.

Chapter 7 explains the machine scheduling problems in general. It is aimed at providing adequate background on the subject so that one can formulate a machine-scheduling problem and can identify where it stands in this diverse field. With the information provided in this chapter the reader is supplied with enough knowledge to understand the literature on this subject. Of course the chapter is written with a focus
on the soaking pits/rolling mill optimisation problem and it is referred to when the emphasis is seen as useful.

Chapter 8 is devoted to dynamic programming. Dynamic programming is the method used to decide the allocation of pit loads to the pits. The chapter starts with a brief background on Dynamic Programming and follows with a summary of the underlying principles. It concludes by explaining the computational difficulties in applying this algorithm and methods to overcome these difficulties.

Chapter 9 explains the Soaking Pit Optimisation (SPO) system in detail. The SPO system is developed to synthesise a schedule for the soaking pits/rolling mill process. It consists of a neural network module for estimating the duration of heating and rolling operations and a scheduling module consisting of heuristic methods working with a dynamic programming algorithm to synthesise a schedule for the process. The chapter explains the architecture of the complete system, the interactions between its components and its place within the expert system and the existing plant systems. It begins with an overview of research on similar problems and continues with the development of the system. The chapter concludes by presenting the results obtained using the SPO system and makes comparisons with the schedules produced by the present system. Significant improvements are seen.

The thesis concludes with chapter 10 which includes a summary of the research carried out during this project and a discussion of the directions for future research.

Figure 2 and Figure 3 present pictures of a soaking pit and a rolling mill respectively. These are the machines that the work presented in this thesis is focused on.
The Soaking-Pits/Rolling-Mill Process

This chapter describes the soaking pit/rolling mill operation at Alcoa Europe FRP, Kitts Green. The existing process is being used to manage the soaking pit/rolling mill system in the new project, while the rolling of the soaking pit/rolling mill system is being developed and scaled up in the new project. The soaking pit, the rolling mill and the auxiliary facilities are described in detail.

2.1 Background

Casting and rolling are the key steps in producing Aluminium. In the casting process, molten Aluminium is poured into a mould and is cooled to form a solid billet. This billet is then cut into slabs and reheated to a temperature of approximately 500°C. This allows for the homogenization of the slab's microstructure before further processing of Aluminium is carried out.

2.2 Process

The soaking pits/rolling mill complex at Alcoa Europe FRP, which is presented in Figure 5, consists of 3 soaking pits arranged in the soaking pit area and one reversing hot rolling mill to produce a variety of slabs and sheets of Aluminium products. One overhead travelling crane that spans the pit and mill area is responsible for charging the incoming slabs to the soaking pits and drawing the slabs from the soaking pits to the mill for the subsequent mill processing. Transfer of slabs on the mill area is carried out through a mill table, which connects the mill and the other auxiliary facilities in the area.

Figure 2. A bottom-fired gas soaking pit, Alcoa Europe FRP, Kitts Green.

Figure 3. A 4-high hot reversing mill, Alcoa Europe FRP, Kitts Green.
2 The Soaking-Pits/Rolling-Mill Process

This chapter describes the soaking pits/rolling mill process at Alcoa Europe FRP, Kitts Green. The existing rolling mill control system and present methods used to manage the soaking pits/rolling mill process are described to place the developed system in context. A physical description of the plant shows the scope of the project, whilst the outline of the existing control system shows where the supervisory control and scheduling system are introduced. For more information on soaking pits, the rolling mill and its auxiliary facilities, [9] and [8] are good references.

2.1 Background

Casting and rolling are the main metal forming techniques in transforming Aluminium into plate Aluminium. The process begins with the blast furnace where molten aluminium is made from raw materials. Next, the molten aluminium is poured into a mould and is cooled to form an ingot in a caster. This process is known as direct chill (DC) casting. The ingot is then cut into slabs. Scalping the surface is done to present a smooth surface for rolling. The slabs are then placed in a preheated furnace at a temperature of approximately 500°C. This allows complete homogenisation of the slab's microstructure to take place. A simple diagram showing the hot processing of Aluminium is shown in Figure 4.

![Diagram of hot processing of Aluminium](image)

**Figure 4. Hot processing of Aluminium**

2.2 Layout of the soaking pits/rolling mill process

The soaking pits/rolling mill complex at Alcoa Europe FRP, which is presented in Figure 5, consists of 9 soaking pits arranged in the soaking pit bay and one reverse hot rolling mill to produce a variety of shapes and sizes of aluminium products. One overhead travelling crane that spans the pits and mill area is responsible for charging the incoming slabs to the soaking pits and drawing the slabs from the soaking pits to the mill for the subsequent roll processing. Transfer of slabs on the mill area is carried out through a mill table, which connects the mill and the other auxiliary facilities in the
mill area. All other transportations are carried out by forklifts. Slabs are produced by the upstream aluminium making process and have to be scalped before being rolled in order to get a shiny and smooth surface. But there are also slabs that are heated in the soaking pits to be homogenised or stress relieved. After being homogenised or stress relieved, slabs are transported to a waiting area to wait until they are scheduled for hot rolling. After rolling, the plates finished rolling are placed on pallets. Figure 6 and Figure 7 summarise the tasks involved with the soaking and rolling processes at Alcoa Europe FRP. Diagram 1 shows in detail the present method that is used to manage the soaking pits and the rolling mill.

Figure 5. The layout of the soaking pit/rolling mill complex at Alcoa Europe FRP.

Figure 6. The layout of soaking pits/rolling mill operation at Alcoa Europe FRP.

Figure 7. The Soaking pits/rolling mill process.
Diagram 1. The present soaking pits/rolling mill process

2.3 Soaking Pits

Although the direct rolling of slabs into finished products is sometimes carried out in facilities such as hot-strip mills, semi finished products must usually be reheated before final rolling. This reheating operation may be necessitated by the logistics or
scheduling of the rolling operations and/or the need to condition the surfaces of the semi-finished workpieces. In the case of an aluminium process, before rolling takes place the surface of the aluminium slabs should be scalped to obtain the desired shape and a shiny surface. Scalping cannot be done when aluminium is hot. So after casting, the slabs are left to cool down and by the time they are charged into the pits for reheating they have reached room temperature.

By the reheating operation it is intended to raise, as uniformly as possible, the temperature of the slabs to levels appropriate for hot rolling. By soaking is meant the final equalisation of temperature within the interior of the workpiece. In this thesis the reheating and soaking processes are assumed as one single process. The slabs are heated to temperatures in the range of 400 to 500°C. Then they are allowed to soak for a few hours at the correct temperature. The preheat duration depends on the dimensions of the slabs in the pit and the alloy characteristics whereas after all the slabs reach the soaking temperature, the soaking duration depends only on the alloy type.

As workpieces increase in cross sectional dimensions, the problem of obtaining satisfactory temperature uniformity in a reasonably short time becomes increasingly difficult when the heat must be applied to their surfaces. Attempts to heat workpieces too rapidly may result in burned metal or the development of stresses within the metal that adversely affect its quality. And when workpieces of various cross-sections are fed into the reheating furnace, it is difficult to avoid overheating the thinner pieces or under-heating the thicker ones.

Soaking pits used in aluminium plants have some specification differences in comparison with the soaking pits used by the steel industry. This comes from the different nature of the two processes. Required rolling temperatures are very different for the two materials because of their different chemical properties. Preferred rolling temperature for aluminium slabs is between 400-500°C. At these temperatures the slabs are relatively soft and ductile and so heavy reductions in thickness can be made as they are rolled. For high-carbon steels reheated in oxidising atmosphere, the preferred temperature is usually in the range 1070 to 1140°C; and for medium-carbon steels somewhere in the range 1100 to 1150°C, while low-carbon steels that contain no alloying elements may be heated to 1300°C.

The quantity of heat that must be supplied to slabs to bring them up to hot-rolling temperatures depends upon the time lapse between the completion of the metal making operation and the delivery of the slabs to the soaking pits. Ideally, the stripping location should be such that a minimum time occurs between stripping and inserting the ingots into the soaking pit [30]. Although it has generally been believed that ingots should be completely solidified during soaking, studies have shown the
feasibility of rolling ingots that are not completely solidified [31]. Associated experiments have demonstrated that such a practice results in improved homogeneity of the workpieces and large reductions in the fuel consumed by the pits. Some examples which exploit this fact are given in [32, 22]. Because of the preliminary processes such as scalping that the aluminium slabs should go through, at this stage, it is not feasible to exploit this fact in the hot rolling of aluminium.

The operation of soaking pits is carefully controlled by computers [33, 34, 35]. Electric, hydraulic or pneumatic controls are frequently used to automatically regulate the pit temperature, fuel-air ratio, fuel-steam ratio, furnace pressure, fuel shut-off and turn-on, automatic reversals in two-way fired pits and waste-gas temperature [36]. To obtain slab temperatures appropriate for rolling with minimum utilisation of fuels and facilities, the following principle instruments are used:

- Thermocouples for the measurement of pit air temperature and the temperature of some slabs. Usually the temperatures of the thickest and the thinnest slabs are monitored.
- Flow instrumentation for measuring the flow rates of the fuels, combustion air and steam.
- Pressure sensors for measuring pit and/or flue pressures.
- Waste gas analysers [37].

The soaking pits that are used at Alcoa Europe FRP are bottom fired gas furnaces. Depending on the size of the pit there are 4 or 8 burners located at the bottom of the pits. The burners are flame pattern controlled in which the size of the flame is controlled rather than using an on-off mechanism. To obtain a uniform temperature everywhere inside the pit, the fans located at the bottom of the pits circulate the air in the pits. The slabs are charged in and drawn out of the pits from the top. For charging and drawing, the covers are lifted and moved sideways by automatic cover cranes. When the soaking is completed the slabs are taken out one by one to be rolled and when a slab is being rolled the other slabs remain in the soaking pit. So a soaking pit cannot accept the next load until all its contents have been rolled. [9] gives detailed information on various types of soaking pits and related equipment.

Soaking pits at Alcoa Europe FRP are also used for other purposes, which are not followed by rolling, such as homogenising and stress relieving of some slabs. These slabs are then stored until they are scheduled for hot rolling.

Soaking pits can take limited volume and weight of slabs. Total heating time (reheating time + soaking time) of a pit load depends on the characteristics of the pit contents, the characteristics of the pit and the recipe applied.
### 2.4 Rolling mill

The rolling mill at Alcoa Europe FRP, Kitts Green is a 4-high hot reversing mill, which rolls slabs down to plates. Slabs are first rolled edgeways to achieve the desired width and then lengthways to achieve the final thickness. By each pass backwards and forwards through the reversing mill a reduction in thickness of slabs is obtained. Upstream of the mill there is an edge rolling facility which is used to suppress edge cracking during the hot working process. Downstream there is a full width shear which can cut the hot plate into shorter lengths which are more manageable in the later processes. A diagram of the rolling process is shown in Figure 8.

![Diagram of the rolling mill](image)

**Figure 8. Single stand reversing rolling mill**

Once the contents of a pit has completed its soaking, an overhead crane transfers one of the slabs via the tilt-pot to the entry table, where initial temperature and gauge measurements are taken. The recording of these values is used as a trigger for the level II supervisory control system to calculate the rolling schedule. The schedule consists of the mill set-points (such as pass gauge) and the calculated necessary number of passes to roll the slab down to the finished plate gauge.

A schematic of the rolling mill and auxiliary facilities at Kitts Green is shown in figure 8, ranging from the soaking pits to the run-out-table of the 4 high, reversing mill stand.
The rolling time of a slab depends on the slab characteristics and the operations carried out on it. The slabs processed at the rolling mill are loaded on pallets to be transported to downstream operations. The order in which slabs in a batch are rolled is the reverse of the order in which they are stacked on a pallet.

A more detailed description of the rolling mill at Kitts Green and comparisons with previous work done by Robinson ([38]) for supervisory control are given in [8]. In the following subsections a brief description of the components of the rolling mill is given.

### 2.4.1 Work and backup rolls

The work rolls are the rotating rolls that are in contact with the slab. The ends of the work rolls are passed through bearings that are lubricated by oil to allow them to rotate. The bearings are housed in a chock. Due to the effect of roll wear, the roll causes markings on the aluminium plate, and hence they are changed every three to four days. The roll wear is caused by contact of the aluminium plate with the roll, hence the width of the wear is dependent upon the width of the aluminium plate rolled. The changing wear pattern changes the required control, but roll wear is hard to predict or measure.

The backup rolls support the work rolls. The backup rolls prevent the work rolls from bending and allow the work rolls to have a smaller diameter and therefore smaller contact area with the strip. This enables a smaller rolling load to be used. Work roll wear cannot be easily measured, but backup roll position is measured. This indirect control method can introduce inaccuracies.

### 2.4.2 Mill housing

The mill housing holds the chocks and they are designed to move the chock vertically. In addition, the mill housing accommodates the mill screws which are connected to the back up roll. The mill screws will exert a force on the aluminium plate. They are threaded through the top of the mill housing and can be used to control the rolling force. Due to these high forces, the mill housing stretches, resulting in difficulties in
controlling the gauge. A process model accounting for these uncertainties is required to develop the expert system for successful supervisory control.

2.4.3 Cooling Sprays

The heat conduction from the aluminium plate causes the work rolls to heat up during rolling. The distribution of work roll expansion along its length is known as thermal camber. Excessive thermal camber distorts the work rolls, reducing product quality by introducing internal stresses and visible shape problems. In order to cool the mill rolls, it is necessary to use the cooling sprays. A spray bar that contains a number of nozzles is positioned along the length of the work rolls. The nozzles’ spray rates can be controlled individually to spray coolant onto the work rolls. Strategies must be developed to control the thermal camber, and indirectly to maximise the quality of the product.

2.4.4 Instrumentation

With regards to instrumentation, the rolling mills are generally poorly-equipped due to the hazardous operating conditions. The following are the main aspects of the mill that are being monitored [39]:

- Main mill drive motor current and voltage.
- Mill speed. The speed at which the work roll rotates and is calculated in m/s.
- Rolling force. Load applied through the backup roll chocks and given in tonnes.
- Plate gauge. The thickness of the plate measured in mm.
- Plate profile. The thickness variation of the plate along its width given as percentage.
- Plate temperature. Temperature of the plate after it exits the work rolls is measured in degrees centigrade using a contact thermocouple.

This limits the number of inputs the expert system can take.

2.4.5 Control Hierarchy

Control of the rolling process is normally divided into three levels. Figure 10 shows a block diagram of these levels.

Level I is the dynamic control of the plant actuators for one pass of the plate Aluminium. This level can be open-loop or closed-loop control depending on whether measurements can be taken. These loops control the quality of the plate by monitoring the variations in the mill’s exit gauge.

Level II is termed supervisory control and controls the mill as a whole. It is responsible for every pass of each plate. This level provides suitable actuator set
points to be passed to Level I, and target values for the closed loops to aim for. This level also includes diagnostics to ensure that the process measurements taken are valid, and that the mill is producing good quality product.

Level III is plant wide control, which looks at the overall plant and evaluates the processing route for a given set of plates. It will decide on the sequence of plates that are to be rolled, and how they will move around the production plant.

The focus of the overall project and the application of the expert system are based on Level II of the control hierarchy.

2.5 Slabs

The aluminium slabs processed at Kitts Green are of various dimensions up to 525 mm thick, 1500 mm wide and 4.5 m long. They are heated in the soaking pits for stress relief and homogenisation or for soaking the slabs ready for rolling. Homogenized or stress relieved slabs are stored to be soaked later. The number of slabs to be scheduled for processing decreases as they are being processed through the soaking pits/rolling mill process and increases as new ones arrive from upstream processes. According to the requirements of the processes downstream of rolling, the slabs are grouped together into batches. A batch should not be divided into separate pits and they should be rolled without mixing with other batches. After rolling, a batch is stacked on a pallet. A batch should not be divided between different pallets either.

2.6 Scalper

The outer surface of the cast slabs can be rough and uneven. To obtain a smooth blemish-free surface ready for rolling, the slabs are subjected to a number of passes through a large milling machine to simultaneously scalp up to 20mm of thickness from each of the two major faces. A slab that will be rolled has to be scalped first whereas a slab that will be stress relieved or homogenised is not scalped.
2.7 Pit Areas

Between the scalper and the soaking pits there are 6 pit areas as a buffer. In each pit area a pit load of slabs is accumulated.

2.8 Transport

Different arrows in Figure 6 show different means of transport.

There are a number of forklifts, which carry slabs from the casting area to the scalper (the slabs that will be rolled) or the soaking pit area (the slabs that will go through homogenising or stress relieving) and from the scalper to the soaking pit area.

A single crane that can carry one slab at a time does charging and drawing of slabs.

There is a mill table that connects the rolling mill and other auxiliary facilities. The slab is moved during the rolling process forward and backward through the roll gap by the mill table, and moving of the plates between shear table and rolling mill is done on the same system.

When the rolling process of a slab is finished, a vacuum crane carries the produced plates from the mill table onto the pallets. Then the pallets are carried to other locations on the plant by other cranes.
3  

Expert Systems

This chapter explains the development of an expert system to enhance the level II control of a rolling mill. An expert system is a computer program that behaves like a human expert for some problem domain. It is different from the conventional programs as it processes knowledge instead of data. Not only is it capable of explaining its decisions and the underlying reasoning, it is also able to deal with uncertain and incomplete information [70]. The aim of this research is to construct a knowledge base, containing control strategies required to perform mill set up and supervisory control. The focus for the control of the expert system is an industrial single stand reversing rolling mill.

The chapter begins with an overview then introduces some useful concepts about expert systems. The reasons for the selection of the G2 expert system shell, the architecture for the integration of the expert system and a brief introduction of G2 are also given. Discussions are given on various components of the expert system which concentrate on the aspects of practical implementation and coordination of all the software packages. In addition, the report highlights the development of mill set up using G2 and the methods used to build the knowledge base.

The development of this expert system is an ongoing project at the University of Leicester. The work reported in this chapter was carried out by a team of researchers (including myself), supervised by Professor Ian Postlethwaite. Publications arising from this work are [3, 6, 7, 8]. I am a co-author on [3,6,7]. [8] is an undergraduate project report.

3.1  

Introduction

In 1956, at a workshop sponsored by IBM at Dortmund College, a group of computer scientists working on algorithms and software which attempted to emulate some of the activities of the human mind gathered to discuss ways of directing their work to develop both hardware and software which could mimic human reasoning. The workshop marked the emergence of Artificial Intelligence (AI). The initial focuses of AI were on theorem proving and general problem solving. AI researchers developed computer programs based on complex search algorithms which have good general problem solving capabilities, independent of any specific problem domain. At that stage, it was believed that intelligent behaviour was primarily reliant on smart reasoning techniques. By mid 60s, researchers began to realise that search techniques alone would not be enough to produce an intelligent program, and the research attention was shifted to search for methods to represent knowledge and to code it into the computer in symbolic forms. Cognitive psychologists performed research on
theories to explain how human beings solve their problems with the aim of uncovering the type of knowledge which is commonly used and how this knowledge can be organized to solve problems efficiently. Artificial intelligence researchers used the results from these studies to develop techniques for representing the knowledge in the context of symbolic programming. In the late 60s, a program named DENDRAL [40, 41] was developed at Stanford University to perform chemical analysis of the Martian soil. DENDRAL captured the heuristics from many knowledgeable chemists for reorganising molecular structures of unknown compounds. This was the first program whose success was attributed to heuristic knowledge representation. The emphasis on knowledge led to the concept of Knowledge Based Systems (KBS) or Expert Systems (ES).

Although having lost popularity in the 70s due to the lack of confidence in usefulness revealed in The Lighthill Report [42], expert systems regained considerable attention in the 80s with the Japanese Fifth Generation Computing project. Expert systems have been a field of intense research ever since. Much of the work is done for medical and healthcare applications which is mainly due to the interest of the Department of Health of the UK. Expert systems have also found a place in financial institutions [43], to improve their operation. Process control is another field where expert systems are widely used. For surveys on the impact of expert systems on chemical process control, see [44]. [45] surveys the progress of expert systems in bioprocess management. Another important application is Computer Aided Design. Expert systems seem to be applicable to all engineering design, from molecular structure design [46, 47] to parametric design of mechanical systems [48] to application of systems engineering practices [49, 50]. Important categories among recent expert systems are forecasting [51, 52, 53] and management applications. Management applications may be further divided into business management [54, 55, 56] and process management applications [57, 58, 59, 60]. Expert systems are also applied in various assistance systems in the application of procedures [61], categorization [62, 63] and military applications [64].

Research on the application of expert system techniques to computer aided control systems design (CACSD) was initiated in the early 80s when Taylor et al. presented an overview of the application of expert systems to control engineering in [65]. They outlined the wide range of activities which should be addressed while designing a control system. Meanwhile, Birdwell [66] addressed some important issues in the development of a CACSD software package. Trankle et al [67] attempted to build an expert system for control system design by using ideas from planning systems to organise the process of controller design into two levels. The higher level concentrated on developing an overall strategy by building a list of goals to be accomplished. The lower level was made up of specific computer-aided control system design functions
for achieving these goals. Most of the recent research on using expert systems in CACSD emphasizes the implementation issues of expert systems and the data organization within the knowledge representation framework. An Intelligent Front End (IFE) has also been recognised as important in transforming expert systems to be more user-friendly. The object oriented and structured approach to organising data and knowledge used for control system design is beginning to gain popularity as the scope of control problems becomes larger. [68] examines the application of object oriented database management systems to computer aided control engineering.

The work presented in this chapter is on the development of an expert system for controlling the set up of an Aluminium hot reversing mill. Competition has forced metal companies to invest heavily on new facilities and technologies to modernize and streamline every step of the manufacturing process. The marketplace is even more competitive today. As a result, companies must be even more efficient in optimising the productivity of operations, while becoming increasingly responsive to customer needs by achieving maximum flexibility in production capabilities.

The UK is the world leader for low-level closed loop controllers in the metals production industry. However, over the years, she has failed to embrace new technologies for supervisory and plant-wide control systems. A major project was initiated to redress this situation by developing a generic expert system for the supervisory control of rolling mills [69].

Metal processing is a complicated control problem with a requirement for improvement of product quality and cost reduction. Human expertise and sophisticated automated systems are needed to operate this process. The strategies, which an experienced operator would use to ready the rolling mill, can now be stored in the expert system. This system has the potential to optimise the output of all the mill operators. An expert system containing a set of rules for the mill set up was previously developed [38]. These results illustrated that expert systems had the potential to enhance the control of rolling mills. This encouraged an industrial application to be developed. The work described in this chapter is a continuation of [38] and focuses on developing an expert system to enhance the level II control of a 4-high stand hot reversing mill at Alcoa Europe FRP, Kitts Green. Expert system allows the expertise and judgement of the operators to be stored into computer programs. This expertise and judgement are used in setting up the mill before rolling. In developing this expert system, the task is to construct a knowledge base that contains the strategies required for mill set up and supervisory control.
3.1.1 Knowledge representation

A person is considered as an expert in a specific field when he or she possesses superior and specialized knowledge about a domain. In the study of expert systems, this type of knowledge is known as domain knowledge. The expert gains skills through experience to enable him or her to solve the problem effectively. An expert system can be considered as a computer program that is able to mimic the intelligent behaviour of an expert in performing a particular task. It reasons with domain knowledge acquired from the human expert with a view to solving problems or providing useful advice. The task of the knowledge engineer is to capture the expertise of the expert into the expert system. After acquiring domain knowledge from the expert, he/she has to organize and structure the knowledge into the expert system so that it can be used to solve problems in a manner similar to that followed by the expert. The various methods which are used to encode knowledge into the knowledge base of an expert system are formally known as Knowledge Representation methods. The general problem of knowledge representation is to develop a sufficiently precise formal method for representing a particular domain knowledge. Despite the existence of numerous knowledge representation schemes, adopting a knowledge representation method remains a difficult problem. Some domains of knowledge, i.e. mathematical knowledge, are relatively straightforward to cope with, due to the precise definition of knowledge involved. Unfortunately a majority of domain knowledge areas do not have simple definition of objects. It is therefore very important to consider the types of objects and their relationships in the problem domain when choosing a knowledge representation scheme. One way is to consider the problem domain to contain a set of individual objects with a collection of relationships existing between them. The collection of all objects and their relationships at any one time constitutes a state. There can be state transformations that cause the creation or destruction of objects, or that can change the relationship among them. Depending on whether the key starting point for a representation scheme is the objects and their relationships or the state transformations, leads to a frame or a production scheme respectively. Examples of other knowledge representation schemes include semantics networks, scripts, objects and many more.

3.1.2 The main structure of an expert system

In general, an expert system has three main components to artificially represent a problem solving environment:

- Working memory
- Knowledge base
- Inference Engine
Most of the commercially available expert system shells normally have other subsystems which make building a problem specific expert system more convenient. Generally, an expert system shell has the following additional subsystems:

- User Interface
- Interface to External Programs
- Developer's Interface

Figure 11 shows the main structure of an expert system.

The following subsections describe the functions of the individual components in more detail.

**Working memory**

The working memory is the central memory area used to store objects that represent facts about the problem. These facts are generally referred to as primitive facts and can be provided by the user or inferred from other facts by triggering rules from the rule knowledge base. Generally, primitive facts represent the working hypothesis to the problem domain, and a set of primitive facts can uniquely define a state of the problem solving process. Primitive facts are used by the inference engine to trigger rules in the rule knowledge base, and in turn they can be modified by the rules. The absence or presence of certain fact elements in the working memory will activate some rules to which the pattern of their premises are matched. Primitive facts can be added, modified or deleted from the working memory in the light of subsequent information and direct the problem solving process to progress to another state towards the goal.

**Knowledge base**

A knowledge base of an expert system contains all the facts, rules and possible methods about the application. It might also contain the conclusions of the problem. The form that the rules are in is similar to conditional statements in conventional
programs, except that the data is replaced with facts. A dedicated inference engine provides the execution mechanism. This is different from conventional programs where the domain knowledge and execution control are closely inter-linked such that the knowledge is implicitly stored in the program. This explicit separation of the knowledge from the control mechanism makes it easier to examine knowledge, incorporate new knowledge and modify existing knowledge.

**Inference engine**

The inference engine is a knowledge processor which mimics the expert’s reasoning process. Essentially the function of the inference engine is to draw conclusions based on all available information coupled with the expert’s knowledge stored in the knowledge base as if-then rules, where the *if* part contains the pre-conditions and the *then* part the action or conclusion. Rules are linked or chained together by the inference engine that matches the conditions of one rule to the conclusions of another. There are two main methods for inference, forward chaining and backward chaining.

![Figure 12 Search direction of forward and backward chaining](image)

Forward chaining is a fact-driven search approach in which the reasoning process begins with some known primitive facts in the working memory. New facts are derived to progress towards the aim of reaching a state where all the primitive facts in the working memory are enough to support the goal state. The search direction of a forward chaining rule based system which initiates form the bottom of the search space and moves toward the goal at the top, as depicted in Figure 12, results in this search approach being known as the bottom-up approach.

Backward chaining is the testing for specified data to prove a particular goal or hypothesis. This is often referred to as goal-directed reasoning. This is harder as the user has to state all the relevant information as facts in advance, before the reasoning process is started. The backward chaining inference engine will first check to see if the
goal exists in the working memory. If the goal has been proven, the inference engine will search through all the rules to find one which contains the goals to its conclusion part. This particular rule is generally referred as the goal rule. The inference engine continues to prove the premises of the goal rule by matching them against the contents of the working memory. The premises which do not exist in the working memory become the new sub-goals to be proven by the inference engine. The sub-goals may in turn be supported by other rules in the rule knowledge base, and the inference engine will continue the goal proving process until it finds premises which constitute the primitive facts in the working memory. Due to the nature of its goal proving inference mechanism, the backward chaining is often called the top-down approach as depicted in Figure 12.

Forward chaining is the simpler of the two chaining methods and is appropriate when the ultimate goal is not known. Expert tasks are usually intricate, and call for a combination of chaining in both directions.

**Developer Interface**

All expert system development packages offer different facets to the programmer to build their problem specific expert system. A good programmer interface will provide an easier path for the programmer to encode all the problem specific knowledge elicited from the human expert to build rules in the rule knowledge base and characterize the behaviour of the interface engine to mimic the human expert reasoning heuristic. Generally, the programmer is provided with various ways to incorporate and develop problem domain knowledge into the expert system shell, either by the traditional 'source code entry and compilation' approach or through a smart question and answer editor via an intelligent front end. In general, the latter approach is more user-friendly. However, the former approach is more widely adopted due to its flexibility in using the lower level commands and, consequently, the programmer can gain better insight into the functions and capabilities of expert system shells.

**User Interface**

A user interface caters for smooth communication between the user and the system, also providing the user with an insight into the problem-solving process carried out by the inference engine. This communication is, in most cases, supplemented by graphical means. The interaction between the user and the expert system is normally a bi-directional communication. The user may supply useful data which describes the problem to be solved while the expert system continuously updates the user by displaying the intermediary states during the problem solving process.
**External software interface**

Due to the formal syntax of the rules in the rule knowledge base to cater for the flexibility in symbolic manipulations, numerical computations needed for solving numerically-rich and algorithmic problems are commonly not well supported by the rule-based expert system. To cope with such a deficiency in expert systems, most of the commercially available expert system shells offer an open architecture that allows the programmer to interface with external software packages. The programmer is able to write a simple interface program so that data and functions from some external software packages can be called within the rule knowledge base. This capability adds to the utility of the expert system, so that problem specific information which can be processed and stored naturally in the external programs can be easily accessed and used by the expert system.

**3.1.3 Selection of the expert system shell**

Expert systems can be developed using logic programming languages such as Prolog, C++ or Lisp. Nevertheless, the development of an expert system from scratch using these languages is generally laborious and requires a great deal of man-power and time. Fortunately, there are numerous powerful expert system development toolkits, generically known as expert system shells, available to be purchased, which provide a more productive and inexpensive way to develop a problem specific expert system.

An expert system development toolkit can provide a quick means of developing a working prototype with minimum requirement of programming capability and prior experience in building a complete expert system. The predefined inference engine and the formal syntax of the knowledge base, allows the user to concentrate on developing the domain-specific knowledge base. A well written expert system toolkit is generally reliable, which results in shorter debugging time whilst normally being sophisticated enough for the complexity of the task. Usually extensive support facilities are provided for debugging together with sufficient built-in functions which are crucial for rapid prototyping. The resulting system is easy to maintain and expand.

In an earlier expert system developed at the University of Leicester to prove the effectiveness of expert systems in supervisory control of rolling mills, [38] uses Flex which is an expert system shell written in the computer programming language Prolog [70]. Unfortunately, this is out of date with today's Artificial Intelligent (AI), technologies and unsuitable for use by a non-technical person. It would be useful to have a package, which does not only incorporate the necessary rules, but is also user-friendly such that it can be used by non-experts in metal processing or AI.
In this project in which we develop an expert system for the supervisory control of an aluminium rolling mill, the software package G2 from Gensym Corporation is chosen to develop the expert system. G2 is a real-time Expert System, with a large installed user base that has proved its stability in industrial environments. The product was chosen ahead of competitors due to functionality, connectivity, support, costs and future life cycle developments [71].

The dynamic and complex decision support facilities make G2 powerful by enabling the complexity of operations to be brought under control through intelligent operations management. It allows knowledge from the operations experts to be captured. It intelligently combines that knowledge with real time data and archival information. Powerful reasoning engines in G2 analyse all these inputs in real time and develop the best possible operating decisions, either as operator recommendations, or as automated actions. Therefore, the mill operation's consistency, efficiency, flexibility and quality are anticipated to be dramatically improved.

G2 enables application developers to represent knowledge as objects, rules, methods and procedures using graphics and a structured natural language. This allows applications to be readily understood, tested and modified. With all these strengths knowledge of rolling mill models can be applied dramatically.

G2 is able to continuously monitor multiple asynchronous events and data streams simultaneously in real time [72]. The contained knowledge is turned into useful information by G2 reasoning from developer’s rules. Its most important concept is the object-oriented technology described in section 3.1.5. G2 code is also portable across different computer vendors. Gensym’s G2 Real Time Expert System integrates multiple technologies, so the developer can deploy applications faster, by an order of magnitude.

G2’s ability to reason about time is essential to the process industry. Multiple threads in G2 can execute parallel operations significantly faster than the serial execution in Flex. The use of a structured, natural language in a graphic-oriented environment cannot be found in Flex. G2 provides better debugging features, such as highlighting the rules when they are invoked which are not possible with Flex.

G2 has multiple interfaces that allow it to interact with other processes and to receive data from external sources. One of these interfaces used is the G2 File Interface (GFI). GFI is a versatile utility that allows G2 to write to and read from data files. The data files can be created and read by outside programs, such as database or spreadsheet programs, as well as by G2. Thus the developer can use GFI to create a file and then to analyse it with an outside program, or the developer can create a data file with an outside program and use it in G2 with GFI. This is applied in the developed expert
system, where the PDI is stored in Excel, and Excel is used by G2 to analyse and plot graphs on the operation. Flex does not have equivalent bridge products.

Telewindows is a remote windowing system that allows several users to access the same G2 simultaneously. Each Telewindows user has all the capabilities of a G2 user. This is a very useful tool as it allows several developers to work in the same application at the same time. This utility was used in developing the expert system, where a few research staff and the postgraduates were able to work simultaneously on different aspects of the project. When the developer uses Telewindows to access a G2’s knowledge base, the G2 process is the server and the Telewindows user is its client.

### 3.1.4 Basic components of G2

In G2, applications are stored in knowledge bases. A knowledge base file (with extension .kb) contains all the information that it needs to run. G2 applications can have a single file or multiple files, giving rise to many parts of the knowledge base. Knowledge comes in many forms within G2, including standard rules. Knowledge appears as objects, which represent the physical systems in the application and the connections between them. Definitions are a way of representing knowledge, as they describe the common features of the objects. Rules, methods and procedures, which describe the behaviour of the objects in the real-time environment, are other means of representing knowledge. Graphical user interface components enable the end-users to interact with the applications, hence affecting the interaction with knowledge.

G2 applications typically consist of numerous knowledge base files, each of which contains one or more modules. A module is a set of related information contained in a knowledge base. A knowledge base must define at least one module. A module that does not require any other modules is an independent module, and a module that requires one or more to run is a dependent module. G2 represents the modules of an application in a hierarchy to show the module dependencies. The module at the top of the hierarchy is called the top-level module. The G2 Foundation Resource (GFR) module helps to manage the modules in an application, for example, which modules G2 initialises first and which modules determine the overall behaviour of the application.

Most G2 objects are located on a workspace. A workspace is an area of the knowledge base that contains objects and that sits on the background of the G2 window. Objects and definitions on a workspace are created, by using the knowledge base workspace menu. The two major purposes of using workspaces are to organise the knowledge in the application and to display an end user interface for the application. Workspaces represent a convenient way of storing and displaying information in the knowledge
base. Name and colour are used to identify the different workspaces in a knowledge base.

An item is the top-level class of everything within G2, including workspaces, definition classes, rules, methods, and procedures. User-defined classes can inherit their definitions from the item class or any sub-class of item. While many instances of classes in G2 are actually items, for example, workspaces, rules, procedures, and methods, we typically refer to them in object-oriented terms as objects, rather than items.

An object gets its definition from its associated class. Every object is an instance of its class. In object-oriented terms, an instance represents just one occurrence of potentially many occurrences of a class. Thus, an object has only a single class, whereas a class can have many instances. The main reason to create class definitions for objects is to eliminate redundancy in a knowledge base. The class definition is created only once, and the unique attribute values in each instance are specified. Another reason for creating class definitions is to avoid mistakes when the attributes of an instance are specified. Every instance of a class has the same attribute specification by default.

Variables and parameters are used to keep a history. The history of an attribute’s values are kept for numerous reasons such as plotting data on a graph, logging and reporting, computing statistical information and performing temporal reasoning, which allows the user to reason about objects over time. A parameter is a special type of object to keep a history of values over time. A variable is a similar type of object which keeps a history and allows us to connect to real time data.

Method is a named object associated with a particular class. This helps to improve encapsulation. Method executes a sequence of actions when the application starts the method. In object-oriented terms, a method represents the object’s behaviours.

Procedure is a named object that executes a sequence of actions when the application starts the procedure. A procedure is independent of any class. Procedures and methods are used to manipulate objects and their relationships in real time as the core repository of system knowledge in the application.

A rule is an object that tests conditions and draws conclusions. G2 can invoke rules in two ways, depending on whether it uses event-driven processing or uses data-driven processing. Rules represent the heart of G2’s inference engine, containing the operator’s knowledge. The heart of any expert system is its ability to reason about the knowledge it contains. G2 uses rules to reason about the knowledge contained in the application. A rule has two parts: firstly, the antecedent, which tests a condition, and
secondly, the consequent, which draws a conclusion. When a rule is invoked, G2 evaluates the rule by testing the condition in the antecedent to see whether it is true. If the condition is true, G2 executes the actions in the consequent. The rules perform data-driven processing by testing the condition in the antecedent and taking actions in the consequent if the conditions are true.

3.1.5 Object oriented philosophy in G2

An advantage of G2 is that it is fully object oriented, allowing a well-structured code to be produced. Historically, a program has been viewed as a logical procedure that takes input data, processes it and produces output data. However in object-oriented programming (OOP), the emphasis is not placed on the logic of programming, but placed on the way the objects are manipulated. The way objects (rather than actions) are organised, is crucial to object-oriented programming.

An object is a piece of information that contains all related knowledge in one location. All the data that defines the object and all the operations that it can perform are all contained in an object. Attributes are defined as an object’s data, and methods as the object’s operations. An object’s type is known as its class.

The benefits when object-oriented programming is used are as follows, as summarised by Robert Lafore [73].

- The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics. Called inheritance, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.
- Since a class defines only the data it needs to be concerned with, when an instance of that class (an object) is run, the code will not be able to accidentally access other program data. This characteristic of data hiding provides greater system security and avoids unintended data corruption.
- The definition of a class is re-useable not only by the program for which it is initially created but also by other object-orientated programs (and, for this reason, can be more easily distributed for use in networks).
- The concept of data classes allows a programmer to create new data types that are not defined in the language itself.

OOP techniques can be used to encapsulate knowledge at the appropriate level in the class hierarchy due to the fact that classes can inherit their definitions from any class. To encapsulate knowledge means to organise related knowledge together in a single object so it can be shared with other objects. Encapsulation is used in the development of the expert system so that objects share the existing knowledge, and complexity can be hidden in their definitions.
3.2 System Design

This section describes the architectural aspects of designing the Expert System, so that it augments the existing system for improved supervisory control [1, 16, 17]. The Expert System is to concentrate on utilising human knowledge, whilst the existing system is to provide the infrastructure for the overall system. The design of the system emphasises the strengths of the Expert System technique, i.e., reasoning about plant knowledge. Figure 13 shows the schematic of the proposed level II supervisory control system.

![Diagram of level II supervisory control system]

Figure 13. Schematic of the level II supervisory control systems

The existing supervisory control system, Hot Mill Adaptation System (HMAS) at Alcoa's Kitts Green plant is supplied by VAI. HMAS was recently commissioned to provide state of the art model-based adaptation control. But it has a number of features that may be improved upon to obtain better control. For example, it was not intended to run fault diagnostics and process optimisation algorithms. There is also a possible lack of accuracy in models. Although this is corrected by adaptation after a few passes, initially accurate predictions of plant measurements would minimise the amount of adjustment made by adaptation and maximise the amount of time that the plant is being correctly controlled.

The shape of plate product from the rolling mill is manually controlled via the existing level II system. The human operators are able to visually quantify shape e.g., wavy edge, and can suggest possible causes e.g., lack of sprays on centre of work rolls. Such human expertise needs to be compiled into the knowledge base so that the Expert System is capable of controlling the shape problems wherever they originate and adopting suitable mechanisms for improvement via a control strategy selection.

Soaking pit optimisation and fault diagnosis can add to the system, whilst shape optimisation focuses the functionality on an increasingly important area. Although
improving just the model parameters and selecting an appropriate rolling strategy would have been beneficial, this may not solve the shape problems on plant.

Fault diagnosis/prediction is not part of the current control system for either the plant or product. The method adopted is for the Expert System to identify important faults, predict their occurrence and automatically diagnose the cause with remedial action suggested.

The method employed to develop the supervisory control system, concentrating on the design rather than implementation, is as follows:

1. Improvements that can be added to the control are generically identified.
2. The overall aim and scope of the system are decided.
3. Modes of system operation are decided.
4. The method for combining the adapted process models with the Expert System is determined.
5. The Expert System is designed, including the functional definition [74].
6. The test system is designed.
7. Data are gathered, the knowledge base is constructed and industrial implementation is carried out [75, 76].

3.2.1 Architecture

The platform of the proposed intelligent knowledge-based supervisory control system is a Windows NT based Personal Computer, selected because it is affordable, available and well understood. Links to existing plant data and systems are straightforward with many technologies and protocols available.

Some additional computer programs are employed for specific tasks. Plant signals are conditioned using National Instruments SCXI back-plane signal conditioning cards. The National Instruments LabVIEW [77] graphical programming language is used to manage this information and coordinate with stored plant information, plant models and man machine interfaces. The stored plant information, such as table lengths, is held in a spreadsheet format for easy maintenance. The plant models and adaptation routines are written in the C programming language [2] due to the speed of operation.

The functionality of the complete system is distributed, as many of the components reside outside the Expert System. Therefore, an important aspect of system architecture design is to decide how the Expert System links to other units of the complete system; especially how the Expert System cooperates with the existing level II controller to fulfil supervisory control tasks. The main decisions taken are as follows:
1. The Expert system is connected to the local network as shown in Figure 13. This network connection allows the Expert System to easily communicate with other networked units in the system, such as the existing level II controller, level I gauge controller and level III plant wide scheduling manager, via TCP/IP protocols.

2. Some plant devices, which do not have any network connections, are indirectly connected via the existing level II system. The indirect connection is implemented via the Direct Data Link (DDL) between LabVIEW and G2.

3. One of the Expert System’s functions is to be a knowledge-based agent. That means the existing level II system can trigger the Expert System to provide a knowledge-based function to generate the schedule. This is implemented via the remote ActiveX Link provided both in LabVIEW and G2 using the COM/DCOM protocols.

4. One of supervisory tasks is to on-line detect any product errors and correct them immediately at real-time via re-generated set-up schedules. Once the Expert System has detected a serious product error in previous passes, it needs to inform the existing level II system to re-generate the schedule to correct the error in the following passes. This trigger is implemented via another DDL link between LabVIEW and G2.

5. The complete system maintains a database for various alloy-related metallic properties. This database is accessed by the Expert System so that engineers only need to maintain one copy of the database in the system.

3.2.2 **Expert system design**

The Expert System could replicate the existing system functionality and infrastructure, but this would lead to a duplication of effort. More importantly, the plant signals could not be split without expensive signal boosting and conditioning. Therefore, the Expert System is designed to work with the existing system, in a way that can still test the ability of Expert Systems to enhance level II control systems. A minimum of data processing and handling is still needed to convert the preconditioned plant signals into conditions for the knowledge base, see Figure 14.

Integration with the existing system focuses on distributing tasks to the programmes and functions best suited to perform them. The trigger for using the Expert System could have been solely from the Expert System, but until confidence on shape improvements is gained, the cost of incorrect action is too great. Instead shape prediction methods within the existing system are used to trigger the Expert System. Once triggered, a complete schedule is passed back to the existing system for rolling.
Figure 14. Infrastructure and data processing components of an expert system.

An important advantage of using Expert Systems instead of conventional programming languages is the visibility of knowledge. In order for operators to interact with the Expert System a human machine interface (HMI) is required. Shape is an important consideration, so feedback of resulting shape observed by the operators from the generated schedule is needed through the HMI. A facility to port Expert System information throughout the plant to engineers and managers is provided by the Telewindows function of G2. Figure 15 shows the functionality of the user interface.

Management of the Expert System, as shown in Figure 16, is required if the knowledge is to be utilised in the best manner. Several modes of operation were investigated: on-line, off-line and 'what-if'. Each is considered to have advantages, so the Expert System is designed to incorporate all three.

The soaking pit optimisation is designed to improve the throughput of product by calculating the timings required for heating the slabs up to the required soak temperature, see Figure 17. Correctly anticipating the loading time is important so that plant is not delayed waiting for slabs to reach the required temperature. Controlling the order of slabs into the mill is important as it is thought to have an effect on shape due to differential heating of work rolls as the product width changes.
The standard fault prediction practice of using statistical process control on alarm conditions of plant equipment is perceived as adequate on plant. However, diagnosing the cause of product quality issues is often a time consuming process. Expert System can utilise human knowledge to improve the speed and consistency of identifying the cause of product quality.
The Expert System can use strategy curves, models and adaptation code from the existing system, whilst enhancing these functions with expert knowledge. It is also
intended to predict shape from the schedules generated in the existing system and the Expert System, with the actual shape entered by operators. The fault and soaking pit information will also be integrated into the schedule generation function. These are coupled with a simulation of any schedule generated, see Figure 18. Therefore, any anomalous events are detected before they occur in real-life, allowing corrections to be made in the schedule before a critical situation is reached.

### 3.3 Project development using G2

In this section, the work done in developing the different aspects of the project is given.

#### 3.3.1 Designing the knowledge base

The knowledge base designed for the supervisory control of the rolling mill, is based upon a combination of a plant and process viewpoints. It is designed to have general application for various kinds of aluminium rolling mill processes, especially for hot rolling mills. It can be divided into two main sections, specific (to a plant) and general (to most plants), to make it reusable.

The plant specific section is the top-level module that consists of Alcoa.KB. This module contains the operator experience, the plant dimensions and all the site information. In order to make the knowledge base reusable, maintainable and available, the general section is split into six appropriate modules as shown in Figure 19. These modules consist of rule bases that govern fault diagnosis, the product, the mill set-up, the mill itself, the operational models, and the interfacing.

![Figure 19. Design of knowledge base](image)

#### 3.3.2 Module hierarchy

The knowledge base described in the previous section can be developed from smaller and more manageable sections called modules. Each module contains a set of related items that together comprise a knowledge base. By using modular development, the codes can be made to be reusable. An added advantage is that several developers can work on a single application by each working on a separate module. These modules
can later be combined to form the entire application. Besides this, class definitions and knowledge that are saved in a single module can be used across multiple applications.

Figure 20 shows the module hierarchy of this application. A set of related modules that have dependencies between them can then be clearly identified.

From Figure 20, it can be seen that there are multiple directly required modules below the top-level module, and that each sub-module can also directly require one or more modules. By developing the modules to be independent of one another, items associated with each module can be stored into separate knowledge bases, assisting the concept of encapsulation. By using the module design, the knowledge in each module can be independently added or changed. In addition, the complete knowledge base can be reloaded by loading all the required modules. Modules from one knowledge base can also be merged into a different knowledge base. Thus modules can be worked both interactively (by the user) and programmatically (by actions in the code).

The Fault Diagnosis module is made up of two sub-modules, namely the product diagnosis and the plant diagnosis. This module mainly allows adjustments to be made to the mill when important problems are identified. Urgent problems are passed through the interface for the operators to take immediate corrective actions.

The Product module contains information related to the product. The product rule base in this module is designed to perform product checking, to ensure that sufficient material is available and to handle scheduling.

The Set-up module governs the mill set-up systems. The current set-up algorithm is used as the basis for this module, with the inclusion of other rule bases to enhance the mill set-up. Actions required to perform the mill set-up are included in this module.

The Mill module contains the physical knowledge for all other rule bases in the system. General plant information that does not change during rolling and knowledge on the types of mill plant are stored here.

The Operational Models' module has six sub-modules: Thermal, Stress, Bending, Sprays, Roll Eccentricity and Shape.

The Interface module handles the connection between the expert system and the external sources. This module allows Level II of the control system, which includes the expert system, to be able to interface with Level I and Level III. This module is also in charge of the links to the data files and other resources.
Figure 20. Module hierarchy of the rolling mill
**Developer's Interface**

The developer builds and runs an expert system through the developer’s interface. The developer builds the expert system on workspaces. A workspace is an area of the knowledge base that contains objects. Workspaces have different colours to identify them with a given module. By using workspaces, the information in the knowledge base can be organised and displayed. Applications often display and animate workspaces to communicate information to end users about the application. The icon representation of any object must reside on a workspace. Figure 21 highlights the main workspaces used in this application.

From Figure 22, it can be clearly seen that the workspaces are divided into 4 levels. The top-level-module, level 1, level 2 and the base level, which is global. The main workspaces created for the rolling mill application is detailed in Figure 22.

![Figure 21. Interface with the knowledge base](image-url)
Figure 22. Workspaces created for the development of the expert system
From the knowledge base design (see Figure 19), the workspaces and the sub-workspaces are created (see Figure 22). In order to understand the concept of the object-oriented philosophy used in G2, a few of the workspaces will be explained in particular.

The following are the workspaces chosen to be studied:

1. BASE-WS
2. PH-DEFINITIONS-WS
3. PL-DEFINITIONS-WS
4. SPR-RULES-WS

**BASE-WS**

BASE-WS is a workspace that is in the base level. It is global in nature, and the 45 objects defined in this workspace can be used by other modules. These 45 objects can be divided into 9 classes and 36 parameters, as shown in Figure 23. The details of some of these objects created are found in Figure 24 and Figure 25.

The flexibility of G2 allows the user to create user-type definition, such as defining ‘True=1’ and ‘False=0’ in the C programming language. By creating the user type definition, the concept of object-oriented programming is not broken. The use of global variables in BASE-WS contains all such user-type definitions. It must be at the base of the module hierarchy, so that all the modules can use the user-defined types.

The two distinct types of user defined definitions are classes and parameters. Normally a class is used to define the general structure of an instance or to be used as a superior class (a class that is located above another class in the hierarchy). None of the defined classes in BASE-WS should be used as a superior class. Similarly, there are no instances created from the classes in this workspace. Instead, global classes should define user types for attributes in other classes.

Global definitions such as ‘uncertain’ and ‘incorrect’ are programmer related. However, definitions for parameters such as max-roll-bend and min-roll-bend are not central to the programming environment. These are site-specific parameters. A decision was made to gather all such parameters in a single location. Although not object-oriented, it allows them to be removed for commercial consideration when deploying the expert system on another site.
Figure 23. Screen capture of BASE-WS
The PH-DEFINITIONS-WS workspace illustrates the creation of instances from classes. Figure 26 shows the PH-DEFINITIONS-WS. Object instances get their definitions from the classes, and the attributes of an instance can be viewed from the table as shown in Figure 27. The total-slab-state class definition describes the common characteristics of its instances, for example, the width attribute, the length attribute and the gauge attribute. Each instance inherits its definition from its class. Each instance has the same icon but a different name and different attribute values. This illustrates one of the important concepts of G2, that is, inheritance whereby an instance obtains its attributes from its class and additional superior classes. Attributes of the instances can be manually overridden to suit the specification. The table for the instance Entry-state is as shown in Figure 26.
Figure 26. Illustration of instances and the table for the instance

Icon: Class

<table>
<thead>
<tr>
<th>No</th>
<th>Class name</th>
<th>Direct superior classes</th>
<th>Class specific Attributes</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>METAL</td>
<td>Object</td>
<td>Alloy</td>
<td>Quantity</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>SLAB</td>
<td>Metal</td>
<td>Length</td>
<td>Quantity</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Width</td>
<td>Quantity</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Gauge</td>
<td>Quantity</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Temperature</td>
<td>Quantity</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Profile</td>
<td>Quantity</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Shape</td>
<td>Quantity</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Homogeny</td>
<td>Quantity</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>SLAB-STATE</td>
<td>State</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>MILL-STATE</td>
<td>State</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>WIDTH</td>
<td>Slab-state</td>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 27. Examples of class definitions in PH-DEFINITIONS-WS
Icon: Instance

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ENTRY-STATE</td>
<td>TOTAL-SLAB-STATE</td>
</tr>
<tr>
<td>2</td>
<td>EXIT-STATE</td>
<td>TOTAL-SLAB-STATE</td>
</tr>
<tr>
<td>3</td>
<td>THIS-SLAB-PDI</td>
<td>SLAB-PDI</td>
</tr>
<tr>
<td>4</td>
<td>NEXT-SLAB-PDI</td>
<td>SLAB-PDI</td>
</tr>
<tr>
<td>5</td>
<td>LEARNT-SCHEDULE-PDI</td>
<td>SLAB-PDI</td>
</tr>
<tr>
<td>6</td>
<td>TEMP-SPEED-DERIVATIVE</td>
<td>DERIVATIVE</td>
</tr>
<tr>
<td>7</td>
<td>PROFILE-BEND-DERIVATIVE</td>
<td>DERIVATIVE</td>
</tr>
<tr>
<td>8</td>
<td>ENTRY-GAUGE-MEASUREMENT</td>
<td>MEASUREMENT</td>
</tr>
<tr>
<td>9</td>
<td>EXIT-GAUGE-MEASUREMENT</td>
<td>MEASUREMENT</td>
</tr>
<tr>
<td>10</td>
<td>ENTRY-WIDTH-MEASUREMENT</td>
<td>MEASUREMENT</td>
</tr>
</tbody>
</table>

Figure 28. Examples of instance definitions in PH-DEFINITIONS-WS

**PL-DEFINITIONS-WS**

The PL-DEFINITIONS-WS workspace is described to illustrate the functions of procedure. Figure 30 shows the procedure MEASUREMENT-CHECK used in the PL-DEFINITIONS-WS. In this example, a procedure programmatically checks the measurement, and determines whether the standard deviation is large or small. In order to build up the procedure, many objects were defined as shown in Table 1.

![Screen capture of PL-DEFINITIONS-WS](image)

Figure 29. Screen capture of PL-DEFINITIONS-WS

<table>
<thead>
<tr>
<th>AGREEMENT</th>
<th>Integer</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALL</td>
<td>Symbol</td>
<td>Small</td>
<td>Small</td>
</tr>
<tr>
<td>LARGE</td>
<td>Symbol</td>
<td>Large</td>
<td>Large</td>
</tr>
<tr>
<td>BELOW-CONTROL-LIMIT</td>
<td>Symbol</td>
<td>Below-control-limit</td>
<td>Below-control-limit</td>
</tr>
<tr>
<td>NORMAL</td>
<td>Symbol</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>ABOVE-CONTROL-LIMIT</td>
<td>Symbol</td>
<td>Above-control-limit</td>
<td>Above-control-limit</td>
</tr>
</tbody>
</table>

Table 1 Examples of parameter definitions in PL-DEFINITIONS-WS
measurement-check(M: class measurement)
begin
  if (the name of M is exit-temperature-measurement or
      the name of M is layon-temperature-measurement or
      the name of M is entry-temperature-measurement) and
      the standard-deviation of M > max-temperature-sd
      or (the name of M is entry-profile-measurement or
           the name of M is exit-profile-measurement) and
           the standard-deviation of M > max-profile-sd
      or (the name of M is entry-shape-measurement or
           the name of M is exit-shape-measurement) and
           the standard-deviation of M > max-shape-sd
      or (the name of M is load-measurement or
           the name of M is power-measurement) and
           the standard-deviation of M > max-other-sd
    then conclude that the type-of-sd of M = symbol (large)
  else conclude that the type-of-sd of M = symbol (small);
{check drift}
  if (the name of M is exit-temperature-measurement or
      the name of M is layon-temperature-measurement or
      the name of M is entry-temperature-measurement) and
      the drift of M > max-temperature-drift
  .......

Figure 30. Procedure definitions in PL-DEFINITIONS-WS
**SPR-RULES-WS**

This workspace illustrates the functionality of rules used in G2. It is possible to highlight the rules when they are invoked at the time to show the operational sequence. If the rule has a scan interval of 0.2 seconds as shown in Figure 31, the rules will be invoked by G2 at every 0.2 seconds.

---

**If the roll-camber of the camber-profile of any work-roll < -15 then**
- conclude that the current-value of the centre-spray-level of every pass-schedule-data in current-schedule = the current-value of the e-spray-level of cold-spray-pattern and
- conclude that the current-value of the edge-spray-level of every pass-schedule-data in current-schedule = the current-value of the e-spray-level of cold-spray-pattern and change the text of state-of-work-rolls to "cold"

**If the roll-camber of the camber-profile of any work-roll < -5.0 and the roll-camber of the camber-profile of any work-roll > -15.0 then**
- conclude that the current-value of the centre-spray-level of every pass-schedule-data in current-schedule = the current-value of the e-spray-level of warm-spray-pattern and
- conclude that the current-value of the edge-spray-level of every pass-schedule-data in current-schedule = the current-value of the e-spray-level of warm-spray-pattern and change the text of state-of-work-rolls to "warm"

**If the roll-camber of the camber-profile of any work-roll >= -5.0 and the roll-camber of the camber-profile of any work-roll <= 5.0 then**
- conclude that the current-value of the centre-spray-level of every pass-schedule-data in current-schedule = the current-value of the e-spray-level of "steady"

**If the roll-camber of the camber-profile of any work-roll >= -5.0 and the roll-camber of the camber-profile of any work-roll < 15.0 then**
- conclude that the current-value of the centre-spray-level of every pass-schedule-data in current-schedule = the current-value of the e-spray-level of "hot"

**Figure 31. Screen capture of the SPR-RULES-WS**
End user's interface

**ALCOA-MIMIC-WS**

Figure 6.17 illustrates the rolling mill process.

The results of the test of spray rules, and transferred infrastructure, are shown in Figure 33, Figure 34, and Figure 35. Figure 33 shows the temperature distribution across the work roll versus the camber across the work roll. The camber profile is relatively smooth with a difference of 2.5 units, which is within the acceptable range. Figure 34 shows the temperature versus camber distribution after several plates have been rolled, without the knowledge base being used. The increase in difference is from
2.5 to 7.0, between the highest and the lowest camber. The profile is significantly sharper increasing the likelihood of introducing undesirable shape into the plate. Figure 35 shows the temperature versus camber distribution after spray rules have been enabled. The difference has decreased to 4.5 units, which is in the acceptable range. The profile is smoothed.

3.4 Rule formulation

This section shows how data-mining and knowledge-elicitation techniques are combined to construct the knowledge base of the expert system. Methods used to extract information from stored plant data and captured knowledge from experienced plant personnel are outlined. Details of how this knowledge is fused into an intelligent knowledge-based system are given. An example of the generated schedule, which consists of mill set points used to control the mill, is analysed to highlight the effect of the additional knowledge. Conclusions are drawn on the best methods for utilising both data mining and knowledge elicitation for supervisory control of modern plant.

The present level II supervisory control system incorporates a number of physically based process models. The use of such models makes it difficult to incorporate imprecisely formulated knowledge. Rules of thumb (heuristics), which are known to help improve product quality, are lost. The expert system is developed to handle this type of knowledge. However, the designer of such a system is faced with the problem of systematically formulating rules. Interviewing mill operators and production personnel is a proven route for knowledge elicitation. However, conflicts and contradictions (although minor) were discovered in defining rule values. Similarly, the areas of expertise did not completely overlap with the scope required to implement a supervisory control system, e.g. operators do not need to predict mill load, which is required by gauge control systems. Moreover, the Expert System can only reason about the knowledge that is entered into it. If gaps in human knowledge exist about plant processes, there will be gaps within the Expert System functionality. A rich and often ignored source of knowledge is logged data. Fortunately, there exist many data mining techniques for learning information from plant data, which the Expert System can utilise [76].

The Kitts Green mill-scheduling computer is linked to a plant wide level III production planning system. Details of the slabs that are waiting to be rolled and the plates to be produced are downloaded to this computer. Based on this information and the temperature of the slab, the level II system determines the pass schedule to be used to roll a given slab. Following the rolling of each piece, the schedule used to roll that piece is returned to the planning system. The schedule data is augmented by
measurements made whilst rolling. Two different techniques are used to formulate the rules from this logged data. The techniques are

1) Knowledge Elicitation Rules obtained from plant experts.
2) Data Gathering i) Modelling of the rolling process
   ii) Examination of Logged Data:
       Statistical Methods
       Data Mining.

3.4.1 Knowledge-elicitation

The conventional method for constructing a knowledgebase is to form rules and associated objects from knowledge elicitation. Often a 'knowledge engineer' elicits important facts and heuristics (rules of thumb) from plant experts, e.g. managers, engineers and operators [78]. This process may contain several well-defined stages as listed in Table 2.

<table>
<thead>
<tr>
<th>Step 0</th>
<th>Agreement with Alcoa/ VAI.</th>
</tr>
</thead>
<tbody>
<tr>
<td>phase 1</td>
<td>Initial presentation.</td>
</tr>
<tr>
<td>phase 2</td>
<td>Us questioned by experts.</td>
</tr>
<tr>
<td>phase 3</td>
<td>Separation and identification of topics, issues and concerns.</td>
</tr>
<tr>
<td>phase 4</td>
<td>Focused discussions, conflicts, gaps and expansions.</td>
</tr>
<tr>
<td>phase 5</td>
<td>Structured feedback / follow up.</td>
</tr>
<tr>
<td></td>
<td>Commissioning of ES system</td>
</tr>
<tr>
<td></td>
<td>Use and benefits of ES system.</td>
</tr>
</tbody>
</table>

Table 2. Stages in knowledge elicitation

*Step 0* is a political requirement, which cannot be underestimated in its importance. If the senior managers, engineers and especially operators are not fully supportive of the project, then the project will have reduced impact. Similarly, the developed system must be seen as cost-effective without threatening job security. Therefore, an advisory system is more feasible than a system designed to replace operators, but it is harder to quantify future cost savings. Projected reductions in scrap, reworking and corrective processing may be estimated. Qualitative measures include improved delivery times, perceived product quality and the resultant customer satisfaction.

*Phase 1* is to introduce the concepts behind expert systems, knowledge-elicitation, and data-mining with the aim to create a culture of openness and honesty. This is extended by *Phase 2*, where any fears or misconceptions can be allayed. Importantly, it can help focus the knowledge-elicitation into areas considered important by plant operators. Areas of opportunity and suggested methods of improving supervisory control can also be forthcoming. No rules are formulated at this stage, but potentially irrelevant sources of information can be identified to reduce the problem domain.
Phase 3 is the first time the experts are questioned directly about the operation of the plant. This phase seeks to complete the scope of the system and generate an initial rule-base. Task analysis can be introduced to assist in assigning rule priorities. Once the initial rule-base is evaluated, Phase 4 considers conflicts and gaps within the knowledge. Rule chaining is added to link different types of rules in an ordered manner.

Phase 5 utilises the knowledge base and the results are fed back via the experts to improve the rules. It is probable that complex interactions of rules, not previously considered, become apparent at this point. Rule chaining and priorities must be tuned to give improved schedules.

Several methods can be utilised during phases three and four to extract the core knowledge [75]. The standard techniques are dialogue, scenarios (where a given situation is discussed), task analysis (the steps in a given task are described) and hypothesised rule evaluation. These are complemented with brainstorming, diagrammatic representations (e.g. constructing fish-bone diagrams), and stimulus (e.g. pictures of poor quality shape). Teach-back, where operators teach the knowledge engineer specific parts of the process, can also prove useful. The more fun methods, such as card sorts and 20-questions were considered valuable, but at the risk of trivialising the knowledge gathering process they were not used.

### 3.4.2 Data-mining

An initial decision was whether to develop a single bespoke technique or utilise a commercial package containing multiple techniques. Flexibility, ease of use, acceptance on plant and reduced time to results were factors considered more important than cost and possibly better performance. Therefore, the commercial data-mining package Clementine from SPSS [79] was selected. Also a bespoke technique can be integrated directly into the software at a later stage if greater rule accuracy is found to be necessary.

Prior to actual data-mining, the plant database was interrogated to determine important features. It was found to have high dimensionality with over 120 fields being recorded. Simple data cleansing was performed as repetition and redundancy (fields with no values) were removed prior to loading into the data-mining package. Pre-processing on parameter correlation, irrelevance and contradiction using domain knowledge were not performed at this stage as bias could be introduced. Features such as modality, obliqueness and epistasis were investigated to determine the data-mining techniques to use.
The database consisted of 64 fields linked to several output variables, which included predicted mill load, predicted plate temperature and final plate shape. Final plate shape, recorded by the operator, was the primary selected output variable in the form of position of shape (operator side, middle or drive side) and severity (none, mild or significant). The database may be separated into training and test sets in order to verify the accuracy of mined rules and prevent overfitting to data.

The main method for producing rules was the decision-tree (see [80, 81]), selected due to its ability to handle high dimensionality. If the domain had high epistasis [82], obliqueness [83] and multimodality, then partitioning the domain linearly would not have been the best option. Therefore, the Neural Network rule generator in Clementine was implemented occasionally to check that this was not a problem. Without including the information from knowledge-elicitation, the rules (or branches) often had more than nine conditions making them difficult to interpret. Similarly, information which was later considered redundant or irrelevant was included in the rules.

3.4.3 Fusion of data mining and data gathering

The results of this work may be split into three important components:

1. Methodology for combining knowledge-elicitation and data mining
2. Improvement in the set-point schedules produced
3. Quality of the final plate

The last two results provide validation for the primary goal of this part of the project, which is to postulate methods for rule-base construction by combining knowledge elicitation and data mining. Figure 36 shows the strengths of knowledge elicitation and data mining. These can be used to select appropriate methods at the various stages of constructing the knowledge base as discussed next.

Stage 1. Determine the scope of the knowledge base required to solve the problem. In this application the knowledge base must produce a schedule of set points in order to control the production of aluminium plate. The results of the knowledge-elicitation proved more useful than data mining in the initial stages as experts had an overview
of important and relevant information. Supplying all the data to the data-mining software produced very poor rules that often included spurious, highly correlated and irrelevant elements that had low predictive and descriptive value.

**Stage 2.** Determine the correctness of the scope identified by experts. The data was reorganised with supposedly irrelevant information removed (e.g. date of rolling), highly correlated information reduced (e.g. removing 'side-guide closing distance' as it is close to 'width of plate') and spurious information removed (e.g. operation 1 data removed as shape does not occur in this operation).

New rules were generated through data mining and considered not for accuracy, but for inclusion of important parameters. A class of parameters were discovered that experts had not discussed, but were clearly important for data mining. The class was characterised by being physical parameters, such as load, that the experts had no direct control over. 'Gap-setting' which results in the load value, was given much higher priority by the experts.

This stage was helpful in overcoming 'the curse of dimensionality' [84], where data-mining techniques struggle in high dimensional search spaces. Minimising the search space in size without removing important parameters improved the accuracy of the rules produced through data mining.

**Stage 3.** Separation of domain into important sub-areas. The combination of wide width and low gauge has an important effect on the likelihood of shape formation. Therefore, it is difficult to autonomously separate rules about width to shape and rules about gauge to shape. Knowledge-elicitation is better able to separate out the effect of individual parameters, often because an expert's experience is greater than the depth of the data recorded. In multimodal and multi-optimisation search spaces, knowledge elicitation produced more accurate rules for single parameters. However, data mining was considered suited to single optimisation and single search peak environments.

Therefore, the knowledge base was constructed to be modular with different parts separated and given priority within the expert system [85]. For example, material hardness was known to affect shape performance, with material composition (fixed for a given alloy) given a higher priority than temperature as the rolling schedule produced has weak control over this parameter (e.g. the less passes in a schedule, the higher the finishing temperature, but temperature cannot be strongly controlled as the direct addition of temperature once rolling has commenced is not possible).

**Stage 4.** Critical rule identification. Now that rule categories have been formed, the most critical rules in schedule generation can be identified. At this point knowledge-elicitation and data-mining produced very similar results. It is required to generate a
rule of the form: 'if the plate is significantly wide, then at what gauge could shape quality issues start appearing?' Both techniques gave quantitative values within a couple of millimetres. The experts tended to be more conservative and remember outliers; useful for forming specific rules, but overcautious for general decision rules. This stage was the most iterative with results from data mining being confirmed by experts, the decision rules used to adjust the sub-domain training data and data mining retrained in an attempt to improve accuracy.

Stage 5. Specific rule identification in sub-domains. The data was further partitioned based on the critical decision rules, e.g. the dataset of plates most likely to have shape based on their gauge was formed for a given alloy. The alloy, and hence starting hardness, was fixed to remove one factor from the search space. Every time a significant parameter was removed, the decrease in dimensionality improved the accuracy of resulting rules, but at the cost of reduced data samples. Rules that had been produced from only one or two examples were considered very sceptically [76]. Ideally, 10 or more examples that fitted a given rule were needed for confidence if not supported by an expert opinion. An initial training set of about 300 examples was needed to produce a set of supported rules governing a particular search sub-domain.

Data mining became the main method to discover specific rules, with experts used to validate these rules - no rules formed were rejected outright for being factually incorrect. These rules often had less than five conditions, which could be manually separated to give five simple rules of differing subject, see Figure 37. Experts were better at analysing chains of simple rules than sets of complex rules. Figure 38 shows the simplifying effect on the output of data mining from fusing with knowledge-elicitation. An increase in rule quality was also observed.

Stage 6. Off-line trials, online trials and commissioning. The ES was designed to simulate rolling, from slab specification through schedule generation (main task of the knowledge base) to final plate. Although it could not determine the shape of the plate with mathematical models (it can predict the shape through rules), the simulation highlighted any missing or conflicting rules. In practice, the transparent and modular design meant that it had been straightforward to scrutinise the rule operation and no

### Figure 37. Examples of generated rules

<table>
<thead>
<tr>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>if the final-width of current-plate &lt;= 1200 then conclude that the plate-width of final-plate is narrow</td>
</tr>
<tr>
<td>if the final-width of current-plate &gt;= 3000 then conclude that the plate-width of final-plate is wide</td>
</tr>
</tbody>
</table>

OK, but not active.
INITIALISE-2.0
6 The default is normal
OK, but not active.
INITIALISE-2.1
6
gaps or conflicts were found. However, great care was needed with rule triggering as the ability to forward chain had to be tightly controlled otherwise unexpected scheduling could occur.

<table>
<thead>
<tr>
<th>Data-mining without Fusion</th>
<th>Data-mining with Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGAP &lt;= 1277 [Mode: T]</td>
<td>STWID &lt;= 2034 [Mode: T]</td>
</tr>
<tr>
<td>ROLLERID &lt;= 15 [Mode: T]</td>
<td>FWD &lt;= 2295 [Mode: T]</td>
</tr>
<tr>
<td>ROLLERID &lt;= 9 [Mode: T]</td>
<td>FLEN &gt; 12995 [Mode: F] -&gt; F</td>
</tr>
<tr>
<td>STLEN &lt;= 3305 [Mode: T]</td>
<td>FWD &gt; 2295 [Mode: T]</td>
</tr>
<tr>
<td>CMODE &lt;= 65 [Mode: T] -&gt; T</td>
<td>STGAUGE &lt;= 6085 [Mode: T]</td>
</tr>
<tr>
<td>CMODE &gt; 65 [Mode: T]</td>
<td>Load-Diff &lt;= 309 [Mode: T] -&gt; T</td>
</tr>
<tr>
<td>ROLLERID &gt; 9 [Mode: T] -&gt; T</td>
<td>STGAUGE &lt;= 5466 [Mode: T]</td>
</tr>
<tr>
<td>STWID &gt; 2017 [Mode: T] {FOLDED}</td>
<td>STGAUGE &gt; 5466 [Mode: T]</td>
</tr>
<tr>
<td>ROLLERID &gt; 15 [Mode: T]</td>
<td>PSD &lt;= 904 [Mode: F] -&gt; F</td>
</tr>
<tr>
<td>ADFCT &lt;= 700 [Mode: T] {FOLDED}</td>
<td>PSD &gt; 904 [Mode: T] -&gt; T</td>
</tr>
<tr>
<td>ADFCT &gt; 700 [Mode: T]</td>
<td></td>
</tr>
<tr>
<td>INTALLOY &lt;= 2618 [Mode: T] -&gt; T</td>
<td></td>
</tr>
<tr>
<td>INTALLOY &gt; 2618 [Mode: F]</td>
<td></td>
</tr>
<tr>
<td>MLOAD &lt;= 2173 [Mode: F]</td>
<td></td>
</tr>
<tr>
<td>MLOAD &lt;= 1793 [Mode: T]</td>
<td></td>
</tr>
<tr>
<td>STGAUGE &lt;= 8151 [Mode: F]</td>
<td></td>
</tr>
<tr>
<td>DRAFT &lt;= 601 [Mode: T]</td>
<td></td>
</tr>
<tr>
<td>MGAGE2 &lt;= 0 [Mode: T] -&gt; T</td>
<td></td>
</tr>
<tr>
<td>MGAGE2 &gt; 0 [Mode: T]</td>
<td></td>
</tr>
<tr>
<td>FLOTNO &lt;= 284179170.0 [Mode: T] {FOLDED}</td>
<td></td>
</tr>
<tr>
<td>FLOTNO &gt; 284179170.0 [Mode: F] -&gt; F</td>
<td></td>
</tr>
<tr>
<td>DRAFT &gt; 601 [Mode: F] -&gt; F</td>
<td></td>
</tr>
<tr>
<td>STGAUGE &gt; 8151 [Mode: T] -&gt; T</td>
<td></td>
</tr>
<tr>
<td>MLOAD &gt; 1793 [Mode: F] -&gt; F</td>
<td></td>
</tr>
<tr>
<td>MLOAD &gt; 2173 [Mode: T] -&gt; T</td>
<td></td>
</tr>
<tr>
<td>TARGAP &gt; 1277 [Mode: T] -&gt; T</td>
<td></td>
</tr>
</tbody>
</table>

Figure 38. Comparison of data-mining with and without fusion.

At this stage, the experts evaluated the complete schedule of set-points, not just the rules used in its formation. Data mining would find this level of overview very difficult. Conversely, experts would find analysing the complex interaction of over 200 rules very difficult. Experts proved intelligent in relating global phenomenon (the quality of the set-points schedule produced) to local causes (individual rules). The data-mining techniques used here were not capable of this level of abstraction.

### 3.4.4 Schedules of set points

Rules could be used to implement a basic mathematical schedule generation algorithm that considers a set draft reduction pattern, see Figure 39. However, this does not include the rules generated to avoid shape problems, which have a significant effect, see Figure 40. Important changes to the schedule include a more even pass reduction throughout the schedule, most reduction during initial passes where shape is unlikely to occur and a smaller reduction during the last shape critical passes. The generated schedule for a representative low gauge plate is shown in Table 3.
3.5 First online trial

Six trial slabs have been planned to incrementally test the practicality, performance and potential of this scheduling method. The first trial had to be prudent, with many offline tests and an online ghost slab (no metal present in the mill) performed to ensure safety of the mill and personnel. The specification was made to ensure that shape would be present if the schedule was incorrect, but the risk to plant was minimised. [Due to quality regulations, any plate designated as a trial is not sold to a customer].

3.5.1 Aims and objectives

The aim of the first trial was to test if the expert system approach is practical (and therefore potentially beneficial) for supervisory control of aluminium plate rolling. That is, can knowledge from people, plant and data be combined to produce schedules of set points for timely, safe and successful rolling?

The objectives were:

- To test the hardware linkage to plant.
• To verify and validate the software links to the supervisory control system.
• To test communication over the links:
  both downloading of plate specifications and
  uploading of schedules generated.
• To confirm ease of switching to (and from) expert system operation.
• To ensure timely operation.
• To test if schedules could be generated for different plates and conditions.
• To evaluate the practicality of online rolling with expert system schedules.
• To check if trial data had been recorded automatically.
• The following were not tested:
  • The ability to regenerate the schedule part way through rolling
    (triggered either by operating command or through adaptation).
  • Adaptation to plant measurements and gauge checks.
  • The accuracy of plant parameter prediction models.
  • Gauge performance (although gauge checks were successfully generated and
    taken, the results were not fed back to the expert system).
  • Quality performance - although this is central to the aims of the expert system,
    the trial would not have been considered a failure if poor quality product was
    rolled; it would have failed if no product had been obtained due to a system
    failure.

In fact, product was obtained. Therefore, the quality of the product was assessed. The
reason for excluding these parts was to test the core technology in isolation, before
building the additional, but necessary functionality. Even when
adaptation/regeneration become available, it may still be necessary to test without
them employed to investigate the accuracy of any improvements to plant prediction
models.

3.5.2 Method

Setting-up the expert system for schedule generation, switching the operation of the
level II control system to the expert system and the procedure to link each system are
detailed in the "Introduction to Expert System" documentation (Internal report). This
section only details the specific actions that were performed for the test.

Prior to the trial, the mill had been down for 12 hours for maintenance, which resulted
in cold work rolls. The mill was calibrated before rolling to cutback, which resulted in
a gauge check when the HMAS system rolled from slab to cutback [this operation is
currently outside the scope of the expert system]. The adaptation component of HMAS
was removed from the level II control system by setting the adapt.dll to return -1. This prevented regeneration from being triggered, which would have attempted to change the schedule initially generated by the expert system. The details of the three specified slabs are shown in Table 4.

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>InGauge (mm)</th>
<th>InLength (mm)</th>
<th>InWidth (mm)</th>
<th>FGMax (mm)</th>
<th>FGMin (mm)</th>
<th>Flength (mm)</th>
<th>Fwidth (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UL1-1</td>
<td>80.0</td>
<td>2778</td>
<td>1588</td>
<td>15.8</td>
<td>14.2</td>
<td>13638</td>
<td>1588</td>
</tr>
<tr>
<td>2</td>
<td>UL1-2</td>
<td>80.0</td>
<td>2776</td>
<td>1587</td>
<td>21.0</td>
<td>19.0</td>
<td>10600</td>
<td>1587</td>
</tr>
<tr>
<td>3</td>
<td>UL1-3</td>
<td>80.0</td>
<td>2776</td>
<td>1587</td>
<td>26.0</td>
<td>24.0</td>
<td>8480</td>
<td>1587</td>
</tr>
</tbody>
</table>

Table 4. Initial specification of trial plates

Where the abbreviations used in this report are:

*InGauge, InLength & InWidth* are the initial dimensions of the cutback. *FGMax* and *FGMin* are the tolerance band of the finishing gauge. *Flength* and *Fwidth* are the final dimensions of the plate. *ARLoad* is the anticipated rolling load. *ExGauge, ExLength, ExWidth* and *ExTemp* are the calculated pass exit dimensions and temperature. *MotorCurr* and *Millispeed* are the motor current and mill speed. *RGSet* is the role gap setting sent to the level I systems. *GCheck* is the requested gauge check (showing obtained value).

Initially, it was intended to progressively decrease the gauge from 25 mm to 20 mm to 15 mm to gradually increase the likelihood of shape occurring. However, stacking the plates in this order could have been unstable, so the rolling order was changed to 15, 20 and then 25 mm finishing gauge.

### 3.5.3 Results

Before the trial started, there were two points of note:

1. The knowledge base is not yet fully tuned:

Schedules can be generated using generic rules, with specific ones (e.g. tailored to MG4 alloy) used to fine-tune the output. Not only does the knowledge base require expansion to include more alloys, products and mill conditions, but also the overall interaction of individual rules to produce the final schedule needs perfecting.

A demonstration of this occurred as the initial schedules generated for each plate contained over 20 passes, which was judged to be too many by operators and rolling experts. The transparency of the generation showed that the hardness rules were too conservative in draft reduction. The flexibility of the system allowed these rules to be quickly adjusted to give schedules that met with approval, having around eight passes.
2. Quality prediction is integral to the schedule generation:

Queries were made regarding the predicted shape of the rolled plates. After a schedule is generated, the shape quality is checked and, if necessary, the schedule adjusted until the predicted shape is flat. In the rare occasions where this is not possible, the system attempts to obtain minimised wavy edge (easier to remove during further processing than wavy middle) and informs the operator that this is the case, together with any remedial action possible (such as delaying rolling to cool the mill). No quality problems were predicted prior to rolling.

The trial began at the cutback to plate operation, where shape is critical, see Figure 41 (initial slab) and Figure 42 (cutback).

The generated schedule for the lowest gauge plate is shown in Table 5. See Appendix A for the complete set of results obtained during the trial.

<table>
<thead>
<tr>
<th>NP</th>
<th>PassType</th>
<th>ARLoad</th>
<th>ExGauge</th>
<th>ExLength</th>
<th>ExTemp</th>
<th>ExWidth</th>
<th>MotorCurr</th>
<th>MillSpeed</th>
<th>RGSet</th>
<th>GCheck</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1079.664</td>
<td>0.067690</td>
<td>3.251</td>
<td>454.56</td>
<td>1.604</td>
<td>921.421</td>
<td>0.5</td>
<td>0.066958</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1132.997</td>
<td>0.057198</td>
<td>3.847</td>
<td>444.64</td>
<td>1.604</td>
<td>2250.211</td>
<td>1.3</td>
<td>0.056355</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1296.599</td>
<td>0.047506</td>
<td>4.632</td>
<td>440.541</td>
<td>1.604</td>
<td>2409.415</td>
<td>1.3</td>
<td>0.046325</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1354.852</td>
<td>0.039554</td>
<td>5.563</td>
<td>435.78</td>
<td>1.604</td>
<td>2231.802</td>
<td>1.3</td>
<td>0.038253</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1304.241</td>
<td>0.032877</td>
<td>6.693</td>
<td>429.599</td>
<td>1.604</td>
<td>1950.506</td>
<td>1.3</td>
<td>0.03168</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1287.706</td>
<td>0.027182</td>
<td>8.095</td>
<td>422.083</td>
<td>1.604</td>
<td>1791.221</td>
<td>1.3</td>
<td>0.026019</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1299.902</td>
<td>0.022275</td>
<td>9.878</td>
<td>413.221</td>
<td>1.604</td>
<td>1728.795</td>
<td>1.3</td>
<td>0.021087</td>
<td>0.02147</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1340.151</td>
<td>0.018188</td>
<td>12.098</td>
<td>402.925</td>
<td>1.604</td>
<td>1218.127</td>
<td>0.917</td>
<td>0.016917</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1312.926</td>
<td>0.015000</td>
<td>14.669</td>
<td>385.322</td>
<td>1.604</td>
<td>1158.876</td>
<td>0.917</td>
<td>0.013785</td>
<td>0.01475</td>
</tr>
</tbody>
</table>

Table 5. Trial UL1-1, Schedule generated for rolling the first plate

The hardware and software links to the existing supervisory control system functioned correctly, with no errors in communication. Switching between standard and expert system operation was straightforward, with the change being made between operations if necessary. Each schedule was generated in a timely manner, i.e. too long a generation time would have resulted in an error from HMAS. Different schedules were generated for the differing plates and initial conditions.

Data from the expert system, which contained the generated schedules that included the predicted plant properties, was successfully stored. Similarly, data from HMAS was also obtained, but plant data for the shop floor reporting system (PROCON) was not successfully recorded. Although there is no direct connection between PROCON and the expert system, it is to be discovered whether the lack of data stored was due to an artefact of using the expert system. Consequently, predicted to actual values, such as anticipated rolling load to rolling load, cannot be compared.
The presented results could stop here as the trial is complete. However, additional interesting data was recorded, which is useful in assessing the expert system performance and how the system should be developed in the future. Gauge and flatness performance will be evaluated:

The gauge performance is critical in producing saleable plates. To convert the pass schedule into roll gap settings, predictions of mill parameters are needed. The correct conversion may be achieved in three ways:

1. Accurate model predictions.
2. Precise adjustments to model predictions based on a mill feedback.
3. Combining good model predictions with good adjustment based on mill feedback.
The expert system used basic models for prediction (although based on HMAS models, simplifying assumptions had been made), without feedback from the mill. Thus, making it highly unlikely that gauge tolerance would be achieved [NB gauge performance was never a part of the first trial]. This is confirmed by the results in Table 6. The gauge error increased as the trial progressed; indicating that roll expansion due to temperature was not fully taken into account by the basic models. However, the errors were in the range that could be compensated by mill feedback when it is included.

<table>
<thead>
<tr>
<th>Plate</th>
<th>Entry Temp</th>
<th>Intermediate</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Aim (mm)</td>
<td>RGS (mm)</td>
</tr>
<tr>
<td>UL1-1</td>
<td>465</td>
<td>22.275</td>
<td>21.087</td>
</tr>
<tr>
<td>UL1-2</td>
<td>439</td>
<td>29.889</td>
<td>28.468</td>
</tr>
<tr>
<td>UL1-3</td>
<td>416</td>
<td>35.905</td>
<td>34.396</td>
</tr>
</tbody>
</table>

Table 6. Gauge performance, intermediate and final gauge checks

The shape performance is critical when deciding to adopt the expert system approach. Figure 43, Figure 44 and Figure 45, showed that each of the three plates rolled were flat. Visual inspection by operators, engineers and scheduling experts were extremely positive for shape performance.

The final result is the commentary from the operator (Mr Gerry Gebb) who rolled the trial. Firstly, he suggested that the mill speed for magnesium alloys should not be too high. This will be formed into a rule of the form "if the alloy-code of the current-plate is Mg and the mill-speed of the set-point > 0.5 then conclude that the mill-speed of set-point = 0.5". Lastly, he considered that he could still produce a schedule that was superior to the current system or the expert system!
3.5.4 Discussion

The specified trial plates were conservative to protect the mill, but in the zone where shape would have occurred if the expert system operation had been poor.

It is worth remarking that this software-based project had no bugs, crashes or runtime errors during the length of its first online rolling trial.

The aim of the trial, to show that using expert systems for supervisory control was practical, was completed successfully.

All data from the expert system was recorded correctly, but the shop floor reporting system did not function correctly, which requires investigating in the unlikely event that it was due to the expert system producing the schedule.
Gauge performance was not part of the trial, but it is interesting to discuss. Given the limitations imposed on the expert system (basic models and no mill feedback), it was not surprising that gauge tolerance was not met. Fortunately, the gauge error could be corrected through mill feedback once the results of gauge checks have been included in the expert system. However, making too great an adjustment after the intermediate gauge check would not be recommended, as the pass pattern would be disrupted, hence potentially reducing shape performance. Therefore, both models and feedback need to be improved, to allow the expert pass schedules to provide shape performance. Further, future trials must consider profile performance where it is critical for customer acceptance.

Shape performance could not have been improved, as all three plates were judged flat.

The comments from the operator show that it is not treated as a threat, but as a tool to assist in the production of quality plate. They also show that there are more improvements that can be made to the system.

3.5.5 Results

Measuring the benefit of the Expert System is not straightforward as many factors are attempting to improve plant performance. Once knowledge has been described, set down or learnt, it becomes a commodity that everyone can, and should, use for improved plant performance. Assigning where the improvement originates becomes impossible. Also, comparing the shape of test pieces rolled with and without the Expert System will not determine conclusively the best system for schedule generation, as no two pieces are exactly the same. However, comparison of predicted with actual shape for a given piece will quantify the accuracy of shape prediction. Therefore, the accuracy of the shape prediction can be used to compare the existing system with the new Expert System. Tracking yield, throughput, availability, operator’s satisfaction and profitability over the course of implementing and using the Expert System will give a quantitative indication of benefit.

The main conclusion drawn from the first trial is that the developed system is capable of obtaining flat plate. Successes included stable linking to the existing level II system, downloading the correct plate specification, generation of comprehensive schedules of set-points for challenging product specifications, returning to standard operation and producing flat plate for three different gauges (15, 20 and 25 mm). Apart from not recording some of the trial data, which will normally be handled outside the expert system, all aims and objectives for the trial were met.
Issues exist about expanding the product range covered by the knowledge base, obtaining further information from plant and ensuring gauge and profile tolerance, but these issues were outside the scope of this first trial.

This gives encouragement for the further development and expansion of the expert system, in order to increase the range and difficulty of the product scope; whilst satisfying customer tolerances on gauge and profile. This low-cost, flexible scheduling system offers many potential benefits in the production of quality aluminium plates.

Further work is required to make it robust over the necessary wide range of material, product and mill conditions. Improvements to the predictive models and incorporation of mill feedback are expected to provide gauge performance.

Joint implementation has ensured acceptance on plant, with personnel willing to continue to improve the system as it is considered a tool to assist operators in obtaining quality product.

Since my involvement with this expert system work ended there have been 2 further successful on-line trials.

3.6 Summary

In this chapter, the background and various important characteristics of the expert system are summarised. The concept of knowledge representation is discussed. The expert system shell G2 is explained. The overall architecture of the system and the design of the expert system are described. The methods used to build the knowledge base of the expert system are explained and finally the results from the first online trial of the developed system are presented.

It is concluded that the expert system methodology is practicable and can produce flat product. The system is stable, timely, transparent and flexible. It offers major benefits in the ease of leveraging knowledge contained within data, plant and human resources, to complement existing model-based systems.
4 Artificial Neural Networks

The aim of this chapter is to introduce some important concepts about Artificial Neural Networks. Various characteristics of ANNs and procedures to develop successful models are summarised. Attention is focused on concepts that are used in building SPO ANN models. This information is later used to describe the SPO ANN models in the next chapter.

The chapter starts with the definition and history of the ANNs. Then types of ANNs and terminology used are explained. Different network topologies, learning mechanisms, preprocessing of data and regularization of network parameters are explained.

4.1 Definition

There is no universally accepted definition of an Artificial Neural Network (ANN). But perhaps most people in the field would agree that an ANN is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information.

The elementary building block of biological neural networks is the neuron (Figure 46) that uses biochemical reactions to receive, process and transmit information. The brain is a collection of about 10 billion interconnected neurons.

The single cell neuron consists of the cell body, or soma, the dendrites, and the axon. The dendrites receive signals from the axons of other neurons. The small space between the axon of one neuron and the dendrite of another is the synapse. The dendrites conduct impulses toward the soma. The axon conducts impulses away from the soma. A neuron's dendritic tree is connected to thousands of neighbouring neurons. When one of those neurons fires, a positive or negative charge is received by one of the dendrites. The strengths of all the received charges are added together through the processes of spatial and temporal summation. Spatial summation is conversion of several weak signals into a single large one, and temporal summation is
conversion of a rapid series of weak pulses from one source into one large signal. The aggregate input is then passed to the soma. If the aggregate input is greater than the axon hillock's threshold value, then the neuron fires, and an output signal is transmitted down the axon. When the action potential reaches a synapse at the end of the axon, the electrical signal is converted to a chemical signal to be communicated across the synaptic gap to the post synaptic neuron. At the membrane of the post synaptic neuron, the chemical signal is converted back to an electrical signal to be conveyed along the dendrite to the soma. The physical and neurochemical characteristics of each synapse determine the strength and polarity of the new input signal. Learning is believed to occur by changing the effectiveness of the synapses so that the influence of one neuron on another changes. One long standing theory, called Hebbian learning states that the strength of a synapse between two neurons increases when both neurons fire simultaneously [86]. A more recent theory, put forth by Alkon is based upon conditioned and un-conditioned stimuli received on neighbouring synapses [87].

Many tasks, which seem simple for us, such as reading a handwritten note or recognizing a face, are difficult for even the most advanced computer. In an effort to increase the computer's ability to perform such tasks, programmers began designing software that simulates the features of neurons and synaptic connections. Thus the field of artificial neural networks was born.

Artificial Neural networks are mathematical systems that are comprised of a number of processing units that are linked via weighted interconnections. A processing unit is essentially an equation that is often referred to as a transfer function. A processing unit takes weighted signals from other neurons, combines them, transforms them and outputs a numeric result. Processing units are considered crudely analogous to real neurons and they are linked together in a network. These networks have the capacity to learn, memorize and create relationships amongst data. ANNs are characterized by

- Local processing in artificial neurons
- Massively parallel processing, implemented by rich connection patterns between the artificial neurons
- The ability to acquire knowledge via learning from experience
- Knowledge storage in distributed memory, the synaptic connections between artificial neurons

ANNs, like people, learn by example. Learning involves adjustments of the weights of connections between the artificial neurons. These connection weights store the knowledge necessary to solve specific problems. The behaviour of neural networks, how they map input data to output data, is influenced by the transfer functions of
neurons, how the neurons are interconnected and the weights of those interconnections. Typically, architecture of neural network is established and one of a variety of mathematical algorithms is used to determine what the weights of the interconnections should be to maximize the accuracy of the outputs produced. If there is any overall pattern to the data, or some consistent relationship between the inputs and result of each record, the network should be able to eventually create an internal mapping of weights that can accurately reproduce the expected output. Once these relationships are established, the neural network can be presented with new input variables and it will generate predictions.

Figure 47 presents an artificial neuron and Figure 48 presents a feedforward artificial neural network.

When talking about an artificial neural network (ANN), it is customary to be lazy and drop the "A" or the "artificial". There are many different ways in which people refer to ANNs. Artificial neural networks are described as connectionist systems, because of the connections between individual processing nodes. They are sometimes called adaptive systems, because the values of these connections can change so that the
neural network performs more effectively. They are also sometimes called parallel distributed processing systems, which emphasize the way in which the many nodes or neurons in a neural network operate in parallel.

A trained neural network can be thought of as an expert in the category of information it has been given to analyse. This expert can then be used to provide projections given new situations of interest and answer what if questions. Although the programming and mathematics behind neural network technologies are complex, using neural network software can be quite simple and the results are often quite extraordinary. ANNs have been used in various fields and the results to this date have revealed robust and elegant solutions. The theory that inspires neural network systems is drawn from many disciplines; primarily from neuroscience, engineering, and computer science, but also from psychology, mathematics, physics, and linguistics.

4.2 History

McCulloch and Pitts first introduced the idea of an artificial neuron for processing data in 1943 [88]. They developed models of neural networks based on their understanding of neurology. These models made several assumptions about how neurons worked. Their networks were based on simple neurons that were considered to be binary devices with fixed thresholds. The results of their model were simple logic functions such as "a or b" and "a and b". Rosenblatt stirred considerable interest and activity in the field when he designed and developed the Perceptron [89]. The Perceptron had three layers with the middle layer known as the association layer. This system could learn to connect or associate a given input to a random output unit. Another system was the Adaline (ADAptive LInear Element), which was developed in 1960 by Widrow and Hoff [90]. Adaline was an analogue electronic device made from simple components. The method used for learning was different to that of the Perceptron, it employed the Least-Mean-Squares (LMS) learning rule.

Following the initial period of enthusiasm, the ANN field survived a period of frustration and disrepute. In 1969 Minsky and Papert wrote a book in which they generalised the limitations of single layer Perceptrons to multilayered systems [91]. Their conclusions supported the disenchantment of researchers in the field. As a result, considerable prejudice against this field was activated. In the 1970s the continuous interest in neural networks dampened. Although public interest and available funding were minimal, several researchers continued working to develop neuromorphically based computational methods. During this period several paradigms were generated which modern work continues to enhance. Grossberg and Carpenter's influence founded a school of thought which explores resonating algorithms. They developed the ART (Adaptive Resonance Theory) networks based on biologically
plausible models. Anderson and Kohonen developed associative techniques independent of each other. Klopf in 1972 developed a basis for learning in artificial neurons based on a biological principle for neuronal learning called heterostasis [92].

Werbos developed and used the backpropagation learning method, however several years passed before this approach was popularised [93]. Backpropagation networks are probably the most well known and widely applied of the neural networks today. In essence, the back-propagation network is a Perceptron with multiple layers, a different threshold function in the artificial neuron, and a more robust and capable learning rule.

Amari was involved with theoretical developments: he published a paper that established a mathematical theory for a learning basis (error-correction method) dealing with adaptive pattern classification [94] while Fukushima developed a stepwise trained multilayered neural network for interpretation of handwritten characters [95].

Progress during the late 1970s and early 1980s was important to the reassurance of interest in neural networks. Perhaps more than any other publications, the paper by Hopfield in 1982 and the two-volume book by Rumelhart and McLelland in 1986 were the most influential publications responsible for this reassurance [96, 97]. Neural Networks have certainly come a long way from the early days of McCulloch and Pitts.

4.3 ANN Types and Terminology

There are many different types of ANN depending on the network topology, the learning algorithm used, the kinds of data they accept, etc. Nobody knows exactly how many. New ones or at least variations of old ones are invented every week. In this section, a collection of some of the most well known ANN types is described, the methods used in Soaking Pit Optimisation are explained in detail and the terminology used is defined. Appendix B presents a list of the well known ANN types and references where more information on them can be found.

4.3.1 Network Architecture

Network layers

The commonest type of artificial neural network consists of three layers of units: a layer of input units is connected to a layer of hidden units, which is connected to a layer of output units.

The activity of the input units represents the raw information that is fed into the network. The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units. The behaviour of the output units depends on the activity of the hidden units and the
weights between the hidden and output units. This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents. In a single-layer organisation, all units are connected to one another whereas multi-layer organisations are structured hierarchically. In multi-layer networks, instead of following a global numbering, units are often numbered by layer.

Two major kinds of network topology are feed-forward and feedback.

**Feed-forward networks**

Feed-forward ANNs allow signals to travel one way only; from input to output. There are no feedback loops. The output of any layer does not affect that same layer. Feed-forward ANNs tend to be straightforward networks that associate inputs with outputs. This type of organisation is also referred to as bottom-up or top-down. Classical examples of feed-forward networks are the Perceptron and Adaline.

**Feedback networks**

Feedback networks can have signals travelling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their state is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes, then a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organisations.

**Multilayer feed-forward networks**

This is perhaps the most popular network architecture in use today, due originally to Rumelhart and McClelland [97] and discussed at length in most neural network textbooks, e.g. [84]. In this structure, each unit performs a biased weighted sum of their inputs and passes this activation level through a transfer function to produce their output, and the units are arranged in a layered feedforward topology. The network thus has a simple interpretation as a form of input-output model, with the weights and biases, the free parameters of the model. Such networks can model functions of almost arbitrary complexity, with the number of layers, and the number of units in each layer, determining the function complexity. Important issues in design include specification of the number of hidden layers and the number of units in these layers [98, 84]. The number of input and output units is defined by the problem. The number of hidden layers and units to use is not clear. A good a starting point is to use one hidden layer, with the number of units equal to half the sum of the number of input and output units.
4.3.2 Transfer Function

The behaviour of an ANN depends on both the weights and the input-output transfer function that is specified for the units. This function typically falls into one of three categories: linear (or ramp), threshold, and sigmoid. For linear units, the output activity is proportional to the total weighted output. For threshold units, the output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value. For sigmoid units, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurones than do linear or threshold units, but all three must be considered as rough approximations.

4.3.3 The Kinds of Data ANNs accept

ANNs also differ in the kinds of data they accept. Two major kinds of data are categorical and quantitative.

Categorical variables take only a finite number of possible values, and there are usually several or more cases falling into each category. Categorical variables may have symbolic values (e.g., "male" and "female", or "red", "green" and "blue") that must be encoded into numbers before being given to the network. Both supervised learning with categorical target values and unsupervised learning with categorical outputs are called classification.

Quantitative variables are numerical measurements of some attribute, such as length in meters. The measurements must be made in such a way that at least some arithmetic relations among the measurements reflect analogous relations among the attributes of the objects that are measured.

Some variables can be treated as either categorical or quantitative, such as number of children or any binary variable.

4.3.4 The Learning Process

The memorisation of patterns and the subsequent response of the network can be categorised into two general paradigms:

Associative mapping

The network learns to produce a particular pattern on the set of input units whenever another particular pattern is applied on the set of input units. The associative mapping can generally be broken down into two mechanisms. If an input pattern is associated with itself and the states of input and output units coincide it is called auto-association. Auto-association is used to provide pattern completion, i.e. to produce a pattern whenever a portion of it or a distorted pattern is presented. If the network
actually stores pairs of patterns building an association between two sets of patterns, it is called hetero-association. *Hetero-association* is related to two recall mechanisms:

- **nearest-neighbour recall**, where the output pattern produced corresponds to the input pattern stored, which is closest to the pattern presented, and
- **interpolative recall**, where the output pattern is a similarity dependent interpolation of the patterns stored corresponding to the pattern presented.

Yet another paradigm, which is a variant associative mapping is classification, i.e. when there is a fixed set of categories into which the input patterns are to be classified.

**Regularity detection**

Units learn to respond to particular properties of the input patterns. In regularity detection the response of each unit has a particular meaning, whereas in associative mapping the network stores the relationships among patterns. This type of learning mechanism is essential for feature discovery and knowledge representation.

ANNs possess knowledge by the values of the connection weights. Modifying the knowledge stored in the network as a function of experience implies a learning rule for changing the values of the weights. Learning is the determination of the weights. Following the way learning is performed we can distinguish two major categories of neural networks:

- **Fixed networks** in which the weights cannot be changed. The weights are fixed a priori according to the problem to solve.
- **Adaptive networks** which are able to change their weights.

All learning methods used for adaptive neural networks can be classified into two major categories:

**Supervised learning**

Supervised learning incorporates an external teacher, so that each output unit is told what its desired response to input signals ought to be. During training, the target values are given to the ANN so that the ANN can adjust its weights to match its outputs to the target values. After training, the ANN is tested by giving it only input values, not target values. Paradigms of supervised learning include error-correction learning, reinforcement learning and stochastic learning. The backpropagation algorithm, which is used in training the SPO ANN models, is an example of supervised learning paradigm.
Unsupervised learning

Unsupervised learning uses no external teacher and is based upon only local information. It is also referred to as self-organisation, in the sense that it self-organises data presented to the network and detects their emergent collective properties. Paradigms of unsupervised learning are Hebbian learning and competitive learning. In unsupervised learning, the ANN is not provided with the correct results during training. Unsupervised ANNs usually perform some kind of data compression, such as dimensionality reduction or clustering.

The distinction between supervised and unsupervised methods is not always clear-cut. An unsupervised method can learn a summary of a probability distribution, then that summarized distribution can be used to make predictions. Supervised methods come in two kinds: auto-associative and hetero-associative. In auto-associative learning, the target values are the same as the inputs, whereas in hetero-associative learning, the targets are generally different from the inputs. Many unsupervised methods are equivalent to auto-associative supervised methods.

Another aspect of learning concerns the distinction of a separate phase, during which the network is trained, and a subsequent operation phase. A neural network learns off-line if the learning phase and the operation phase are distinct and it learns on-line if it learns and operates at the same time. Usually, supervised learning is performed off-line, whereas unsupervised learning is performed on-line.

Backpropagation Algorithm

Neural networks can be explicitly programmed to perform a task by manually creating the topology and then setting the weights of each link and threshold. However, this bypasses one of the unique strengths of neural nets: the ability to program themselves. The most basic method of training a neural network is trial and error. If the network isn't behaving the way it should, change the weighting of a random link by a random amount. If the accuracy of the network declines, undo the change and make a different one. It takes time, but the trial and error method does produce results. But the number of possible weightings rises exponentially as new neurons are added, making large general-purpose neural nets impossible to construct using trial and error methods.

The best-known example of a neural network training algorithm is backpropagation [98, 99]. The central idea behind this algorithm is that errors for the units of the hidden layer are determined by back propagating the errors of the units of the output layer. Backpropagation learning takes only a fraction of the time that trial and error method takes. A feed-forward ANN that uses the backpropagation algorithm is
referred to as a Backpropagation network. Backpropagation ANNs are excellent at prediction and classification tasks.

Although a backpropagation algorithm can be applied to networks with any number of layers, it has been shown that only one layer of hidden units suffices to approximate any function with finitely many discontinuities to arbitrary precision, provided the activation function of the hidden units are nonlinear [100]. This does not necessarily imply that a network with more layers might not more conveniently or easily model a particular problem. In practice, however, most problems seem to yield to a single hidden layer, with two an occasional resort and three practically unknown.

The backpropagation algorithm is named after the method of taking the error from the output layer, to the middle layers and through them back to the input layer. Then mathematical methods are used to adjust the weights of the connections so that the desired outcome is achieved. Below, the method used to back propagate the error is described.

It is convenient to represent the pattern of connectivity in the network by a weight matrix $W$ whose elements are the weights $w_{ij}$. $w_{ij}$ is the weight of the connection from unit $u_i$ to unit $u_j$.

A unit in the output layer determines its activity by the following two-step procedure.

First, it computes the total weighted input $x_{ji}$, using the formula:

$$X_j = \sum_i y_i W_{ij} \quad (4.1)$$

where $y_i$ is the activity level of the $i^{th}$ unit in the previous layer and $W_{ij}$ is the weight of the connection between the $i^{th}$ and the $j^{th}$ unit.

Next, the unit calculates the activity $y_j$ using some function of the total weighted input. Typically we use the sigmoid function:

$$y_j = \frac{1}{1+e^{-x_j}} \quad (4.2)$$

Once the activities of all output units have been determined, the network computes the error $E$, which is defined by the expression:

$$E = \frac{1}{2} \sum_j (y_j - d_j)^2 \quad (4.3)$$

where $y_j$ is the activity level of the $j^{th}$ unit in the top layer and $d_j$ is the desired output of the $j^{th}$ unit.

The back-propagation algorithm consists of four steps:
1. Compute how fast the error changes as the activity of an output unit is changed. This error derivative (EA) is the difference between the actual and the desired activity.

\[ EA_j = \frac{\partial E}{\partial y_j} = y_j - d_j \quad (4.4) \]

2. Compute how fast the error changes as the total input received by an output unit is changed. This quantity (EI) is the answer from step 1 multiplied by the rate at which the output of a unit changes as its total input is changed.

\[ EI_j = \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \times \frac{\partial y_j}{\partial x_j} = EA_j y_j (1 - y_j) \quad (4.5) \]

3. Compute how fast the error changes as a weight on the connection into an output unit is changed. This quantity (EW) is the answer from step 2 multiplied by the activity level of the unit from which the connection emanates.

\[ EW_{ij} = \frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{\partial W_{ij}} = EI_j y_i \quad (4.6) \]

4. Compute how fast the error changes as the activity of a unit in the previous layer is changed. This crucial step allows back propagation to be applied to multilayer networks. When the activity of a unit in the previous layer changes, it affects the activities of all the output units to which it is connected. So to compute the overall effect on the error, we add together all these separate effects on output units. Each effect is simple to calculate. It is the answer in step 2 multiplied by the weight on the connection to that output unit.

\[ EA_i = \frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{\partial y_i} = \sum_j EI_j W_{ij} \quad (4.7) \]

By using steps 2 and 4, we can convert the EAs of one layer of units into EAs for the previous layer. This procedure can be repeated to get the EAs for as many previous layers as desired. Once we know the EA of a unit, we can use steps 2 and 3 to compute the EWs on its incoming connections.

**Gradient Descent, Conjugate Gradient and Levenberg-Marquardt Methods**

Training a neural network is, in most cases, an exercise in numerical optimisation of a usually non-linear objective function that is the error function in which other quantities such as penalties for weight decay may be included.

There is no single best method for non-linear optimisation. You need to choose a method based on the characteristics of the problem to be solved. For objective functions with continuous second derivatives, which would include feedforward nets
with the most popular differentiable activation functions and error functions, three
general types of algorithms have been found to be effective for most practical
purposes:

- For a small number of weights, stabilized Newton and Gauss-Newton algorithms,
  including various Levenberg-Marquardt and trust-region algorithms, are efficient.
  The memory required by these algorithms is proportional to the square of the
  number of weights.

- For a moderate number of weights, various quasi-Newton algorithms are efficient.
  The memory required by these algorithms is proportional to the square of the
  number of weights.

- For a large number of weights, various conjugate-gradient algorithms are efficient.
  The memory required by these algorithms is proportional to the number of
  weights.

Once the number of layers, and number of units in each layer, has been selected, the
network's weights and thresholds must be set so as to minimize the prediction error
made by the network. This is the role of the training algorithms. The historical cases
that have been gathered are used to automatically adjust the weights and thresholds
in order to minimize this error. This process is equivalent to fitting the model
represented by the network to the training data available. The error of a particular
configuration of the network can be determined by running all the training cases
through the network, comparing the actual output generated with the desired or
target outputs. The differences are combined together by an error function to give the
network error. The most common error function is the sum-squared error, where the
individual errors of output units on each case are squared and summed together.

In traditional modelling approaches like linear modelling it is possible to algorithmically
determine the model configuration that absolutely minimizes this error. The price paid
for the non-linear modelling power of neural networks is that although we can adjust a
network to lower its error, we can never be sure that the error could not be lower still.

A helpful concept to understand and solve optimisation problems is the error surface.
Each of the \(N\) weights and thresholds of the network is taken to be a dimension in
space. The \((N+1)^{th}\) dimension is the network error. For any possible configuration of
weights the error can be plotted in the \((N+1)^{th}\) dimension, forming an error surface.
The objective of network training is to find the lowest point in this many-dimensional
surface. In a linear model with sum squared error function, this error surface is a
parabola, which means that it is a smooth bowl-shape with a single minimum. It is
therefore easy to locate the minimum. Neural network error surfaces are much more
complex, and are characterized by a number of unhelpful features, such as local minima, flat-spots and plateaus, saddle-points, and long narrow ravines. It is not possible to analytically determine where the global minimum of the error surface is, and so neural network training is essentially an exploration of the error surface. From an initially random configuration of weights and thresholds (i.e. a random point on the error surface), the training algorithms incrementally seek for the global minimum. Gradient descent does this by calculating the gradient of the error surface at the current point, and then using that information to make a downhill move. The gradient vector of the error surface points along the line of steepest descent from the current point, so the error decreases if we move along it a short distance. A sequence of such moves will eventually find a minimum. The difficult part is to decide how large the steps should be.

Large steps may converge more quickly, but may also overstep the solution or if the error surface is very eccentric go off in the wrong direction. In practice, the step size is proportional to the slope so that the algorithm settles down in a minimum and to a special constant: the learning rate. The correct setting for the learning rate is application-dependent, and is typically chosen by experiment; it may also be time varying, getting smaller as the algorithm progresses. The algorithm is also usually modified by inclusion of a momentum term: this encourages movement in a fixed direction, so that if several steps are taken in the same direction, the algorithm picks up speed, which gives it the ability to sometimes escape local minimum, and also to move rapidly over flat spots and plateaus.

The algorithm therefore progresses iteratively, through a number of epochs. On each epoch, the training cases are each submitted in turn to the network, and target and actual outputs compared and the error calculated. This error, together with the error surface gradient, is used to adjust the weights, and then the process repeats. The initial network configuration is random, and training stops when a given number of epochs elapses, or when the error reaches an acceptable level, or when the error stops improving.

More sophisticated second-order algorithms such as conjugate gradient descent and Levenberg-Marquardt are an order of magnitude faster than gradient descent for many problems [84, 101]. They also use a strategy designed to travel towards a minimum as quickly as possible. Conjugate gradient descent and Levenberg-Marquardt are very successful forms of two types of algorithms: line search and model-trust region approaches. [102] has a good elementary discussion of conjugate gradient and Levenberg-Marquardt algorithms in the context of ANNs. In practice,
Levenberg-Marquardt often finds better optima for a variety of problems than do the other usual methods.

A line search algorithm works as follows: pick a sensible direction to move in the multi-dimensional landscape. Then project a line in that direction, locate the minimum along that line and repeat. It is relatively trivial to locate a minimum along a line, by using some form of bisection algorithm. A sensible choice for the direction is the direction of gradient descent. But this intuitively obvious choice proves to be rather poor. Having minimized along one direction, the next line of steepest descent may "spoil" the minimization along the initial direction. Even on a simple surface like a parabola a large number of line searches may be necessary. A better approach is to select conjugate or non-interfering directions - hence conjugate gradient descent. The idea here is that, once the algorithm has minimized along a particular direction, the second derivative along that direction should be kept at zero. Conjugate directions are selected to maintain this zero second derivative on the assumption that the surface is a nice smooth surface. If this condition holds, \( N \) epochs are sufficient to reach a minimum. In reality, on a complex error surface the conjugacy deteriorates, but the algorithm still typically requires far less epochs than back propagation, and also converges to a better minimum.

A model-trust region approach works as follows: instead of following a search direction, assume the surface is a simple shape such that the minimum can be located and jumped to directly which will be true if sufficiently close to a minimum. Elsewhere, the assumption may be grossly violated, and the model could choose inappropriate points to move to. The model can only be trusted within a region of the current point, and the size of this region is not known. Therefore, choose new points to test as a compromise between that suggested by the model and that suggested by a standard gradient-descent jump. If the new point is good, move to it, and strengthen the role of the model in selecting a new point; if it is bad, don't move, and strengthen the role of the gradient descent step in selecting a new point and make the step smaller.

Levenberg-Marquardt uses a model which assumes that the underlying function is locally linear and therefore has a parabolic error surface. Levenberg-Marquardt is typically the fastest of the training algorithms [103], although unfortunately it has some important limitations. Specifically: it can only be used on single output networks, can only be used with the sum squared error function, and has memory requirements proportional to the square of the number of weights in the network; this makes it impractical for reasonably big networks. Conjugate gradient descent is nearly as good, and doesn't suffer from these restrictions.
Gradient Descent can still be useful, not least in providing a quick (if not overwhelmingly accurate) solution. It is also a good choice if the data set is very large, and contains a great deal of redundant data. Gradient Descent's case-by-case error adjustment means that data redundancy does it no harm (for example, if you double the data set size by replicating every case, each epoch will take twice as long, but have the same effect as two of the old epochs, so there is no loss). In contrast, Levenberg-Marquardt and Conjugate Gradient Descent both perform calculations using the entire data set, so increasing the number of cases may significantly slow each epoch, but does not necessarily improve performance on that epoch (not if data is redundant; if data is sparse, then adding data will make each epoch better). Gradient Descent may also be equally good if the data set is very small, for there is then insufficient information to make a highly fine-tuned solution appropriate. A more advanced algorithm may achieve a lower training error, but the verification error is unlikely to improve in the same way.

4.3.5 Data Gathering

The training data set includes a number of cases, each containing values for a range of input and output variables. The first decisions that have to be made are: which variables to use, and how many and which cases to gather.

Selection of Input Variables

The choice of variables is mainly guided by intuition. Experience in the problem domain would give some idea of which input variables are likely to be influential. At first, all the variables that are thought to have an influence should be included. Part of the design process is to whittle this set down. The selection of a good set of inputs is complicated by a number of important considerations.

Each additional input unit in a network adds another dimension to the space in which the data cases reside. Thought of in this way, there must be sufficient data points to populate an \( N \) dimensional space sufficiently densely to be able to map the relation between inputs and outputs. The number of points needed to do this properly grows very rapidly with the dimensionality, roughly in proportion to \( 2^N \) for most modelling techniques. BP ANNs can concentrate on a lower-dimensional section of the high-dimensional space by setting the outgoing weights from a particular input to zero, and can entirely ignore that input. Nevertheless, the performance of a network can certainly be improved by eliminating unnecessary input variables. Indeed, even input variables that carry a small amount of information may sometimes be better eliminated if this reduces the effects of high dimensionality. It would be extremely useful if each input variable candidate could be independently assessed for usefulness. Unfortunately, it is not always possible to do this. Two or more inter-dependent
variables may together carry significant information that a subset would not. Although they may carry no useful information alone, together they can be carrying important information. A number of variables may carry to some extent or other the same information. It may be sufficient to use as inputs a subset of such correlated variables, and the choice of subset may be arbitrary. The superiority of using a subset of correlated variables over the full set is the consequence of reduction in dimensions.

Selection of input variables is a critical part of neural network design. In addition to the expert knowledge of the problem domain, standard statistical tests can be used to make the selection of variables. Probabilistic and generalized regression networks can be extremely useful in doing these tests. PNNs and GRNNs suffer from poor inputs. A radially-based network such as a PNN or GRNN cannot ignore spurious inputs like MLPs which can set weights of the spurious data to 0. Clusters in the relevant lower-dimensional space get smeared out through the irrelevant dimension, requiring larger numbers of units to encompass the irrelevant variability. This weakness is an advantage when trying to eliminate such inputs.

Another approach in dealing with dimensionality problems, which may be an alternative or a complement to variable selection, is dimensionality reduction which can be done by processing the original set of variables to produce a smaller set of variables which contain as much information as possible from the original set. The most common approach is principal components analysis [84]. This is a linear transformation, which locates directions of maximum variance in the original input data, and rotates the data along these axes. Typically, the first principal components contain most information. PCA can often extract a very small number of components from quite high-dimensional original data and still retain the important structure. One problem with PCA is that it is a linear technique, and thus may miss important structure. A kind of non-linear PCA can be performed using an autoassociative network [84]. This is a neural network, which is trained to reproduce its inputs at its outputs, and has a middle layer with less neurons than the input and output layers. Thus, to reproduce the inputs the network must learn a lower-dimensional representation, squeezing the cases into this representation in the middle layer, before reinterpreting it to form the outputs. Once trained, the front-end of the autoassociative network can be extracted to perform dimensionality reduction. An autoassociative network is typically an MLP with three hidden layers, the middle of which holds the reduced-dimensionality representation. The other two hidden layers are necessary to allow the network to perform a non-linear transformation from inputs to middle layer, and middle layer to output layer, respectively. An autoassociative network with only one hidden layer can only do linear dimensionality reduction, and actually learns to approximate standard PCA.
Data Selection

The number of cases required for neural network training depends on the size of the network and the unknown complexity of the underlying function that the network is trying to model. As the number of variables increases, the number of cases required increases non-linearly. With even a fairly small number of variables, a huge number of cases are required. For most practical problem domains, the number of cases required will be hundreds or thousands. For very complex problems, even more may be required. If the data is sparse, it would be better to fit a linear model rather than using ANNs.

Some variables may be corrupted by noise, or values may be missing altogether. The missing values can be patched using the mean variable value, or other statistics. If there is a shortage of data, cases with missing values can be included, although obviously this is not ideal and should be avoided if not necessary. Neural networks are noise tolerant. However, there is a limit to this tolerance. If there are occasional outliers far outside the range of normal values for a variable, they may bias the training. The best approach to such outliers is to identify and remove them by either discarding the case, or converting the outlier into a missing value. The training, verification, and test data must be representative of the underlying model and the three sets must be independently representative. If training data is not representative, then the model is useless. Training data is typically historical. If circumstances have changed, relationships that held in the past may no longer hold. All eventualities must be covered. A neural network can only learn from cases that are present. It would not make a correct decision when it encounters one of the previously unseen cases. A network learns the easiest features it can. If all possible cases are not introduced, it will learn the most trivial relation in the available cases.

Since a network minimizes an overall error, the proportion of types of data in the set is critical. A network trained on a data set with 900 good cases and 100 bad will bias its decision towards good cases, as this allows the algorithm to lower the overall error which is much more heavily influenced by the good cases. If the representation of good and bad cases is different in the real population, the network's decisions may be wrong. In such circumstances, the data set may need to be crafted to take account of the distribution of data (the less numerous cases can be replicated, or remove some of the numerous cases), or the network's decisions modified by the inclusion of a loss matrix [84]. Often, the best approach is to ensure even representation of different cases, then to interpret the network's decisions accordingly.

Pre- and Post-processing

All neural networks take numeric input and produce numeric output. The transfer function of a unit is typically chosen so that it can accept input in any range, and
produces output in a strictly limited range. It has a squashing effect. Although the input can be in any range, there is therefore a saturation effect so that the unit is only sensitive to inputs within a fairly limited range. The most common transfer function is the sigmoid function. The output, in this case, is in the range (0,1), and the input is sensitive in a range not much larger than (-1, 1). The limited numeric response range, together with the fact that information has to be in numeric form, implies that neural solutions require pre-processing and post-processing stages [84].

Numeric values have to be scaled into a range that is appropriate for the network. Typically, raw variable values are scaled linearly. In some circumstances, non-linear scaling may be appropriate. For example, if a variable is distributed exponentially, a logarithmic scaling might be taken.

Neural networks process numeric data in a fairly limited range. This presents a problem if data is in an unusual range, if there is missing data, or if data is non-numeric. Numeric data is scaled into an appropriate range for the network, and missing values can be substituted for using the mean value or other statistic of that variable across the other available training cases [84].

The most common type of non-numeric data consists of nominal-value variables such as Gender={Male, Female}. A two-state nominal variable is easily represented by transformation into a numeric value (e.g. Male=0, Female=1). Many-state nominal variables are more difficult to handle. They can be represented using a single numeric value (e.g. Dog=0, Budgie=1, Cat=2) but this probably implies a false ordering on the nominal values. In this case, Budgies are in some sense midway between Dogs and Cats. A better approach is to use a number of numeric variables to represent the single nominal variable. The number of numeric variables equals the number of possible values; one of the N variables is set, and the others cleared (e.g. Dog={1,0,0}, Budgie={0,1,0}, Cat={0,0,1}). Unfortunately, a nominal variable with a large number of states would require a prohibitive number of numeric variables for one-of-N encoding, driving up the network size and making training difficult. In such a case it is possible to model the nominal variable using a single numeric index. A better approach is to look for a different way to represent the information. Other types of non-numeric data must either be converted to numeric form, or discarded. Dates and times, if important, can be converted to an offset value from a starting date/time. Currency values can easily be converted. Unconstrained text fields such as names cannot be handled and should be discarded.

Prediction problems may be divided into two main categories: classification and regression. In classification, the objective is to determine to which of a number of discrete classes a given input case belongs. In all these cases, the output required is
clearly a single nominal variable. The most common classification tasks are (as above) two-state, although many-state tasks are also not unknown. In regression, the objective is to predict the value of usually a continuous variable. In this case, the output required is a single numeric variable.

Neural networks can actually perform a number of regression and/or classification tasks at once, although commonly each network performs only one. In the vast majority of cases, the network will have a single output variable, although in the case of many-state classification problems, this may correspond to a number of output units. The post-processing stage takes care of the mapping from output units to output variables.

4.3.6 Generalization

Generalization refers to the ANN producing reasonable outputs for inputs not encountered during learning. The error that is really important when using ANNs is the expected error the network will make when new cases are submitted to it. In other words, the most desirable property of a network is its ability to generalize to new cases. In reality, the network is trained to minimize the error on the training set. This is not the same thing as minimizing the error on the error surface of the underlying and unknown model [84]. The most important manifestation of this distinction is the problem of over-learning, or over-fitting. The network has memorized the training examples, but it has not learned to generalize to new data. One method for improving network generalization is to use a network that is just large enough to provide an adequate fit. The larger a network you use, the more complex the functions that the network can create. If we use a small enough network, it will not have enough power to overfit the data. The problem is that it is difficult to know beforehand how large a network should be for a specific application.

A network with more weights models a more complex function, and is therefore prone to over-fitting. A network with fewer weights may not be sufficiently powerful to model the underlying function. For example, a network with no hidden layers actually models a simple linear function. To select the right complexity of network we can use cross verification. The available data is divided into three subsets for training validation and testing. Some of the training cases are reserved to keep an independent check on the progress of the learning algorithm. The training set is used for computing the gradient and updating the network weights and biases. The error on validation set is monitored during the training process. The validation error will normally decrease during the initial phase of training, as does the training set error. However, if the verification error stops dropping, or indeed starts to rise, this indicates that the network is starting to overfit the data, and training should cease. When overfitting occurs during the
training process, it is called overlearning. In this case, it is usually advisable to
decrease the number of hidden units and/or hidden layers, as the network is
overpowerful for the problem at hand. If the network is not sufficiently powerful to
model the underlying function, overlearning would not occur, but neither training nor
verification errors drop to a satisfactory level either.

To make decisions over the size of network to use during the design phase involves
experimenting with a large number of different networks, probably training each one a
number of times to avoid being fooled by local minimum, and observing individual
performances. The test set is used to compare the performance of different models.

Another method for improving generalization is regularization. Regularization involves
modifying the performance function, which is normally chosen to be the sum of
squares of the network errors on the training set. We may add to the performance
function a term that consists of the mean of the sum of squares of the network
weights and biases. This would cause the network to have smaller weights and biases,
and the network response to be smoother and less likely to overfit. However it is
difficult to determine the optimal ratio of this value to be added. If the ratio is too
small, we may get overfitting. If it is too large, the network will not adequately fit the
training data. It is desirable to determine the optimal regularization parameters in an
automated fashion. One approach to this process is the Bayesian framework where the
biases and weights of the network are assumed to be random variables with specified
distributions. The regularization parameters are related to the unknown variances
associated with these parameters using statistical techniques.

Automated Regularization

Conventional training methods for multilayer feedforward ANNs can be interpreted in
statistical terms as variations on maximum likelihood estimation. The idea is to find a
single set of weights for the network that maximise the fit to the training data,
perhaps modified by some sort of weight penalty to prevent overfitting.

The Bayesian school of statistics is based on a different view of what it means to learn
from data, in which probability is used to represent uncertainty about the relationship
being learned. Before we have seen any data, our prior opinions about what the true
relationship might be can be expressed in a probability distribution over the network
weights that define this relationship. After we look at the data, our revised opinions
are captured by a posterior distribution over network weights. Network weights that
seemed plausible before, but which do not match the data very well, will now be seen
as being much less likely, while the probability for values of the weights that do fit the
data well will have increased.
The purpose of training is to make predictions for future cases in which only the inputs to the network are known. The result of conventional network training is a single set of weights that can be used to make such predictions. In contrast, the result of Bayesian training is a posterior distribution over network weights. If the inputs of the network are set to the values for some new case, the posterior distribution over network weights will give rise to a distribution over the outputs of the network, which is known as the predictive distribution for this new case. If a single-valued prediction is needed, one might use the mean of the predictive distribution, but the full predictive distribution also tells you how uncertain this prediction is.

It is desirable to determine the optimal regularization parameters of a network in an automated fashion. One approach to this is the Bayesian framework of David MacKay [104]. In this framework the weights and biases of the network are assumed to be random variables with specified distributions. The regularization parameters are related to the unknown variances associated with these distributions. We can then estimate these parameters using statistical techniques. A detailed discussion of the use of Bayesian regularization, in combination with Levenberg-Marquardt training, can be found in [105].

4.4 Linear Regression Analysis

The goal of regression analysis is to determine the values of parameters for a function that cause the function to best fit a set of data observations. In linear regression, the function is a linear straight line. The dependent variable is assumed to be a linear function of independent variables plus an error introduced to account for all other factors as in the regression equation below:

$$y_i = \beta_1 x_{i1} + \cdots + \beta_k x_{ik} + \epsilon_i \quad (4.8)$$

where $y_i$ is the dependent variable, $x_{i1}, \ldots, x_{ik}$ are the independent or explanatory variables, and $\epsilon_i$ is the disturbance or error term. The goal of regression analysis is to obtain estimates of the unknown parameters $\beta_1, \ldots, \beta_k$ which indicate how a change in one of the independent variables affects the values taken by the dependent variable. If a perfect fit existed between the function and the actual data, the target value in the data would exactly equal the predicted value by the model. Typically, however, this is not the case, and the difference between the actual value of the dependent variable and its predicted value for a particular observation is the error of the estimate which is known as residual. The goal of regression analysis is to determine the values of the parameters that minimize the sum of the squared residual values for the set of observations. This is known as least squares regression fit. This
procedure is based upon the least mean of squared errors technique for fitting a
straight line to irregular data invented by Carl Gauss.

Let $b_1, \ldots, b_k$ denote the estimates of $\beta_1, \ldots, \beta_k$, then the predicted value of $\hat{y}_i$ is:

$$\hat{y}_i = b_1 x_{i1} + \cdots + b_k x_{ik} \quad (4.9)$$

The error in the prediction of $y_i$, the residual, is:

$$r_i = y_i - \hat{y}_i \quad (4.10)$$

The basic idea of a least squares regression fit is to chose estimates $b_1, \ldots, b_k$ to
minimize the sum of squared residuals:

$$\sum_{i} r_i^2 \quad (4.11)$$

The linear regression model can be expressed in matrix format as:

$$y = X\beta + \varepsilon \quad (4.12)$$

where $X$ is an $n \times k$ matrix with $(i,k)^{th}$ element $x_{ik}$, $y$ is an $n \times 1$ vector with typical
element $y_i$, and $b$ is a $k \times 1$ vector with typical element $b_k$. It can be shown that:

$$b = (X'X)^{-1}X'y \quad (4.13)$$

If $b$ is plugged back into the model formula, the predicted $y$ values at the data points
would be:

$$\hat{y} = Xb = Hy \quad (4.14)$$

where $H = X(X'X)^{-1}X'$

then residuals would be:

$$r = y - \hat{y} = (I-H)y \quad (4.15)$$

The residuals are useful for detecting failures in the model assumptions, since they
correspond to the errors, $\varepsilon$, in the model equation. By assumption, these errors each
have independent normal distributions with mean zero and a constant variance. The
residuals, however, are correlated and have variances that depend on the locations of
the data points. It is common practice to scale (studentize) the residuals so they all
have the same variance.

The manner in which Neural Networks approximate functions can be thought of a
generalization of statistical regression analysis. It has been shown that a perceptron is
equivalent to a regression model, and that an MLP with one hidden unit carries out
logistical regression. Neural network classifiers have also been shown to be equivalent
to Bayesian models [106]. A primary advantage of NN over classical statistical
regression analysis is that the neural networks have more general functional forms
than the well developed statistical methods can effectively deal with. Neural Networks are free from dependency on linear superposition and orthogonal functions which linear statistical regression approaches must be because without them the known mathematics upon which linear regression is based does not go through. In linear regression analysis the fitting functions can be nonlinear functions of the input data, but only linear functions of the parameters. In nonlinear statistical regression the fitting function can be a nonlinear function of both the input data and the parameters. Thus, the methods of nonlinear statistical regression analysis resemble those of neurocomputing. However, few if any individual statistical regression function methods have been developed as thoroughly as the neurocomputing architectures. Enough experimental evidence has now been gathered to state with some confidence that mapping networks are, in general, comparable to the best nonlinear statistical regression approaches. The function approximations that arise from properly applied mapping networks (at least in instances where sufficient training data is available) are usually better than those provided by linear regression techniques. This difference is particularly important in high dimensional spaces (input dimensions greater than 3 to 10), where linear regression techniques often fail to produce an appropriate approximation.

4.5 Summary

In this chapter a concise review of Artificial Neural Networks is given. Information on ANN topologies, processing of data, learning algorithms, and methods for regularization of network parameters are explained. A simple introduction of linear regression technique is presented too. These concepts are used in the next chapter to explain the SPO ANN models.
5 SPO ANN Models

In this chapter, the ANN models developed to estimate the heating and rolling times of pit loads are explained. These models are referred to as "SPO ANN models". The SPO scheduling module uses the estimated durations to synthesise a schedule for the operations of the soaking pits and the rolling mill. Accurate estimations of the durations of the heating and rolling processes are necessary in order to produce useful schedules. The aim of this work is to obtain better accuracy and resolution in calculation of these processing times which are estimated manually at present.

Detailed information on the soaking pits/rolling mill process is presented in chapter 2. A review of ANNs is given in chapter 4. In order to understand this chapter better, the reader is advised to read these two earlier chapters first before continuing with the rest of this chapter.

Accurate estimation of heating and rolling durations is a challenging problem, due to: many different types of slabs to be processed, the intercorrelation among process variables, the large dimension of the input space, and nonlinear behaviour.

Traditionally, mathematical models have been developed to predict the temperature profile of a slab and consequently determine the heating time required. It is also possible to build a mathematical model to predict the rolling time of a slab. However, detailed nonlinear models tend to be computationally intensive, and are expensive to build. Moreover such models are only valid for local operating regions and in the case of a real process, not enough data are available on the physical properties and model parameters. As a result, these processes have usually been operated on the basis of experience and knowledge of the operators. A lookup table reference method has been used widely. However, since such a lookup table keeps set point values for only discreet conditions, process operators have used interpolation for other conditions. But interpolation is not always accurate and reliable enough to estimate the duration of these processes.

Neural networks have been proven by many researchers to be universal approximators of any continuous function with arbitrarily desired accuracy. They offer a flexible structure that can map arbitrary nonlinear functions, making them ideally suited for the modelling of complex, nonlinear systems. They are particularly appropriate for multivariable applications, where they can readily characterise the interactions between different inputs and outputs. A further benefit is that the neural architecture is inherently parallel, distributed and has the potential for real time implementation.
Some of the recent work on ANN applications in the metal industry is reported by [107, 108, 109, 110, 111, 112, 113]. However, to the best of our knowledge no research on using ANNs for estimating process durations is reported yet.

To predict the heating and rolling times of pit loads at Alcoa Europe's Kitts Green Plant, two separate ANN models are used. In this chapter, the specifications of these SPO ANNs are presented.

Artificial neural network design follows a number of stages:

1. Collect and filter data.
2. Select the input variables.
3. Select the model family.
4. Select the structural parameters of the model in the family (which is equivalent to finding the structure and size of the neural network and the number of nodes in the hidden layer). In general, these dimensions would be chosen by trial-and-error.
   4.1. Select an initial configuration. Typically, one hidden layer with the number of hidden units set to half the sum of the number of input and output units.
   4.2. Iteratively, conduct a number of experiments with each configuration, retaining the best network found in terms of verification error. A number of experiments are required with each configuration to avoid being fooled if training locates a local minimum.
      On each experiment, if under-learning occurs, the network does not achieve an acceptable performance level, try adding more neurons to the hidden layer(s). If this does not help, try adding an extra hidden layer.
   4.3. If over-learning occurs (verification error starts to rise) try removing hidden units and possibly layers.
5. Train the neural network
6. Validate the performance of the model
7. Implementation

In the previous chapter the above steps are described in detail. The choices made in developing the SPO ANN models are discussed in the following sections where the structure of the networks, the training method used, the preprocessing of the input data and the method used to regularise the network parameters are explained. Then a comparison with a linear regression model is given. The chapter concludes with analysis of simulation results.
5.1 Data Collection

As previously mentioned in section 3.4, the Kitts Green mill computer is linked to a plant wide level III production planning system (SFR). Details of the slabs and the rolling operation that slabs go through is kept on this system and it is possible to obtain any amount of data regarding the rolling process with confidence. However, the control of soaking pits is carried out by a stand-alone system (Nucleus) and past heating durations are stored for a short period only.

SFR stores data on each slab for every pass at the rolling mill. When presented in a matrix format, each row represents a pass with each column describing the slab properties and rolling process variables. Nucleus data is stored as log files in text format. These log files are a record of every event that takes place during the heating operation. Data needs to be extracted manually from the log files and stored in an excel worksheet to be used in modelling the heating time estimator.

Data was collected for three pits (4, 5, 7) over a period of 4 months. 31180 data points from the SFR database and 150 data points from the Nucleus system were collected. A Visual Basic program was written to choose data corresponding to the complete rolling operation of each slab rather than each pass at the rolling mill, so the 31180 (every pass) data points were reduced to 1304 (every slab) data points. These data were used to develop the neural network model for estimating the rolling time of a slab. Estimated rolling time of each slab in a pit load is then summed to find the estimated rolling time of a pit load. The data set used for heating time estimations was combined from three sources. The properties of the slabs were taken from the SFR database, the heating time was taken from the Nucleus log files and the properties of the applied heating recipe was taken from the work practice sheets. However, heating times were not very reliable as the log files were keeping the record of the complete heating duration which can also contain the waiting times. In addition, data were observed to be contradictory at times, arising from faulty readings, as explained by the operators.

Variables for inclusion in the models were found after discussion with manufacturing engineers responsible for the soaking pits and rolling mill process.

The input variables chosen for the rolling time ANN consists of:

- Slab dimensions (width, thickness, length)
- Target gauge \((\text{FmaxG} + \text{FminG})/2\)
- Number of cutbacks (number of plates produced from the slab)
- Temperature
- Hardness constant \((Yk)\)

Table 7 gives an example of the data used for developing the rolling time estimation model for slabs.
Table 7. Data used for rolling time estimation model

<table>
<thead>
<tr>
<th>Width (m)</th>
<th>Thickness (m)</th>
<th>Length (m)</th>
<th>Final Gauge (m)</th>
<th>Num of Operations</th>
<th>Temperature (°C)</th>
<th>Hardness Constant</th>
<th>Rolling Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.34</td>
<td>3.25</td>
<td>25.36</td>
<td>0.26</td>
<td>2.00</td>
<td>428.00</td>
<td>2.00</td>
<td>16.28</td>
</tr>
<tr>
<td>11.99</td>
<td>3.25</td>
<td>31.36</td>
<td>0.10</td>
<td>5.00</td>
<td>394.00</td>
<td>2.00</td>
<td>33.59</td>
</tr>
<tr>
<td>10.99</td>
<td>5.25</td>
<td>28.20</td>
<td>0.19</td>
<td>3.00</td>
<td>419.00</td>
<td>2.00</td>
<td>25.43</td>
</tr>
<tr>
<td>10.99</td>
<td>5.25</td>
<td>28.65</td>
<td>0.16</td>
<td>4.00</td>
<td>366.00</td>
<td>2.00</td>
<td>34.23</td>
</tr>
<tr>
<td>9.20</td>
<td>3.40</td>
<td>21.10</td>
<td>0.76</td>
<td>1.00</td>
<td>511.00</td>
<td>2.70</td>
<td>13.68</td>
</tr>
<tr>
<td>9.20</td>
<td>3.40</td>
<td>21.10</td>
<td>0.46</td>
<td>1.00</td>
<td>503.00</td>
<td>2.70</td>
<td>7.02</td>
</tr>
<tr>
<td>9.20</td>
<td>3.40</td>
<td>27.30</td>
<td>0.30</td>
<td>1.00</td>
<td>468.00</td>
<td>2.70</td>
<td>8.02</td>
</tr>
<tr>
<td>9.20</td>
<td>3.40</td>
<td>30.20</td>
<td>0.30</td>
<td>1.00</td>
<td>504.00</td>
<td>2.70</td>
<td>10.06</td>
</tr>
<tr>
<td>12.00</td>
<td>3.25</td>
<td>26.36</td>
<td>0.36</td>
<td>1.00</td>
<td>469.00</td>
<td>2.70</td>
<td>8.77</td>
</tr>
</tbody>
</table>

The heating time ANN estimates the total heating time (reheating + soaking time) of a pit load. A pit load may be made of a few batches which can have different metallurgical characteristics. The details of each batch are entered separately in the heating time ANN. A pit load may consist of a maximum of three batches. When the pit load is made of less than 3 batches, the details of absent batches are set with 0.

The input variables for the heating time ANN consists of:

- From soaking recipe
  - Soaking temperature
  - Soaking duration
  - Preheat, cutback and cooling temperature
- From each batch
  - total volume and surface area of the metal
  - volume and surface area of each batch
  - number of slabs
  - heating coefficients

Table 8 gives an example of the data used for developing the heating time estimation model. The volume and area values are scaled to decrease the values of their range.

Table 8. Data used for heating time estimation model

<table>
<thead>
<tr>
<th>Soak Time</th>
<th>Temp1</th>
<th>Temp2</th>
<th>Temp3</th>
<th>Heat Coe1</th>
<th>Heat Coe2</th>
<th>Heat Coe3</th>
<th>Visual</th>
<th>Atotal</th>
<th>V1</th>
<th>A1</th>
<th>V2</th>
<th>A2</th>
<th>V3</th>
<th>A3</th>
<th>Num Slabs</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>580</td>
<td>485</td>
<td>435</td>
<td>0.3024</td>
<td>0</td>
<td>0</td>
<td>0.1095</td>
<td>0.0986</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>500</td>
<td>437</td>
<td>0</td>
<td>0.701</td>
<td>0.705</td>
<td>0</td>
<td>0.2034</td>
<td>0.1494</td>
<td>0.5795</td>
<td>0.6392</td>
<td>0.4202</td>
<td>0.3608</td>
<td>0</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>590</td>
<td>487</td>
<td>0</td>
<td>0.6042</td>
<td>0.6061</td>
<td>0</td>
<td>0.1720</td>
<td>0.1508</td>
<td>0.0892</td>
<td>0.8221</td>
<td>0.1709</td>
<td>0.1779</td>
<td>0</td>
<td>0</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>500</td>
<td>437</td>
<td>0</td>
<td>0.7175</td>
<td>0.7475</td>
<td>0.7075</td>
<td>0.1527</td>
<td>0.1378</td>
<td>0.0470</td>
<td>0.0477</td>
<td>0.0535</td>
<td>0.0977</td>
<td>0.9060</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>590</td>
<td>487</td>
<td>0</td>
<td>0.6042</td>
<td>0</td>
<td>0</td>
<td>0.0829</td>
<td>0.0710</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>500</td>
<td>437</td>
<td>0</td>
<td>0.7075</td>
<td>0</td>
<td>0</td>
<td>0.0139</td>
<td>0.0116</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>500</td>
<td>437</td>
<td>0</td>
<td>0.701</td>
<td>0</td>
<td>0</td>
<td>0.2182</td>
<td>0.1248</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>520</td>
<td>472</td>
<td>417</td>
<td>0.7022</td>
<td>0</td>
<td>0</td>
<td>0.1822</td>
<td>0.1344</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>500</td>
<td>437</td>
<td>0</td>
<td>0.701</td>
<td>0</td>
<td>0</td>
<td>0.2040</td>
<td>0.1364</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Lack of adequate reliable data on heating process durations has resulted in poor performance in SPO heating time estimations. For this reason, although we explain the work done for the development of both models, we only present the results of rolling time estimations. For improving heating time model performance, gathering further
reliable data is necessary. There are many other parameters stored on the SFR
database that maybe taken into account for further improving the performance of the
SPO rolling time ANN too. The next phase of this research includes a connection to the
nucleus system and collecting more reliable data for improving heating time
estimations as well as including more inputs and gathering a more evenly distributed
data across the operation range for improving rolling time estimations.

5.2 Outlier Detection and Data Cleaning

One problem with using industrial process data to build empirical models is that the
data may contain points that are incorrect for one reason or another. These are known
as outlying data points or outliers. If the training data contains faulty points especially
in an area of low data density, ANN models tend to fit toward the incorrect data point.
Without adequate data cleaning and model testing there is a danger of implementing a
model that does not accurately represent the underlying process in the region of that
faulty data. Another important check to perform on a training set is that it is not
attempted to model a function which consistently maps the same input onto two or
more different outputs. If it does, the average of the possibilities will be learnt, that
would yield wrong results. Such cases are examples of ill posed problems. If it is
possible, splitting the data into several parts and training a different network on each
set can provide a simple solution. It may also be possible to recode a data set so that
it becomes a one to one mapping. One way is to throw away contradictory examples
by simply keeping training examples which are consistent and discarding all others.
Another method is to add an extra term to the network error function to constrain the
network behaviour.

Before constructing a model based on industrial process data, it is important to be
familiar with both the process and its data. Familiarization provides an understanding
of data accuracy and can indicate how faulty data points might be detected. Basic
topics such as data format, distribution, variable importance, trends, ranges, and units
compatibility must be understood. This helps to obtain the most use from the available
data, and it helps to judge how many useful data points are available for model
construction. Codes which relate to the production of the slabs are also important as
they show the slabs with similar characteristics. This information is useful during
outlier detection and correction. Once the operating levels of the inputs and outputs
have been defined an obvious check is whether the data set contains any points that
violate these limits. A graphical inspection method can be effective in detecting these
outliers. If there are many similar examples of a given input pattern, an outlying data
point can be classified as the one furthest from the median value. A set of similar
input variables would usually relate to similar output values. A check can be made
through the data set to find similar input patterns together with their respective
output patterns. When a neural network model has been constructed, the residuals
should also be examined. There are two reasons for high residuals. The first is a data
point which the model cannot cope with but one which is correct, and the second is a
point which is incorrect, and therefore does not fit with the model.
When a data point is found to be incorrect, there are several options to deal with it. First, data point can be deleted from the set. This is acceptable if other similar examples are available and only a minor effect in the data numbers results. If data are in short supply for that area of the input space, a replacement may be necessary, treating the corrupted value as a missing value. The ideal replacement is that provided by an expert. This can be achieved if a partial expert has access to data from the original time of manufacture. If a faulty point is treated as a missing value and a replacement is required without expert knowledge, there are a number of solutions. If the outliers occur at random, missing values can be found by interpolation, assuming that the data is uniformly distributed. Linear or more advanced regression can be used, however this is only done with less than 3 consecutive missing points. Principal Component Analysis (PCA) and Partial Least Squares (PLS) techniques can also be used to treat missing values, since these techniques allow blanks in the data. PCA can be applied to the training data with some missing values, then a neural network model based on principal components can be constructed. Another approach is to estimate the missing value for an n-dimensional input vector, from knowledge of the other n-1 input variables. The nearest neighbour within the training data set can be identified and the value for the nearest neighbour can be substituted as the missing value. Auto-associative neural networks can also be used to fill in the missing value, although this requires a high level of redundancy within the neural network.

For cleaning the data that was gathered for building the SPO ANN models, a basic detection in the form of maximum-minimum variable values has been employed. It was noted that there were some faulty data readings such as 1.5 years of rolling. Five data points were discarded at this stage. The remaining data 1299 points were saved as OrigCa.xls.

Further analysis of data resulted in identifying 89 more outliers which were discarded too. These data points were the cases where the NN model produced large errors and compared to data points with similar values they had unusual output values which could only be explained as faulty data. Table 9 shows an example of such a data point in the first row. The remaining 1210 data points were saved as OrigCi.xls. Table 10 shows the statistics of the cleaned data, OrigCi.xls. Median value is the 50th percentile of a sample. Since outliers have little effect on median value, it is a robust estimate of the centre of a sample of data. The difference between the mean and median hints the presence of more outliers. Skewness is a measure of the asymmetry of the data around the sample mean. If skewness is negative, the data are spread out more to the left of the mean than to the right. If skewness is positive, the data are spread out more to the right. The skewness of the normal distribution (or any perfectly symmetric distribution) is zero. The values on Table 10 indicate that it is possible to remove more outliers from the data and that more data should be collected for operation modes where data samples are rare.
The effectiveness of cleaning the data is demonstrated by comparison of the rolling time ANN model performances. Data sets were divided into training and test sets. Training data is made of ¾ of the gathered data while test data is ¼ of the gathered data. Feedforward ANN models with ten hidden layer neurons were trained on cleaned and uncleaned data. Models were then used to predict test results. Figure 49 shows regression analysis between predicted and actual values of rolling time for each case. A perfect prediction would fall along a \( \text{Estimate} = 1.0 \times \text{Actual Value} + 0.0 \) best linear fit line. Deviation of the best linear fit line from the \( \text{Estimate} = 1.0 \times \text{Actual Value} + 0.0 \) line indicates that the model is biased towards estimating consistently either higher or lower than the actual value which is usually caused by uneven distribution of training data. The best linear fit for unclean data is:

\[
\text{Estimate} = 0.223 \times \text{Actual Value} + 24.7
\]  

(5.1)

whereas for the cleaned data it is

\[
\text{Estimate} = 0.831 \times \text{Actual Value} + 0.886
\]  

(5.2)

Deviations from the best linear fit is desired to be low, ideally zero, for a successful model. As can be seen from Figure 49, 95% deviation bands on the best linear fit are much smaller in the case of cleaned data. The correlation coefficient (normalized measure of linear relationship strength between variables) between the estimates and actual values is 0.343 for uncleaned data and 0.916 for cleaned data. It can easily be concluded that data cleaning has resulted in a significant improvement in successfully developing a rolling time estimation model.

When learning a continuous function, it is also desirable to collect an even distribution of data covering all aspects of the system to be modelled. The main problem occurs when models are developed to cover the entire operation of a system which may rarely, if ever, display certain modes of operation. The simplest such case involves a system which covers its entire functionality but not with an even frequency. It is important, to ensure that the training set covers the whole range evenly by duplicating examples of rare behaviour or explicitly collecting more examples of such cases. The harder case involves a system which only enters certain states of interest.
when things have gone wrong. This problem is particularly severe in system monitoring networks which need to detect failures which would be dangerous or impossible to induce for data collection purposes.

Figure 49. Performances of ANN models trained with unclean and clean data.

Figure 50 shows a three dimensional plot of the distribution of the training and test data used in developing SPO ANNs, where Data consists of 7 columns corresponding to the inputs and Data Centres correspond to the bins that data is divided into. An ideal even distribution would have looked like a smooth plateau. However, it can be seen that for some regions of the operation range there are plenty of samples whereas samples of certain modes of operation are rare. This would cause the ANNs performance to vary depending on the mode of operation estimated at. The estimations corresponding to the regions rich with samples are expected to be accurate whereas reliability of the estimates at regions with rare samples would be low. In order to use the ANN modelling technique effectively it is necessary to have a measure of variance in performance. This is achieved by using an ensemble modelling approach explained in section 5.4.

5.3 Preprocessing

The hidden layer neurons of the ANN model structure used for developing the SPO ANNs have tansig transfer functions. The target outputs of these neurons lie in the range from minus one to plus one. The reason behind this is that the output from the
tansig function can only reach values in that range. Some output functions operate over different ranges. The sigmoid, for example, covers the range from zero to one. Linear output units are unbounded in the range they can cover. It is necessary in the former cases and desirable in the latter to scale a data set so that every value falls within a given range. Although input units are usually linear it is still desirable to scale the data before training. One advantage of such a practice is that we are forced to make a few basic statistical considerations concerning the distribution of the training data and effect of outliers in the training set. Another reason for scaling each variable to within a fixed range is that there may be situations where some variables cover a range of significantly larger numbers compared to other variables (e.g. [100 1000] and [0.1 0.3]). The errors due to the higher valued variable will have a greater effect during training than those with lower values, as their magnitude will be greater. Ensuring that every variable covers the same range ensures that errors on each variable contribute the same proportion to the change in network weights.

In the case of the input units, it might seem prudent to try and ensure that the activation they pass to the hidden layer falls within the correct operating input range for the hidden unit activations functions. In reality we need not worry about the input range on this account as the weights and the bias unit move to the incoming values to the correct position for the hidden units' transfer functions. It can however, make the task facing the bias unit easier if the input values are small and have a mean of zero. For this reason and those discussed in the previous paragraph it can be prudent to normalize each variable to zero mean and a standard deviation of one for input to the network.

The simplest method for squashing a set of data into a given range is by subtraction and division. If the desired range falls between zero and one, then each value, must have the lowest of all the values subtracted from it and the result divided by the original range to generate the new value. This has the advantage of preserving the relative positions of each data point along the range. Linear scaling works best when the data is spread evenly over its range. Using linear scaling on data set which contains outliers or an uneven spread of values results in a large proportion of the data values being squashed into a small part of the input range, leaving most of the rest unused.

After gathering and cleaning the data, the next step taken in developing the SPO ANN was to scale the data into the range [-1, 1]. First they were scaled to convert all the variables to similar units. Then the data was preprocessed by normalising using the formula:

\[
X_n = 2 \times \frac{X - \text{minimum}(X)}{\text{maximum}(X) - \text{minimum}(X)} - 1
\]  

(5.3)
Another preprocessing method that was tried was done by normalising the data so that they have means of zero and standard deviations of 1. This method eliminates the offsets in the data which saves the models from having high bias weights to correct the levels. The following formula was employed for this normalization:

\[ X_n = \frac{X - \text{mean}(X)}{\text{standard_deviation}(X)} \]  

There are various other methods that are commonly used for preprocessing data. One of which is transforming the input variables using Principal Components Analysis (PCA). PCA is most useful when some variables are correlated and there is high dimensionality problem. The dimension of the input vector that is used in SPO ANN models is not very high and inputs were not highly correlated so a principal component analysis step was not employed.

### 5.4 Ensemble Model

If the answer derived from a neural network model influences the decisions made concerning the real world processes being modelled, it would be useful to know how much confidence to place on a result. If the variance or reliability of the training data is not consistent across its entire range, then it would be necessary to be able to model not only the system’s behaviour under certain conditions, but its predictability under each condition too. Detecting answers given with less than 100% certainty in a continuous valued mapping function is harder than it is for classification tasks as there is no clue in the output value itself as to whether we are in a doubtful region or not. Points from different areas of input space could give rise to different levels of error for several reasons. Firstly, some parts of complex nonlinear functions are easier to learn than others. For example, areas of high curvature are harder to learn. Secondly, the training data may represent some parts of the function to be learned better than others. Thirdly, there could be more variance in some parts of the training data than others. Poor generalization could result from areas where the system is very complex, data is very sparse or variance is very high.

Building a neural network for any but the most trivial applications requires that several networks are built, each of different complexity, or stopped at different points during training, or simply started from different random weight configurations. Each network should be saved, tested and analysed, and the most appropriate finally chosen. A single final solution may be chosen from the population of networks produced during training, or a collection of networks may be used which is referred to as an ensemble model or committee. An ensemble of networks can improve prediction accuracy, especially when the errors of the networks are not correlated. Results from several networks in the committee can be combined in several different ways. The average of all committee members can be calculated, or the output of the network with the most confidence may be used. Networks can be weighted with respect to their test error.
scores so that better networks contribute to the answer with greater strength. Better might mean more accurate or may relate to costs associated with choosing an answer. Another advantage of using a panel of neural networks is that the variation in output values from each may be used as an indication of the reliability of the final answer.

The ensemble modelling approach adopted to build SPO ANNs was first to generate a diversity of individual candidate models with good performances then to combine all the individual models into an ensemble model. Nine individual models that had three layered feedforward structure were created with varying numbers of hidden layer neurons. In the ensemble stage each individual model was treated equals and their mean prediction was taken as the final output of the ensemble model. Output of the ensemble model was calculated as:

\[
y(k) = \frac{1}{N} \sum_{i=1}^{N} y^i(k)
\]

where \( y(k) \) is the output of the ensemble model for its \( k \)th input, \( y^i(k) \) is the corresponding output of the \( i \)th neural network model.

The variations in the predictions arising from individual models for the same input indicate how reliable the predictions are. The standard deviations of these variations was used to calculate the confidence in the predictions with the following formula.

\[
EB(k) = 2\sigma(k) = 2\sqrt{\frac{1}{N} \sum_{i=1}^{N} (y^i(k) - y(k))^2}
\]

where \( EB(k) \) is the error bounds for the \( y(k) \) with 95% confidence, and \( \sigma(k) \) is the standard deviation among the output prediction \( y^i(k) \) of the individual models.

Table 11 shows the comparison of candidate models and the ensemble model. Ensemble model has high prediction performance both on training data and unseen test data. Training data is learned with a higher accuracy as can seen from the comparison of errors. This shows that predictions of ensemble model are more reliable.

<table>
<thead>
<tr>
<th>Number of Hidden Layer Units</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>Ensemble</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation Target v Predicted TEST</td>
<td>0.915</td>
<td>0.917</td>
<td>0.919</td>
<td>0.916</td>
<td>0.912</td>
<td>0.902</td>
<td>0.916</td>
<td>0.902</td>
<td>0.902</td>
<td>0.919</td>
</tr>
<tr>
<td>Correlation Target v Predicted TRAINING</td>
<td>0.934</td>
<td>0.935</td>
<td>0.938</td>
<td>0.940</td>
<td>0.941</td>
<td>0.946</td>
<td>0.945</td>
<td>0.947</td>
<td>0.948</td>
<td>0.942</td>
</tr>
</tbody>
</table>

### 5.5 Structure

In this section, we explain the structure of individual candidate models used to build the ensemble model which was described in section 5.4. Different neural network paradigms have been proposed with individual advantages and limitations for different application domains. In general they all consist of large numbers of simple processing elements called neurons, connected together by weighted links known as synapses.
The latter are adapted during training to produce the required network function. Neural networks can differ in several respects including network topology and information flow, the choice of neuron function and whether supervised or unsupervised training is employed.

The neural network architecture used in SPO ANN models is a multi-layer feed-forward structure. The structure consists of three layers: the input layer, the hidden layer, and the output layer. Kurkova has shown that only one layer of hidden units suffices to approximate any function with finitely many discontinuities to arbitrary precision, provided the activation function of the hidden units are nonlinear [114]. This does not necessarily imply that a network with more layers might not more conveniently or easily model a particular problem. In practice, however, most problems seem to yield to a single hidden layer, with two an occasional resort and three practically unknown. The first layer of SPO ANNs is the input layer, which consists of neurons where input variables are plugged into the network. Each input layer neuron is connected to the middle layer neurons. These connections are weighted by the importance of the input neurons. The middle layer, which is also known as the hidden layer, is made up of neurons with tansig transfer functions. The last layer is the output layer where there is only one neuron with a purelin transfer function. Only one neuron is needed because only one output, the heating or rolling time is requested. This neuron is connected by weighted links to all of the neurons in the middle layer. The neurons in each layer receive their input from neurons of a layer directly below and send their output to the neurons in a layer directly above. There are no connections within the layer. Information flows from input to output, with no feedback or connections within the layers themselves. Figure 51 shows a diagram of the architecture used.

Figure 51. Multi-layer feedforward ANN architecture (bias weights not shown)
The activation function of a neuron sums the weighted inputs from all connected units and squashes it into a set range. Equation (5.7) is the definition of an activation function of the weighted inputs plus a bias.
\[ A_k(t+1) = F_k \left( \sum_j w_j(t)A_j(t) + \theta_k(t) \right) \]  

(5.7)

where \( \theta \) is the node bias, \( A(t) \) is the nonlinear activation function and \( w_j \) is the weight connecting the \( j^{th} \) output of the layer proceeding that node. Figure 52 shows a hyperbolic tangent activation function and Figure 53 is a linear transfer function.

Modern computers can calculate a hyperbolic tangent function faster than a sigmoid although derivatives of each are similar in nature, so neurons with tansig activation function should be faster. Another advantage of the tansig function is related to its \([-1 1]\) range. It can often be desirable to normalise a training set so that it has zero mean and unit standard deviation. Such normalization is only possible with an activation function which is able to output negative values. Finally, an asymmetric function such as tansig leads to faster learning. Networks with asymmetric activation functions (a function is asymmetric if \( f(-x) = -f(x) \)) have been found to require fewer training epochs than networks with non-symmetric activation functions. In SPO ANN models activation functions are selected as tansig(n) for the hidden layer nodes and purelin(n) for the output neuron.

The characteristics of the slab groups, and heating and rolling recipes are the nine inputs of each SPO ANN model. The output of one ANN is the heating time and the other the rolling time.

The complexity of a model is controlled by the number of hidden units. The inferred noise level decreases as the number of hidden units increases. However, the complexity of the model also increases with the number of hidden units. A high degree of complexity may not be justified, and in an extreme case, the model may in a meaningless way attempt to fit the noise in the experimental data. To find the optimal number of hidden nodes, tests have been run several times for the number of hidden nodes as described in section 4.3.6. SPO ANNs with 4 to 12 hidden layer neurons have shown the best performance.
5.6 Training Method

The specification of the network structure, together with the set of weights is a complete description of the formula relating the inputs to the output. The weights are determined by training the network. A supervised learning structure is used for training the SPO ANNs, in which the error between the output of the network and the desired output is used to adopt the network weights.

Because of the training method used, the SPO ANNs can be classed as backpropagation ANNs. Backpropagation is composed of taking the error from the output layer, through the middle layer and back to the input layer. Then mathematical methods like Conjugate Gradient Descent and Levenberg-Marquart are used to adjust the weights to minimize the error so that the desired outcome is achieved. Each method has different computation and storage requirements, and no one algorithm is best suited to all applications. Comparing the converging speed, the training time and the prediction accuracy, it was found that the Levenberg-Marquardt algorithm works best for the data used to train SPO ANNs. Hence the Levenberg-Marquardt optimisation algorithm was employed.

The simplest implementation of backpropagation learning updates the weights in the direction in which the performance function decreases most rapidly (the negative of the gradient). There are two different ways in which this gradient descent algorithm can be implemented: incremental mode and batch mode. The act of presenting the entire training set to the network once is known as an epoch. In the incremental mode the gradient is computed and all weights are updated after each input is applied to the network. Due to the fact that the task of a neural network is to reduce the average error over the entire training set, it might seem a good idea to keep a running total of the errors produced by each output unit given each pattern in an epoch. Once the epoch has been completed, a single average error can be calculated and the network updated once according to that average error. This mode of operation is referred to as batch mode learning. In training experiments on SPO ANNs, the batch training method worked better when compared in terms of the converging speed, the training time and the prediction accuracy.

5.7 Generalization

Generalization is the ability of the neural network to make decisions with the data different than the data it was trained with. One of the difficulties with blind data modelling is that of overfitting, in which spurious details and noise in the training data are overfitted by the model. This gives raise to solutions that generalise poorly.
MacKay and Neal have developed a Bayesian framework for neural networks in which the appropriate model complexity is inferred from the data [115, 116].

To improve the generalization of the SPO neural networks the optimal regularization parameters are determined in an automated fashion using the Bayesian framework in which the weights and biases of the network are assumed to be random variables with specified distributions [104]. The regularization parameters are related to the unknown variances associated with these distributions. These parameters are estimated using statistical techniques. A discussion on the Bayesian framework for regularization and references for further details is given in section 4.3.6.

### 5.8 Linear Regression Model

As modern industrial processes are associated with large amounts of data, a range of empirical modelling techniques are increasingly being used to describe these processes. The neural networks can learn from a dataset to describe nonlinearities and interaction effects that are usually encompassed by these processes. This makes neural network technique more popular, compared with the more traditional linear and polynomial regression techniques. Section 4.4 explains the linear regression technique and advantages of using ANNs over a regression model. In order to develop models for the rolling and heating time estimations, a multiple linear regression (MLR) technique was tried too. This section presents the results of the MLR models obtained from the data used in developing SPO ANNs. Figure 54 and Figure 55 present the performance analysis of the MLR model on the training and test data respectively. Table 12 summarizes the analysis of the performance.

![Figure 54. MLR performance on training data](image)

![Figure 55. MLR performance on test data](image)

<table>
<thead>
<tr>
<th>Table 12. Analysis of MLR performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Best Linear Fit (BLF)</td>
</tr>
<tr>
<td>Magnitude of 95% deviation band from BLF</td>
</tr>
<tr>
<td>Correlation coefficient of Estimates &amp; Actual</td>
</tr>
</tbody>
</table>
Although the results obtained from the ANN models illustrated in section 5.9 are much better, it can be seen that there is a strong linear relation between the selected input variables and the output. However, error bounds of the MLR model are too high and it does not offer all the capabilities of an ensemble ANN model. ANN models are better in dealing with the interactions among the input variables and the variations among the outputs of individual networks that make up an ensemble model is a good measure of the reliability of the estimations along the model’s operation range. Figure 56 presents the comparison of the MLR model estimations with the actual rolling times together with the error bounds of the estimations. In this figure, the data is sorted to present the output in an ascending order so that the behaviour of the model across the operating range can easily be examined. Figure 57 and Figure 58 show a smaller number of the data cases in order to make a comparison easily. The MLR model predicts rolling times with same reliability level along the whole operation range unlike an ANN ensemble model which is very accurate at points for which there is plenty data to train with and less reliable at the points where there is few data to learn from and ensemble model’s confidence bounds provides the information on the reliability of the estimations.

![Comparison of MLR model predictions and actual rolling times](image)

**Figure 56.** Comparison of the MLR model predictions and actual rolling times

![MLR model estimations and error bounds](image)

**Figure 57.** MLR model estimations and error bounds

![Comparison of MLR model estimations and actual rolling times](image)

**Figure 58.** Comparison of MLR model estimations and actual rolling times
5.9 Results and Analysis for ANN model

In this section, the results of the SPO ANN model for rolling time estimations is presented. First the result of estimating rolling time of slabs is presented, then the results of estimating the rolling time of pit loads which consist of a number of slabs. Figure 59 and Figure 60 show the regression analysis of estimations versus actual durations for rolling times of slabs. Table 13 summarises the results of this analysis. The results demonstrate the ability of ANN models to estimate rolling times. However, it can also be seen that this performance can be further improved. There are a number of outliers that lie outside the 95% confidence bounds of the best linear fit of (estimate, actual duration) data points. These outliers need to be further analysed and corrected. This would improve the performance of the ANN models further. Figure 61 shows the ANN ensemble model estimations and the corresponding 95% confidence bounds. In this figure, the data is sorted to present the output in an ascending order so that the behaviour of the model across the operating range can easily be examined. The confidence bound fluctuates depending on the corresponding data points, but generally the level is quite small. It gets bigger at the higher and lower ends of the operation range. This can be explained by the uneven distribution of the data samples available across the operation range and possible outliers, which draws attention to another direction for future work for improving the performance of the ANN models further. A smaller number of data samples are examined in Figure 62 and Figure 63. It can be seen from these figures that ANN models can successfully predict the rolling times and also help in identifying the data points at which we should expect a higher error in estimations. For example, at the 114th data sample, the error bound is very small in Figure 62 and the corresponding estimate in Figure 63 is very accurate whereas at the 127th data sample, the error bound is high and the corresponding estimation is less accurate.
Table 13. Analysis of ANN ensemble performance for slab duration

<table>
<thead>
<tr>
<th></th>
<th>Training data</th>
<th>Test data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Linear Fit (BLF)</td>
<td>E = 0.879A + 1.92</td>
<td>E = 0.815A + 1.18</td>
</tr>
<tr>
<td>Magnitude of 95% deviation band from BLF</td>
<td>5.9</td>
<td>6.8</td>
</tr>
<tr>
<td>Correlation coefficient of Estimates &amp; Actual</td>
<td>0.942</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Figure 61. ANN ensemble model predictions and 95% confidence bounds

Figure 62. Ensemble model estimates for slabs & confidence bounds

Figure 63. Ensemble model estimates & actual rolling times of slabs

To estimate the rolling time of a pit load, the ANN model used to estimate the rolling time of slabs is used. The rolling time is estimated for each slab in the pit load and then summed. The model is built with ¾ of the collected data as mentioned earlier. The pit loads that are analysed here correspond to all the data collected. Figure 64 presents the regression analysis between pit load rolling time estimations (by ANN) and actual values. Table 14 summarises the results of this regression analysis. These results reinforce the proof of the capability of ANN models in successfully estimating the rolling times. Figure 64 and Figure 66 present the comparison of estimations and actual values of a smaller number of data samples. The green uncertainty line in Fig 66 corresponds to the standard deviation of the estimations by the ANNs forming the ensemble model. ANN ensemble model can accurately estimate rolling times and identify the data points at which higher errors should be expected. For example, at the...
55th data sample the 95% confidence bound is very small and the corresponding estimation is very accurate whereas at 65th the 95% confidence bound is large and the corresponding estimation is less accurate.

Table 14. Analysis of ANN Ensemble performance for Pitloads

<table>
<thead>
<tr>
<th>Pit load rolling durations</th>
<th>Best Linear Fit (BLF)</th>
<th>Magnitude of 95% deviation band from BLF</th>
<th>Correlation coefficient of Estimates &amp; Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[A = (1.23) T + (-0.0763)]</td>
<td>[0.64\text{hours} = 38.4 \text{mins}]</td>
<td>0.955</td>
</tr>
</tbody>
</table>

The expected error in estimations of pit load rolling times is 27.6 minutes and standard deviation of errors is 26.9 minutes. The rolling time can be estimated with less than 15 minutes of error for 45% of the cases, 30 minutes of error for 65% of the cases, and 45 minutes of error for 78% of the cases.

At present both heating and rolling time of pit loads are estimated manually when soaking-pits/rolling-mill schedules are produced. In discussions with the scheduling operators and engineers in charge of the process, it was noted that an error of an order of an hour is anticipated for rolling time estimates used in the present
schedules. Using the ANN model, this can easily be reduced to half an hour and by using the confidence bounds as additional information for decision making, major improvements can be achieved in comparison to the present schedules. In fact, reducing the error window for a schedule of 16 jobs (for the example in section 9.4) can reduce an additional 8 hours from the makespan of an optimal schedule which is a major gain in terms of saved costs and opportunity created for more manufacturing.

5.10 Implementation

For developing SPO ANN models, an initial decision was whether to develop complete software in a basic programming language or use Matlab. Initially we successfully implemented the structure described above in C++ by modifying the classes provided in [117]. This exercise proved very useful to understand the mechanisms of ANNS. However the training algorithm was a simple gradient descent method and the performance obtained did not match with that obtained from Matlab. Due to time considerations needed for improving the software, it was decided to continue the work using Matlab. Matlab can connect to the other software easily and can be used as a part of the SPO in the completed project. The data that is used to develop the networks offline was saved in Excel files. The selection of relevant data was done in excel using macros written in Visual Basic. The detected outliers were removed from the excel files. The connection to Matlab was established via a 'dde conversation' [e.g. >>channel = ddeinit ('excel', 'OrigCi.xls')]. Dividing the data into training and test data sets, the preprocessing of data and the rest of the development work was done in Matlab.

5.11 Summary

In this chapter, SPO ANN models that are used to predict the duration of the heating and rolling processes are explained. The aim of the work is to predict these durations, which are estimated manually at present, with better accuracy and resolution. The results for rolling time estimations are presented and discussed. It is shown that the rolling time ANN model achieves major improvements in accuracy of the estimations and consequently in the schedules prepared using these estimates. Directions for further improving these results and completing the heating time model are also noted. A comparison with a multiple linear regression model built using the same data is presented too. The MLR model performance shows that there is a strong linear relation between the inputs and rolling time. But MLR model error bounds are very high which make it not useful in producing reliable schedules. However, the ANN ensemble model's performance is observed to be very accurate at most data points and confidence bounds derived from the variations in the predictions of the individual models in the ensemble model provides a measure to identify estimations that can be
110

wrong. Increased accuracy in estimating the duration of the processes using SPO ANN models makes the schedules more reliable and shortens the makespan, which result in major benefits in terms of the costs incurred and opportunity lost for not using the resources for more production.
6 Optimisation and Computational Complexity

Scheduling of the soaking pits/rolling mill process is a difficult combinatorial optimisation problem. In order formulate this problem and to understand the challenges that are addressed, it is necessary to have an appreciation of state of the art optimisation theory and similar combinatorial optimisation problems. To evaluate the applicability of proposed solution methods and to compare performance of different algorithms, one needs to understand the computational complexity theory. For these reasons, this chapter is devoted to providing necessary background on optimisation in general, combinatorial optimisation, and computational complexity.

The chapter starts with a brief introduction to optimisation theory and follows with a review of combinatorial optimisation theory and solution methods for combinatorial optimisation problems. Then computational complexity is explained and details on measuring the efficiency of algorithms are presented. The chapter concludes with a description of the different classes of problems according to their complexity. The information provided in this chapter is later used in discussing the developed SPO system in chapter 9.

6.1 Optimisation

Optimisation means finding a best solution among several feasible alternatives. The term a best solution is used because there may be more than one optimal solution. Optimisation deals with problems of finding the values of variables that minimize or maximize a function of several variables subject to constraints. It is used for calculating the best possible utilization of resources like people, time, processes, vehicles, equipment, raw materials, supplies, capacity, securities, etc. needed to achieve a desired result, such as minimizing cost or process time or maximizing throughput, service levels, or profits.

Almost all optimisation problems have a single objective function. But in some cases, the goal is to find a set of variables that satisfies the constraints of the model. The user does not particularly want to optimise anything so there is no reason to define an objective function. These types of problems are called feasibility problems. Another exception arises when we have multiple objective functions. Here the user would like to optimise a number of different objectives at once. Usually, the different objectives are not compatible; the variables that optimise one objective may be far from optimal for others. In practice, problems with multiple objectives are reformulated as single-objective problems by either forming a weighted combination of the different objectives or else replacing some of the objectives by constraints.
Constraints are not always essential although it can be argued that almost all problems really do have constraints. In practice, answers that make good sense in terms of the underlying physical or economic problem can often be obtained without putting constraints on the variables.

The formal development of optimisation theory came from calculus although theories of optimisation existed long before the development of calculus. After the invention of calculus, mathematicians worked actively on optimisation problems. The theory was developed for mathematical models containing continuous variables and differentiable functions. The theory provided solution procedures for many problems with several variables, but was not adequate to deal computationally with models containing large number of variables. The classical development of optimisation theory through calculus was essentially complete by the end of the nineteenth century. A good exposition to this is given in [118].

In the 1940's there was a reawakening and a change of direction in the study of optimisation theory. The work of scientists and mathematicians on military operational problems, and the invention and development of the digital computer were the two occurrences especially significant for this reawakening.

The scientific approach to military, industrial and other institutional problems became the field of study known as operations research [119, 120, 121]. The formulation and solution of mathematical models of optimisation is an integral part of operations research. These models of complex logistic, production, and distribution systems are generally characterized by a large number of variables, and are often of a form not amendable to solution by calculus.

Pioneering operations research analysts developed models and solution techniques with successes partly due to the rapid evolution of high-speed digital computers. The capacity to do thousands of calculations per second made solving problems containing hundreds or even thousands of variables possible and stimulated the study of iterative optimisation schemes and eventually led to the development of linear and non-linear programming, dynamic programming, and various search methods.

The systematic search for methods that solve the global optimisation problem probably began in 1975 with [122]. Before this date, researchers in the optimisation community regarded the solution of general smooth, nonconvex global optimisation problems as being beyond tractability, and the multiple random start heuristic was the only means to try to find better minima.
Of course, there are global optimisation problems that are exceptionally hard. For some classes of global optimisation problems, even NP-hardness, the mathematical seal for the existence of intractable instances can be proved [123].

However, many problems are much easier than the worst case. [124] shows how much the picture has changed since 1975, and that a wide variety of global optimisation problems arising in applications can be successfully treated. [125] and [126] are two other reference books on global optimisation. [127] presents the state of the art in deterministic global optimisation. A good source on information about heuristic global optimisation methods for combinatorial problems is [128]. [129] is a good reference on genetic algorithms.

6.2 Combinatorial and integer optimisation

Combinatorial optimisation is the process of finding one or more optimal solutions in a well defined discrete problem space. Such problems occur in almost all fields of management, as well as in many engineering disciplines. A survey of related applications of combinatorial optimisation is given in [130].

Combinatorial optimisation problems are concerned with the efficient allocation of limited resources to meet desired objectives when the values of some or all of the variables are restricted to be integral. Constraints on basic resources, such as labour, supplies, or capital restrict the possible alternatives that are considered feasible. Still, in most such problems, there are many possible alternatives to consider and one overall goal determines which of these alternatives is best. For example, a flexible manufacturing facility needs to schedule the production for a plant without having much advance notice as to what parts will need to be produced that day. In today’s changing and competitive industrial environment the difference between using a quickly derived solution and using sophisticated mathematical models to find an optimal solution can determine whether or not a company survives.

The versatility of the combinatorial optimisation model stems from the fact that in many practical problems, activities and resources, such as machines, airplanes and people, are indivisible. Also, many problems have only a finite number of alternative choices and consequently can appropriately be formulated as combinatorial optimisation problems. The word combinatorial refers to the fact that only a finite number of alternative feasible solutions exist. Combinatorial optimisation models are often referred to as integer programming models where programming refers to planning so that these are models used in planning where some or all of the decisions can take on only a finite number of alternative possibilities.
Although some research has centred on approaches to problems where some or all of the functions are nonlinear, most of the research on combinatorial optimisation to date covers only the linear case. A survey of nonlinear integer programming approaches is given in [131].

Finding an optimal solution to combinatorial optimisation problems can be a difficult task. The difficulty arises from the fact that unlike linear programming, for example, whose feasible region is a convex set, in combinatorial problems, one must search a lattice of feasible points or, in the mixed-integer case, a set of disjoint half-lines or line segments to find an optimal solution. Thus, unlike linear programming where, due to the convexity of the problem, the fact that any local solution is a global optimum can be exploited, integer programming problems have many local optima and finding a global optimum to the problem requires one to prove that a particular solution dominates all feasible points by arguments other than the calculus-based derivative approaches of convex programming.

There are, at least, three different approaches for solving integer programming problems, although they are frequently combined into hybrid solution procedures in computational practice. They are enumerative techniques, relaxation and decomposition techniques, and cutting planes approaches based on polyhedral combinatorics.

The simplest approach to solving a pure integer programming problem is to enumerate all finitely many possibilities. However, due to the combinatorial explosion resulting from the parameter size, only the smallest instances can be solved by such an approach. Sometimes one can implicitly eliminate many possibilities by domination or feasibility arguments. Besides straight-forward or implicit enumeration, the most commonly used enumerative approach is called branch and bound, where the branching refers to the enumeration part of the solution technique and bounding refers to the fathoming of possible solutions by comparison to a known upper or lower bound on the solution value [132]. To obtain an upper bound on the maximization problem, the problem is relaxed in a way which makes the solution to the relaxed problem, relatively easy to solve. A variety of strategies that have been used within the general branch and bound framework is described in [133].

An alternative approach to the solution of integer programming problems is to take a set of complicating constraints into the objective function in a Lagrangian fashion with fixed multipliers that are changed iteratively. This approach is known as Lagrangian relaxation. By removing the complicating constraints from the constraint set, the resulting sub-problem is frequently considerably easier to solve.
Both the size and the complexity of the problems solved decreased considerably when polyhedral theory, developed over the past twenty five years, was applied to numerical problem solving. The underlying idea of polyhedral combinatorics is to replace the constraint set of an integer programming problem by an alternative convexification of the feasible points and extreme rays of the problem. H. Weyl established the fact that a convex polyhedron can alternatively be defined as the intersection of finitely many half spaces or as the convex hull plus the conical hull of some finite number of vectors or points [134]. If the data of the original problem formulation are rational numbers, then Weyl's theorem implies the existence of a finite system of linear inequalities whose solution set coincides with the convex hull of the mixed-integer points. Thus, if we can list the set of linear inequalities that completely defines the convexification of the solution set, then we can solve the integer programming problem by linear programming. [135] derived a cutting plane algorithm for integer programming problems.

Another solution approach for combinatorial and integer problems is that of heuristic methods which obtain good but not necessarily optimal solutions quickly and, in general, without any guarantee as to their closeness to an optimal solution. Heuristics are, however, important for a variety of reasons. They may provide the only usable solution to very difficult optimisation problems for which the current exact algorithms are incapable of providing an optimal solution in reasonable times; when heuristics are used within an exact algorithm, they provide a bound to fix variables and to fathom branches on a search-tree.

Recent research into heuristic algorithms has applied techniques from the physical sciences to the approximate solution of combinatorial problems. For surveys of research in simulated annealing (based on the physical properties of heat), genetic algorithms (based on properties of natural mutation) and neural networks (models of brain function), see [136, 137 and 138], respectively. [139] has generalized some of the attributes of these methods into a method called tabu-search. Worst-case and probabilistic analysis of heuristics are discussed in [140, 141 and 142]. A more recent area of research relates issues in computational logic to those associated with combinatorial optimisation [143]. For general textbooks on integer programming and related topics see [144, 145, 146, 147, 148 and 149].

Another important topic related to combinatorial and integer programming is the complexity of integer programming problems [150]. In the next section a concise introduction of computation complexity theory is given.
6.3 Computational Complexity

Computational problems can vary tremendously in the effort required to solve them precisely. Computational complexity studies the efficiency of algorithms and the inherent difficulty of problems of practical and theoretical importance. Complexity theory deals with the resources required during computation to solve a given problem. The most common resources are time and space which are measured by how many steps a solution algorithm takes to solve a problem and how much memory it requires to solve the problem.

6.3.1 Measuring the efficiency of algorithms

In this section the criteria to assess the performance of algorithms proposed to solve optimisation problems is explained.

A useful measure of algorithm performance is the rate of growth of the time or space required to solve a problem as a function of the problem size.

Size of a problem is the length of the data string necessary to specify the instance of a problem. When an algorithm is run, it behaves differently depending on the data it has to process. The more data that is processed, the longer the algorithm takes to run. To accurately measure this quantity, the notion of the size of a problem must be defined.

The size of the problem can be related to the amount of data entered, the number of results returned, or the length of a particular file that is being processed.

- In a program in which the user enters some integers and the program returns the median among those integers, the size of the problem is the number of integers entered
- In a program in which the user requests that the fibonacci number be printed, the size of the problem is the number of fibonacci numbers printed
- In a program, which sorts a file, the size of the problem is the size of the file

Technically, the size of the problem is the number of bits that pass through the input and output of the algorithm.

A standardized measure of time complexity of an algorithm is the number of elementary operations such as additions and comparisons needed to obtain an optimal solution as a function of the problem size.

An algorithm, which needs $5n^3 + 100n^2 + 500$ steps, is referred to as a polynomial time algorithm. It is guaranteed to terminate within a number of steps which is a polynomial function of the size of the problem. The primary interest is in the rate of growth as $n$ increases for distinguishing between mild and exploding growth rates. Therefore, differences between $5n^3$ and $n^3$ are not really important and lower order terms can be discarded because at large problem sizes the highest degree term...
determines the rate of growth. So the complexity of this algorithm can be sufficiently described by \( n^3 \) and formally it is denoted by "O\((n^3)\)" which is read as "of order \((n^3)\)". An algorithm with complexity \( O(3^n) \) is referred to as an exponential time algorithm. It is guaranteed to terminate within a number of steps which is an exponential function of the size of the problem.

Order notation, also known as Big "O" notation, is a measure of the running time of an algorithm, as it relates to the size of the input to that algorithm. It is intended, not to measure the performance of the machine on which the algorithm is run, but rather to strictly measure the performance of the algorithm itself. Thus, since different machines can vary in their speeds by some constant factor, all constant factors are removed from consideration in order notation. \( O(2) \) and \( O(1) \) are considered to be the same. Similarly, \( O(n) \) is the same as \( O(2n) \).

Note that for sufficiently large \( n \): \( \log n < n < n \log n < n^2 < n^3 < 2^n < n! \)

Consider a problem of scheduling \( n \) jobs that has to be processed at a machine with minimum total waiting time for the jobs and two proposed solution algorithms:

- \( A_1 \) lists the \( n \) processing times in decreasing order and assigns this list to the machine.
- \( A_2 \) explicitly enumerates all possible solutions.

\( A_1 \) is a good heuristic algorithm that finds the optimal solution with a time complexity of \( O(n \log n) \). On the other hand, \( A_2 \) has a time complexity \( O(n!) \).

If a computer is able to make 1000 operations per second, then \( A_2 \) would find an optimal schedule for 100 jobs in \( 2.9594 \times 10^{147} \) years, whereas \( A_1 \) would only take 0.4605 seconds to find the same solution. This is an important issue, because even a radical increase in computing power would not make a big difference in the performance of a bad algorithm.

**Some polynomial running times**

**\( O(1) \)**

An algorithm of order \( O(1) \) has constant running time regardless of the size of the input. The algorithm will never perform more than a certain number of steps, no matter how large the input gets. Although constant time is the best running time an algorithm can have, that algorithm could still be considered bad if the total amount of time to run the algorithm were too large, perhaps because there were many complex or unnecessary steps in the algorithm.
**$O(n)$**

An algorithm of order $O(n)$ has a linear running time proportional to the size of the input. The algorithm never performs more than certain number of steps for each element in the input.

**$O(n^2)$**

An algorithm of order $O(n^2)$ has quadratic running time which increases by a factor of $n^2$.

**$O(\log n)$**

An algorithm of order $O(\log n)$ has logarithmic running time which increases by a factor of the logarithm of the size of the problem. This running time is better than $O(n)$, but not as good as $O(1)$. As the input size gets large, however, the behaviour becomes comparable to $O(1)$ in many circumstances.

**$O(n \log n)$**

An algorithm of order $O(n \log n)$ has a running time which increases in proportion to the size of the input times the logarithm of the size of the input. This running time is better than $O(n^2)$ but not quite as good as $O(n)$.

**Almost constant running times**

**$O(\log^* n)$**

This running time is called log-star time. The log-star function calculates how many times you would need to take the log of $n$ before you would go below 2. For example: $\log^*4 = 2$, $\log^*65536=4$. This function grows so slowly that it may be considered constant. Technically it is not constant, but no computer in the world can store enough data to cause the total running time to increase more than a factor of 5 of the total running time for an algorithm with an input of 2 elements.

**$O(\alpha(m,n))$**

This function is called the inverse of Ackerman's function. It performs similar to the log-star function. For $m=2$ it is equivalent to the log-star, for $m>2$ it grows even slower.

**Worse than polynomial running times**

**$O(2^n)$**

An algorithm of order $O(2^n)$ is said to be exponential which means its running time will double every time you add another element to the input. An algorithm with this running time is too slow to be useful for anything but the smallest of problems. An $O(2^n)$ algorithm which takes an input with 30 elements may need to perform over 1 billion steps. For an input of 40 elements, 1 trillion steps may be necessary which is not possible to process in a reasonable time with any computing power available in the world.
**O(n!)**
An algorithm of order $O(n!)$ has factorial running time which is worse than exponential. An algorithm with this running time and 8 elements in its input, needs a number of steps proportional to 40320. When the input size reaches 15, the number of steps exceeds 1 trillion.

**O(n^n)**
This running time is even worse than factorial. An algorithm with this running time and 10 input elements needs to perform over 10 billion steps.

Algorithms of order $O(g(n))$ where $g(n)$ is either a polynomial or a function bounded by a polynomial are practical and in a much better position to exploit increases in computers speed. The effect of improved technology is multiplicative in polynomial-time algorithms and only additive in exponential-time algorithms. The situation is much worse if complexities involve factorials.

<table>
<thead>
<tr>
<th>Time complexity function</th>
<th>Size of Largest Problem Instance solvable in 1 Hour</th>
<th>With present computer</th>
<th>With computer 100 times faster</th>
<th>With computer 1000 times faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>$N_1$</td>
<td>$100N_1$</td>
<td>$1000N_1$</td>
<td></td>
</tr>
<tr>
<td>$N^2$</td>
<td>$N_2$</td>
<td>$10N_2$</td>
<td>$31.6N_2$</td>
<td></td>
</tr>
<tr>
<td>$N^3$</td>
<td>$N_3$</td>
<td>$4.64N_3$</td>
<td>$10N_3$</td>
<td></td>
</tr>
<tr>
<td>$2^n$</td>
<td>$N_4$</td>
<td>$N_4+6.64$</td>
<td>$N_4+9.97$</td>
<td></td>
</tr>
<tr>
<td>$3^n$</td>
<td>$N_5$</td>
<td>$N_5+4.19$</td>
<td>$N_5+6.29$</td>
<td></td>
</tr>
</tbody>
</table>

Polynomial time algorithms, once discovered, undergo a series of improvements, which result in reductions in the constants in their complexity functions and the degree itself. This is why complexities like $O(n^{80})$ or $O(10^{100}n)$ do not appear in practice. The real breakthrough for the solution of a problem is in finding the first polynomial time algorithm. For combinatorial problems, the jump to the polynomial class usually requires a deep insight into the nature of the problem if this jump can be made at all.

There are exceptions to these rules. For example, the simplex method for linear programming has always worked very well in practice, although it is provably exponential. It might be that the method captures some significant property of the problem that has yet to be discovered. In general, however, a problem can be considered efficiently solved when a polynomial time algorithm has been found for it.
6.3.2 Recognition problems

Recognition problems are a special class of optimisation problems which require only a yes/no answer. They are also referred to as feasibility or decision problems. The theory of Computational Complexity restricts attention to these problems for uniformity. However, the results can easily be generalized, since for every optimisation problem, there is a recognition version. Any complexity results for the recognition version also hold for the original problem. The original problem is not much harder than the recognition version.

A decision problem is equivalent to a language, which is a set of finite-length strings. For a given decision problem, the equivalent language is the set of all strings for which the answer is YES.

Given an optimisation problem, the corresponding recognition problem, which requires a yes or no answer, can be constructed. In the case of a minimization problem, \( \min\{f(x)\} \), the recognition problem is asking for the existence of a solution \( x \) such that \( f(x) \leq z \) for some threshold value \( z \). If the answers to recognition problem are known, then any reasonable minimization problem can be solved using this information polynomial number of times. If \( a \) is the lower bound and \( b \) is the finite upper bound for the optimum value of the objective function and the answer to \( a \) is "no" and to \( b \) is "yes", then \( u = b - a \) would be the initial interval of uncertainty. By using a bisection search in the interval of uncertainty, the feasibility of \( z = a + u/2 \) can be checked. Depending on a yes or a no answer, the new interval of uncertainty will be either \([a, a + u/2]\) or \([a + u/2, b]\). Thus, the previous interval of uncertainty is reduced by a factor of 2 in one iteration. After performing \( m \) bisections, the interval of uncertainty is reduced by a factor of \( 2^m \). If the requested final interval of uncertainty is requested to be of length 1, then \( m \) bisections have to be performed, where \( m \) is given by:

\[
u / 2^m = 1 \Rightarrow m = \log u.
\]

The quantity, \( \log u = \log(b-a) \), although may be large, is not a function of the problem size. Thus, the optimal solution can be obtained after \( \log u \) times using the recognition problem. Therefore, for the purposes of computational complexity, it is sufficient to be concerned with the recognition problems.

6.3.3 Non-deterministic algorithms

A non-deterministic algorithm is a theoretical tool that does not exist in reality. Some say such features will sometime be possible with quantum computers [151]. It is like an ordinary algorithm, except that we are allowed to use the following impossible instruction:
goto both label 1, label 2

This divides the computation into two parallel processes in a single step on the same processor (nothing to do with parallel processing).

To get an idea of the power of this non-deterministic computing, consider the recognition version of 0-1 Integer Programming problem which is a hard problem. No polynomial algorithm is known for it.

Given an integer matrix $A_{m \times n}$ and an integer vector $b_m$, is there a vector $x_n$, with elements 0 or 1, such that $Ax=b$?

This powerful instruction allows to solve the problem in polynomial time, just by exploring all possible combinations of values for the $x$ vector elements:

\begin{verbatim}
begin
  for j=1,...,n
    goto both A, B
    A: $x_j=0$
goto again
    B: $x_j=1$
  :again
  next j
  if $x=(x_1,...,x_n)$ satisfies $Ax=b$ then output "yes" else output "no"
end
\end{verbatim}

In fact, if we had computers capable of executing this extra instruction, there would be no need for Computational Complexity theory.

6.3.4 Problem Classes

We are now in a position to define formally important classes of recognition problems. The following definitions are used to classify problems according to their inherent complexity.

**Class P**

A recognition problem belongs to Class P if, for any instance of the problem, a yes or a no answer can be determined by a polynomial algorithm.
Class NP

Decision problems for which answers can be checked by an algorithm whose run time is polynomial in the size of the input belong to the NP complexity class. This does not require or imply that an answer can be found quickly, only that any claimed solution can be verified quickly. NP stands for Non-deterministic Polynomial, because of an alternative and equivalent definition based on the notion of non-deterministic algorithms but not for Non-Polynomial.

The information needed to verify a positive answer is referred to as a certificate. So given the right certificates, positive answers to a problem in class NP can be verified in polynomial time.

The recognition version of the scheduling of two parallel machines problem is a member of Class NP. If a list of jobs assigned to each machine is given, and a threshold value z, it is possible to check in polynomial time if maximum of the sums of processing times in each machine is less than or equal to z. How the list of the assignment of jobs is obtained is irrelevant. On the other hand, whether this problem is also in P or not is not known, because no polynomial algorithm, which finds the correct assignment list, has yet been found. In other words for the scheduling problem $P_2||C_{\text{max}}$ verifying an answer is simpler than finding the sequence.

Class NP consists therefore of all reasonable problems of practical and/or theoretical importance. For problems not in NP, even verifying that a solution is valid can be extremely difficult.

It follows from the definitions that Class P is a subset of Class NP, i.e. $P \subseteq \text{NP}$. The important question is whether P is a proper subset of NP, i.e. $P \subset \text{NP}$, which means not all reasonable problems can be solved efficiently. Although no formal proof exists yet, there is strong evidence for the validity of this hypothesis.

NP-Complete

NP-Complete is the complexity class of decision problems for which answers can be checked for correctness, given a certificate, by an algorithm whose run time is polynomial in the size of the input (i.e. it belongs to class NP) and no other NP problem is more than a polynomial factor harder.

An NP-complete problem has the following most important property:

If there is an efficient (i.e. polynomial) algorithm for some NP-complete problem, then there is an efficient algorithm for every problem in NP

Because, if $P^*$ is this problem:
• all problems in NP polynomially transform to P* and since polynomial transformability is a special case of polynomial reducibility
• (all problems in NP reduce to P*) and (P* is "easy") \(\Rightarrow\) all problems in NP are easy

Hence, NP-complete problems are the main targets in the search for algorithmic efficiency. But no algorithm that solves an NP complete problem is developed yet.

The technical term for a hard problem is NP-complete which essentially means: abandon all hope of finding an efficient algorithm for the exact solution of this problem. Proving or knowing that a problem is NP-complete is not at all that negative of course. Knowing such limitations, experts do not waste time in impossible projects and instead turn to less ambitious approaches, for example to find approximate solutions (i.e. a heuristic algorithm), to solve special cases or to alter the problem a little so that it becomes tractable even at the loss of some fit to the real-life situation. The goal of this theory is therefore to assist algorithm designers in directing their efforts towards promising areas and avoid impossible tasks.

A problem \(p_1\) is reducible to another problem \(p_2\) if for any instance of \(p_1\), an instance of \(p_2\) can be constructed in polynomial time such that solving the instance of \(p_2\) will solve the instance of \(p_1\) as well. The reducibility of \(p_1\) to \(p_2\) implies that \(p_1\) can be considered as a special case of \(p_2\).

To show that a problem in NP is NP-complete, it is sufficient to show that some other problem already known to be NP-complete polynomially transforms to it. Polynomial transformability is transitive which implies that all other problems in NP polynomially transform also to \(A\). Therefore \(A\) is also NP-complete. In order to use this method of proof, some first NP-complete problem is needed to start with. For this reason, if a polynomial time algorithm can be found for any NP-Complete problem, then every problem in Class NP can be solved in polynomial time, and thus proving P = NP. But this is very unlikely. NP contains some very difficult combinatorial problems which have received considerable amount of research without finding any polynomial time algorithm for solving them.

The best guide to the theory of NP-Completeness is the book by Garey and Johnson, Computers and Intractability [150], which contains a list of NP-Complete problems. The online source for NP-Complete machine scheduling problems is Bruckner & Knust's http://www.mathematik.uni-osnabrueck.de/research/OR/class/

A current list of research report, surveys, and books on computational complexity can be found at http://www.eccc.uni-trier.de/eccc/
NP-Hard

When a decision version of a combinatorial optimisation problem is proven to belong to the class of NP-complete problems, which includes well-known problems such as travelling salesman, bin packing, etc., the optimization version is NP-hard.

If there exists a polynomial time algorithm for the problem P, then it is called "NP hard in the ordinary sense and the algorithm is called pseudo-polynomial if they exist only in certain coding. Class of problems without polynomial time algorithms even when all numbers in the input are bounded by some polynomial in the length of the input are called "NP hard in the strongest sense" or "Strongly NP hard".

The P=NP Question

In essence, the P = NP question asks: if positive solutions to a yes/no problem can be verified quickly, can the answers also be computed quickly? This is the most important open question in theoretical computer science. Most people think that the answer is probably "no"; some people believe the question may be undecidable from the currently accepted axioms.

The NP-complete problems are the toughest problems in NP in the sense that they are the ones most likely not to be in P. This means that if a single NP-complete problem could be shown to be in P, then it would follow that P = NP. Unfortunately, many important problems have been shown to be NP-complete and not a single fast algorithm for any of them is known. No one knows whether polynomial-time algorithms exist for NP-complete problems.

Although whether P=NP or not is not known, other problems outside both P and NP are known. The problem of finding the best move in Chess is EXPTIME-Complete, which means it requires exponential time, and so is outside P and NP. There are many other problems such as Presburger arithmetic and the halting problem which are even harder.

It is usually assumed that P means easy and not in P means hard. While this is a reasonably accurate assumption in complexity theory, it is not always true in practice for several reasons:

- It ignores constant factors. A problem that takes time $10^{1000}$ is P (in fact, it's linear time), but is completely intractable in practice. A problem that takes time $10^{-1000}2^n$ is not P (in fact, it's exponential time), but is very tractable for values of n up into the thousands.
- It ignores the size of the exponents. A problem with time $n^{1000}$ is P, yet intractable. A problem with time $2^{n/1000}$ is not P, yet is tractable for n up into the thousands.
- It only considers worst-case times. There might be a problem that arises in the real world. Most of the time, it can be solved in time n, but rarely an instance of
the problem might take $2^n$ time. This problem might have an average time that is polynomial, but the worst case is exponential, so the problem wouldn't be in P.

- It only considers deterministic solutions. There might be a problem that you can solve quickly if you accept a tiny error probability, but a guaranteed correct answer is much harder to get. The problem would not belong to P even though in practice it can be solved fast. This is in fact a common approach to attack NP-complete problems.

- New computing models such as quantum computers, which also work probabilistically, may be able to quickly solve some problems not known to be in P.

### Famous Complexity Classes

In Table 15, most of the classes of problems considered in complexity theory are listed along with rough definitions.

- **P**: Solvable in polynomial time
- **NP**: YES answers checkable in polynomial time
- **Co-NP**: NO answers checkable in polynomial time
- **NP-complete**: The hardest problems in NP
- **Co-NP-complete**: The hardest problems in Co-NP
- **NP-hard**: Either NP-complete or harder
- **NP-easy**: non-decision-problem analogue to NP
- **NP-equivalent**: non-decision-problem analogue to NP-complete
- **#P**: Count solutions to an NP problem
- **#P-complete**: The hardest problems in #P
- **NC**: Solvable efficiently on parallel computers
- **P-complete**: The hardest problems in P to solve on parallel computers
- **PSPACE**: Solvable with polynomial memory and unlimited time
- **PSPACE-complete**: The hardest problems in PSPACE
- **EXPTIME**: Solvable with exponential time
- **EXPSPACE**: Solvable with exponential memory and unlimited time
- **BQP**: Solvable in polynomial time on a quantum computer (answer is probably right)
- **BPP**: Solvable in polynomial time by randomised algorithms (answer is probably right)
- **RP**: Solvable in polynomial time by randomised algorithms (NO answer is probably right, YES is certainly right)
- **ZPP**: Solvable by randomised algorithms (answer is always right, average running time is polynomial)

Table 15. Famous complexity classes
6.4 Summary

In this chapter, various important concepts on optimisation and computational complexity are introduced. In order to formulate the problem of scheduling the soaking pits/rolling mill process, to understand the challenges and evaluate the applicability and efficiency of the proposed solutions, it is necessary to have the understanding of these concepts. The background provided in this chapter is used in discussing the SPO system in chapter 9.
7 Machine Scheduling

This chapter explains machine scheduling problems in general. It is aimed at providing adequate background on the subject so that one can formulate a particular machine-scheduling problem and can identify where it stands in this diverse field. With the information provided in this chapter the reader is supplied with enough knowledge to understand the literature on this subject. Of course the chapter is written with a focus on the soaking pits/rolling mill optimisation problem and it is referred to when the emphasis is seen as useful.

The chapter starts with an introduction and follows with the notation and definitions used in the machine scheduling field. Then a classification notation that captures the structure of most deterministic models considered in the literature is presented. And finally the chapter concludes with a discussion of the computational complexity of machine scheduling algorithms. The reader is recommended to read section 6.3 before reading section 7.4 of this chapter.

7.1 Introduction

Scheduling problems can be understood as allocating limited resources over time to perform a set of tasks that are parts of some processes. The goal is to optimise one or more objectives subject to various constraints. Tasks individually compete for resources, which can be of a variety of nature like manpower, money, machines, energy, tools, etc. The same is true for task characteristics like ready times, due dates, relative urgency weights, etc. The objectives can also take many forms. For example, one objective may be the minimization of the completion time of the last task, and another may be the minimization of the number of tasks completed after their respective due dates.

Consider a manufacturing system in which a set of jobs, each requiring a sequence of operations, is to be performed by using a number of machining centres. To perform a job, each of its operations must be processed in the order given by the sequence of operations. The processing of an operation requires the use of a particular machining centre for a given duration. Given a criterion to measure the quality of each possible schedule such as maximum throughput, the target of scheduling is to find a preferably optimum processing order on each machine and to specify the exact starting and finishing times of each operation. This example is very similar to what SPO aims to do.

In the current competitive environment effective sequencing and scheduling has become a necessity for survival in the marketplace. Companies have to meet shipping
dates that have been committed to customers. They also have to schedule activities in such a way as to use the resources available in an efficient manner.

Scheduling theory arose originally in an industrial production context. However, various other interpretations are possible. Jobs and machines can stand for patients and hospital equipment, classes and teachers, ships and dockyards, dinners and cooks, programs and computers, take-offs and landings at an airport and runways, stages in a construction project and construction crews, or cities and salesmen. Each of these situations falls within the scope of scheduling theory and algorithms.

Scheduling theory is concerned primarily with the development and analysis of mathematical models. Development of such models leads to solution techniques and practical insights. The theoretical perspective is largely a quantitative approach that attempts to capture problem structure in mathematical formulation. This quantitative approach begins with a description of resources and tasks and with the translation of the decision making goal into an explicit objective function. The purpose of modelling is to find optimal or suboptimal schedules in the sense of a given criterion, by applying best suited algorithms. These schedules are then used for original setting to carry out the various activities.

Scheduling problems are in general of an optimising nature. However, some of them are formulated as decision problems. An example is scheduling to meet deadlines, i.e. given a set of deadlines, find a schedule with no late tasks. With every optimisation problem, a decision problem can be associated. For example, the decision problem corresponding to minimization of makespan is whether there exists a schedule with makespan less than the given value. Both optimisation and decision cases are analysed in the same way when complexity issues are considered. A polynomial time algorithm for the decision case implies a polynomial time algorithm for the optimisation case and if a polynomial time algorithm does not exist for the decision case then it does not exist for the optimisation case either.

In scheduling terminology, a distinction is made among a sequence, a schedule, and a scheduling policy. A sequence corresponds to a permutation of the jobs or the order in which jobs are to be processed on a given machine. A schedule refers to an allocation of jobs within a more complicated setting of machines, allowing possibly for preemptions of jobs by other jobs that are released at later points in time. It covers timing of uniquely identified orders. The concept of a scheduling policy is often used in stochastic settings. A policy prescribes an appropriate action for any of the states the system may be in. In deterministic models, usually only sequences or schedules are of importance.
A feasible sequence corresponds to an ordering of the operations on each machine that allows the sequence of operations corresponding to each job to be processed in the proper order. Corresponding to each feasible sequence there is an infinity of feasible schedules obtained by further specifying the exact starting time and finishing time of each operation. A feasible schedule does not violate any of the constraints. An infeasible schedule is a schedule that violates one or more constraints. An optimal schedule is the best schedule among all feasible schedules in terms of the objectives.

It may not be immediately clear what impact schedules have on given objectives. But the choice of schedule does have a major impact on system performance. A detailed schedule of tasks to be performed helps maintain efficiency and control of operations. So it is logical to invest time and effort searching for a good schedule rather than just choosing a schedule at random. A scheduling algorithm is an algorithm that constructs a schedule for a given problem. In general, optimisation algorithms are sought after, but because of the inherent complexity of many problems, approximation or heuristic algorithms are used too. In most practical situations a scheduling problem can be considered to be satisfactorily solved if a good, not necessarily optimal solution has been found. The complexity results confirm that the search for an optimal schedule would often be too time consuming in any case. However, even in the construction of heuristic solution methods a study of optimising methods may contribute valuable insights and lead to better heuristics than the very general ones.

Deterministic scheduling problems are those in which no variable with a non-deterministic (e.g. probabilistic) description appears. In a manufacturing environment deterministic scheduling is also known as predictive. Its complement is reactive scheduling, which can also be regarded as deterministic scheduling with a shorter planning horizon. The problems that have probabilistic variables are classed as stochastic scheduling problems.

In scheduling problems classed as static, all characteristics of the set of tasks and the set of resources are known in advance. Whereas dynamic problems have some parameters such as ready times that are unknown in advance. Scheduling problems in practice are never static, because the input data continuously change. For example, the weight of job j, which is very hard to measure to begin with, may not be constant, but instead may be time dependent. A job that is not important one day may suddenly become important next day.

Scheduling problems are classified as open shop and closed shop according to their requirements. Open shop takes orders from customers; closed shop is satisfied for inventory.
Although the growth of scheduling literature has been spectacular since Johnson's pioneering work in 1954 [152], relatively fewer applications of scheduling theory have been reported. Scheduling can be difficult from a technical as well as an implementation point of view. The difficulties encountered on the technical side are similar to the difficulties encountered in other forms of combinatorial optimisation and stochastic modelling. The difficulties on the implementation may depend on the accuracy of the model used for the analysis of the actual scheduling problem and on the reliability of the input data required. Various assumptions underlie the problem formulation implicitly. Particularly vital among these is the restriction to deterministic problems, in which stochastic aspects are ignored, and restriction to machines of capacity one. A further restriction on the applicability of scheduling theory arises out of the choice of specific cost functions as optimality criteria.

Scheduling in a production system or service organization must interact with many other functions. These interactions are system dependent and may differ from one situation to another. They often take place within an enterprise-wide information system. The decision hierarchy takes place in the order:

Planning->Scheduling->Control

A modern factory often has an information system in place that includes a central computer and database. Local area networks of personal computers, workstations, and data entry terminals are connected to this central computer, which may be used either to retrieve data from the database or enter new data. The software controlling such an elaborate information system is typically referred to as an enterprise resource planning (ERP) system. Such an ERP system plays the role of an information highway that traverses the enterprise with, at all organizational levels, links to decision support systems. Scheduling is often done in interactively with decision support system that is installed on a personal computer or workstation that is linked to ERP system. Terminals at key locations linked to ERP system can give departments throughout the enterprise access to all current scheduling information. These department, in turn can provide the scheduling system with up-to-date information concerning the statues of jobs and machines. There are, of course, also many other environments in which the communication between the scheduling function and other decision-making entities occur in meeting or through memos. Figure 67 from [153] depicts a diagram of the information flow in a manufacturing system.

The shop floor is not the only part of the organization that impacts the scheduling process. It is also affected by the production planning process that handles medium to long term planning for the entire organization. This process attempts to optimise the firm's overall product mix and long-term resource allocation based on its inventory
levels, demand forecasts, and resource requirements. Decisions made at this higher planning level may impact the scheduling process directly. In manufacturing, the scheduling function has to interact with other decision-making functions within the plant. One popular system that is widely used is the material requirements planning (MRP) system. After a schedule has been generated, it is necessary that all raw materials and resources are available at the specified times. The ready dates of all jobs have to be determined jointly by the production planning and scheduling system and the MRP system. MRP systems are normally fairly elaborate. Each job has a bill of materials (BOM) itemizing the parts required for production. The MRP system keeps track of the inventory of each part. Furthermore, it determines the timing of the purchases of each one of the materials. In doing so, it uses techniques such as lot sizing and lot scheduling that are similar to those used in scheduling systems. In the cases where the facility does not have a scheduling system, the MRP system may be used for production planning purposes. However, in complex settings, it is not easy for an MRP system to do the detailed scheduling satisfactorily.

Scheduling began to be taken seriously in manufacturing at the beginning of this century with the work of Henry Gantt and other pioneers. However, it took many years for the first scheduling publications to appear in the industrial engineering and operations research literature. Some of the first publications appeared in Naval Research Logistics Quarterly in the early 1950s and contained results by W.E. Smith, S.M. Johnson, and J.R. Jackson. During the 1960s a significant amount of work was done on dynamic programming and integer programming formulations of scheduling problems. After Richard Karp's famous paper on complexity theory, [154], the research in the 1970s focused mainly on the complexity hierarchy of scheduling

![Diagram of the information flow in a manufacturing system.](image-url)
problems. In the 1980s several different directions were pursued in academia and industry with increasing amount of attention being paid to stochastic scheduling problems. Also, as personal computers started to permeate manufacturing facilities, scheduling systems were being developed for the generation of usable schedules in practice. This system design and development was, and is, being done by computer scientists, operations researchers and industrial engineers.

Over the last four decades, many books have appeared that focus on sequencing and scheduling. Three most recent ones, [153, 155 and 156], provide comprehensive information on the subject. The book by Pinedo deals with combinatorial problems of deterministic scheduling, stochastic models, scheduling in practice and popular heuristics and also covers system design and development issues. The book by Blazewicz et al. deals primarily with the computational aspects of deterministic scheduling models and their applications to manufacturing. And the book by Pinedo and Chao is more application oriented and describes a number of different scheduling models for problems arising in manufacturing as well as services.

7.2 Notation and Definitions

The focus of this chapter is on deterministic scheduling of manufacturing processes. Because scheduling of soaking pits and rolling mill system is also a deterministic scheduling problem. There are many terms related to scheduling problems. In order to understand the scheduling theory developed until now, it is important to clarify the meanings of these terms first. In this section most of these terms are explained and a notation used to characterize and classify the variety of problem instances is described.

In general, scheduling problems are characterised by three sets:

\[ T = \{T_1, T_2, \ldots, T_n\} \] set of n tasks,

\[ M = \{M_1, M_2, \ldots, M_m\} \] set of m machines,

\[ R = \{R_1, R_2, \ldots, R_s\} \] set of s additional resources,

Scheduling means to assign machines from M and possibly resources R to assign to tasks from T in order to complete all the tasks under the imposed constraints. The scheduling problem can be defined as a set of parameters together with an optimality criterion. These parameters and optimality measures are described in the rest of this section. An instance of the scheduling problem is obtained by specifying particular values for all the problem parameters.

An entity that has to be processed on a machine is typically referred to as a job. A job can be made up of any number of tasks. If a job is thought of as making a product,
then each task is an activity that contributes to making that product. A task is an
operation of the job. A job usually has only a single task. The exceptions are the cases
of job shop and flow shop where a job is broken down into tasks that are completed
on dedicated machines.

A machine is available to execute jobs and tasks. A machine can represent a hotel
room, a car, a pier, a runway, an operating room, and so on.

An additional resource means a facility besides machines which the tasks to be
performed compete for. The competition aspect in this definition should be stressed,
since facilities dedicated to only one task are not treated as resources. In
manufacturing environments tools, material, transport facilities, etc can be treated as
additional resources.

In scheduling problems, the number of jobs and machines are assumed to be finite.
The number of jobs is denoted by n and the number of machines by m. Usually, the
subscript j refers to a job, and the subscript i refers to a machine. If a job requires a
number of processing steps or operations, then the pair (i,j) refers to the processing
step or operation of job j on machine i.

There are two general constraints in classical scheduling theory. Each task is to be
processed by at most one processor at a time plus possibly specified amounts of
additional resources and each processor is capable of processing at most one task at a
time.

A schedule is an assignment of machines from set M and possibly resources from set
R to tasks from set T in time such that the following conditions are satisfied:

- at every moment each processor is assigned to at most one task and each task is
  processed by at most one processor.
- task $T_j$ is processed in time interval $[r_j, \infty)$,
- all tasks are completed,
- if tasks $T_i$, $T_j$ are in relation $T_i < T_j$, the processing of $T_j$ is not started before $T_i$ is
  completed.
- in the case of non-preemptive scheduling no task is preempted (then the schedule
  is called non-preemptive), otherwise the number of preemptions of each task is
  finite (then the schedule is called preemptive),
- resource constraints, if any, are satisfied.
7.2.1 Task Characteristics

In general, task $T_j \in T$ is characterized by the following data: processing time, arrival time, due date, deadline, weight which are denoted by $p_j$, $r_j$, $d_j$, $\bar{d}_j$, and $w_j$ respectively.

We assume that all these parameters, $p_j$, $r_j$, $d_j$, $\bar{d}_j$, and $w_j$, are integers. This assumption is not very restrictive, since it is equivalent to permitting arbitrary rational values. We assume moreover, that tasks are assigned to all resources whenever they start or resume their processing and that they release all the assigned resources whenever they are completed or preempted. These assumptions imply that deadlock cannot occur.

Processing Time

Processing time is the length of time to process a job or a task. It is denoted by $p_{ij}$ which means the processing time of $j^{th}$ job on $i^{th}$ machine. The subscript $i$ is omitted if the processing time does not depend on the machine where machines are identical or if the job is only to be processed on one given machine.

$$p_{ij} = p_j, \quad i = 1, 2, \ldots, m$$

If the machines in $M$ are uniform then

$$p_{ij} = \frac{p_j}{b_i}, \quad i = 1, 2, \ldots, m$$

where $p_j$ is the standard processing time usually measured on the slowest processor and $b_i$ is the processing speed factor of processor $M_i$. In case of shop scheduling the vector of processing times describes the processing requirements of particular tasks comprising one job; that is for job $J_j$ we have

$$\mathbf{p}_j = [p_{1j}, p_{2j}, \ldots, p_{nj}]^T$$

where $p_{ij}$ denotes the processing time of $T_{ij}$ on the corresponding machine. $p_j=p$ implies that all processing times are equal.

In many systems, processing times are not known a priori. Despite this, solution of a deterministic scheduling problem may also have an important interpretation in these systems. When the task processing times are not known in the case of scheduling to meet deadlines, one approach is to solve the problem with assumed upper bounds on the processing times. Such a bound for a given task may be implied by the worst-case complexity function of an algorithm connected with that task. Then if all deadlines are met with respect to the upper bounds, no deadline will be exceeded for the real task processing times. Another method is instead of taking exact values of processing times one can take their mean values and, using the procedure described by Coffman
and Denning in [157], calculate an optimistic estimate of the mean value of the schedule length. An alternative approach is to measure the processing times of tasks after processing a task set scheduled according to a certain algorithm A. Taking these values as an input in the deterministic scheduling problem, one may construct an optimal schedule and compare it with the one produced by algorithm A, thus evaluating the latter. Apart from the above, optimisation algorithms for deterministic scheduling problems give some indications for the construction of heuristics weaker assumptions than those made in stochastic scheduling problems, cf. [158]. In the SPO system the processing times of heating and rolling tasks are estimated by ANN models, which are explained in detail in chapter 5.

Release date
Earliest time, at which a job begins to be available for processing. For example, a job may be ready at a later time than time 0 because it has not been completed in the last shop. Release date may also be referred to as release time, arrival time, or ready time. Release date of jth task is denoted by \( r_j \).

If the symbol, \( r_j \), is present in the problem notation field, job j may not start its processing before its release date \( r_j \). If \( r_j \) does not appear in the notation, the processing of job j may start at any time.

In deterministic scheduling theory a priori knowledge of ready times of tasks is usually assumed. Ready times are obviously known in systems working in an offline mode and in control systems in which measurement samples are taken from sensing devices at fixed time moments. In SPO jobs are ready at time \( t=0 \).

Due Date
The due date \( d_j \) of task j represents the committed shipping or completion date, date the job is promised to the customer. It specifies a time limit by which \( T_j \) should be completed. Completion of a job after its due date is allowed, but then a penalty is incurred. Most penalty functions are defined in accordance with due dates. Due dates, in contrast to release dates, are usually not explicitly specified in the classification notation of a scheduling problem. The type of objective function gives sufficient indication whether the jobs have due dates. \( d_j=d \) implies that all due dates are equal.

Deadline
When a due date must be met, it is referred to as deadline and denoted by \( \bar{d}_j \). Deadline is a hard real time limit by which \( T_j \) should be completed.
**Weight**

The weight $w_j$ of task $j$ is basically a priority factor, denoting the relative urgency or importance of task $j$ relative to the other tasks in the system. For example, this weight may represent the actual cost of delaying the task in the system. This cost could be a holding or inventory cost; it also could represent the amount of value already added to the task.

### 7.2.2 Job Characteristics and Penalty Functions

Job processing has many distinctive characteristics and penalty function. In this section, some of the most common characteristics and penalty function are described.

**Completion Time**

Completion time of a job is the time the job exits the system. Its completion time on the last machine which it requires processing. The objective to be minimized for finding the best schedule is always a function of the completion time of the jobs. The completion time of the task of job $j$ on machine $i$ is denoted by $C_{ij}$. $C_j$ denotes the time job $j$ leaves the systems, i.e. the completion time of the last task of job $j$.

**Waiting Time**

Waiting time is the length of time between the ready time of a job and the beginning of processing of a job. It is denoted by "W".

**Slack Time**

Slack time is the difference between the amount of time until a job's due date and the processing time of the job.

**Lateness**

Lateness of a job is the difference between the completion time and the due date of the job. It is defined as:

$$L_j = C_j - d_j,$$

where $C_j$ is the completion time and $d_j$ is the due date of the job $j$.

Lateness is positive when the job is completed late and negative when it is completed early.

**Tardiness**

The tardiness of job $j$, $T_j$, is defined as:

$$T_j = \max(C_j - d_j, 0),$$

where $C_j$ is the completion time and $d_j$ is the due date of the job $j$.

The difference between the tardiness and the lateness is the fact that tardiness is never negative.
Flow Time

Flow time is the amount of time job \( j \) spends in the system. It is defined as:

\[ F_j = C_j - r_j \]

where \( C_j \) is the completion time of the \( j^{th} \) job, and \( r_j \) is the ready time of the \( j^{th} \) job.

Unit Penalty

The unit penalty of job \( j \) is defined as

\[ U_j = \begin{cases} 1 & \text{if } C_j > d_j \\ 0 & \text{otherwise} \end{cases} \]

Like lateness, tardiness and earliness explained below, unit penalty is a due date related penalty function too. Flow time is an example of a penalty function not related to due date.

Earliness

The earliness of job \( j \), \( E_j \), is defined as:

\[ E_j = \max(d_j - C_j, 0) \]

where \( C_j \) is the completion time and \( d_j \) is the due date of the \( j^{th} \) job.

Earliness is nonincreasing in \( C_j \) as opposed to the all other penalty functions explained above. Lateness, tardiness, flow time and unit penalty are all nondecreasing in \( C_1, \ldots, C_n \).

7.2.3 Constraints

Job processing is often subject to constraints that are peculiar. In this section, some of the most common scheduling constraints are described.

Preemptions

Preemptions imply that it is not necessary to keep a job on a machine, once it started, until completion. The scheduler is allowed to interrupt the processing of a job at any point in time and put a different job on the machine instead. The amount of processing a preempted job already has received is not lost. When a preempted job is afterward put back on the machine or on another machine in the case of parallel machines, it only needs the machine for its remaining processing time. If preemption of all the tasks is not allowed, the schedule is called nonpreemptive.

Job Splitting

Jobs that comprise a collection or batch of items may be partially processed on one machine and partially on another machine. Job splitting is similar to preemption. However preemption is more general, since a job can be divided into a number of segments that can be processed simultaneously on several machines in parallel. Another form of job splitting can occur when a job is a batch of items that has to go in a flow shop from one machine to the next. After a part of the batch has been
completed at one stage, it can start its processing on the next machine before the entire batch has been completed on the first machine.

**Precedence Constraints**

In a scheduling problem if a job can start only after a given set of other jobs have been completed, it has precedence constraints. In set $T$ precedence constraints among tasks may be defined as $T_i < T_j$ which means that the processing of $T_i$ must be completed before $T_j$ can be started. The tasks in set $T$ are called dependent if the order of execution of at least two of the tasks are restricted by this relation. Otherwise, the tasks are called independent. A task $T_j$ is called available for processing at time $t$ only if $r_j \leq t$ and all its predecessors with respect to the precedence constraints have been completed by time $t$.

A tasks set with precedence relation is usually represented as a directed graph in which nodes correspond to tasks and arcs to precedence constraints. No transitive arcs exist in precedence graphs. A precedence graph may have a specific structure. It may take the form of a set of chains, or a tree. If each job has at most one predecessor, the constraints are referred as an intree. If each job has at most one predecessor, the constraints are referred to as an outtree.

Precedence constraints may appear in a single machine or parallel machine environment. In the case of dedicated processors, except in open shop systems, tasks that constitute a job are always dependent, but the jobs themselves can be either independent or dependent.

**Machine Eligibility Restrictions**

When the machine environment consists of parallel machines, there may exist machine eligibility restrictions. The $M_j$ symbol in the classification notation of a problem means that not all machines are capable of processing job $j$. The set $M_j$ denotes the set of machines that can process job $j$. If the classification notation does not contain $M_j$, job $j$ may be processed on any one of the machines.

**Routing Constraints**

Routing constraints specify the route a job takes through a system, for example, a flow shop or a job shop. A given job may consist of a number of tasks that must be processed on specified machines in a given sequence. Routing constraints are very common in most manufacturing processes.

**Sequence Dependent Setup Times**

Sequence dependent setup times are denoted by "$s_{jk}$". The symbol $s_{jk}$ in the classification notation of a scheduling problem represents the sequence dependent
setup time between the jobs \( j \) and \( k \). \( s_{0k} \) denotes the setup time for job \( k \) if job \( k \) is first in the sequence and \( s_{j0} \) the cleanup time after job \( j \) if job \( j \) is last in the sequence. Of course, \( s_{0k} \) and \( s_{j0} \) may be zero. If the setup time between jobs \( j \) and \( k \) depends on the machine, then the subscript \( i \) is included (\( s_{ijk} \)). If no \( s_{jk} \) appears in the classification notation of the problem, all setup times are assumed to be zero or sequence independent, in which case they are simply included in the processing times.

**Breakdowns**

Machine breakdowns imply that machines are not continuously available. The periods that a machine is not available are assumed to be fixed due to shifts or scheduled maintenance. If there are a number of identical machines in parallel, the number of machines available at any point in time is a function of time (i.e., \( m(t) \)).

**Permutation**

A constraint that may appear in the flow shop environment is that the queues in front of each machine operate according to the First In First Out (FIFO) discipline. This implies that the order or permutation in which the jobs go through the first machine is maintained throughout the system.

**Blocking**

Blocking is a phenomenon that may occur in flow shops. If a flow shop has a limited buffer in between two successive machines, the upstream machine is not allowed to release a completed job when the buffer is full. This situation is known as blocking. The completed job has to remain on the upstream machine preventing or blocking that machine from working on another job. The most common occurrence of blocking is the case with zero buffers in between any two successive machines. In this case, a job that has completed its processing on a given machine cannot leave a machine if the preceding job has not completed its processing yet on the next machine. Thus the blocked job also prevents or blocks the next job from starting its processing on the given machine.

**No-wait**

The no-wait like permutation and blocking is requirement that may occur in flow shops. Jobs are not allowed to wait between two successive machines. This implies that the starting time of a job at the first machine has to be delayed to ensure that the job can go through the flow shop without having to wait for machine. A steel rolling mill, in which a slab of steel is not allowed to wait as it would cool off during a wait, is a good example of such an operation.
Recirculation

Recirculation may occur in a job shop when a job may visit a machine or work centre more than once.

7.2.4 Machine Environments

There are many important machine configurations. The most basic ones are single machine, parallel machines that perform same functions, and dedicated machines that are specialized for execution of certain machines. Three types of parallel machines are distinguished depending on their speeds: identical, uniform and unrelated. There are three models of processing sets of tasks in the case of dedicated processors: flow shop, open shop and job shop. In the rest of this section these configurations are explained in more detail.

In real world many machine environments are significantly more complicated than the models described here. Nonetheless, these models are so fundamental that their analysis provides insights that are useful in studies of more complicated environments.

Single Machine

Only one machine is available to process jobs. Each job has a single task. Every job is performed on the same machine. The case of a single machine is the simplest of all possible machine environments and is a special case of all other machine environments.

Parallel Machines

In the parallel machines environment, there are multiple machines performing the same functions to process the jobs. A job consists of a single task and can be processed on any one of the machines or on anyone that belongs to a given subset. The machines can be identical, of different speeds, or specialized to only processing specific jobs.

Identical Machines

"Pm" in the classification notation of a scheduling problem, denotes identical parallel machines configuration. It means that there are m identical machines in parallel and they have equal processing speeds. A job j requires a single operation and may be processed on any one of the machines or any one that belongs to a given subset. If job j is not allowed to be processed on just any machine, but rather only on any one belonging to a specific subset $M_j$, then the entry $M_j$ appears in the classification notation too.
Uniform Machines
If the parallel machines differ in their speeds, but the speed of each processor is constant and does not depend on the tasks in \( T \), then they are called uniform machine. A Uniform parallel machines configuration is denoted by "Q". Qm in the classification notation means that there are m machines in parallel with different speeds. The speed of machine i is denoted by \( v_i \). The time \( p_{ij} \) that job j spends on machine i is equal to \( p_j/v_i \) where it is assumed that job j receives all its processing from machine i. If \( v_i=1 \) for all i and \( p_{ij}=p_j \), then the environment is identical to the identical machines configuration.

Unrelated Machines
Unrelated machines environment is a generalization of the uniform parallel machines configuration. If the speeds of the machines depend on the particular task processed, then they are called unrelated. In the classification notation an unrelated machines configuration is denoted by "R". Rm means that there are m different machines in parallel. \( i^{th} \) machine process \( j^{th} \) job at speed \( v_{ij} \). The time \( p_{ij} \) that job j spends on machine i is equal to \( p_j/v_{ij} \) where it is assumed that job j receives all its processing from machine i. If \( v_{ij}=v_i \) for all i and j, then the machine environment becomes identical to the uniform machines configuration.

Dedicated Machines
To describe dedicated machines models more precisely, we assume that tasks form n subsets. Each subset is called a job. A job \( J_j \) is divided into \( n_j \) tasks, \( T_{1j}, T_{2j}, \ldots, T_{nj} \) and two adjacent tasks are to be performed on different processors. A set of jobs will be denoted by J.

In open shop and flow shop configurations the number of tasks is the same for each job and is equal to m, i.e.

\[
n_j = m, \quad j=1,2,\ldots,n
\]

In a general job shop system the number \( n_j \) is arbitrary. Below different configurations of dedicated machines are explained in more detail.

Flow Shop
In many manufacturing environments, jobs consist of multiple tasks that are processed on a number of different machines. If all jobs visit the same machines in the same order, the environment is referred to as a flow shop. In the general flow shop configuration, there are m machines in series and each job has exactly m tasks. The first task of every job is done on machine 1, second task on machine 2 and so on. The sequence of the jobs may vary from machine to machine, since jobs may be resequenced between machines. In some flow shops, if a job does not need processing at a particular machine, it may bypass that machine and go ahead of the
jobs being processed or waiting for processing there. In such cases where not every job has m tasks, the processing times of the tasks that don't exist are zero. The processing time each task spends on a machine varies depending on the job that the task belongs to. The precedence constraint in flow shop configuration requires that for each job, task i-1 on machine i-1 must be completed before task i can begin on machine i.

**Flexible Flow Shop**
A flexible flow shop is a generalization of the flow shop and the parallel machine environments. It consists of a number of stages in series with a number of machines in parallel at each stage. Each job has to be processed first at stage 1, then stage 2, and so on. A stage functions as a bank of parallel machines. At each stage, a job requires processing on only one machine and any machine can do.

**Open Shop**
There are m machines. Each job has to be processed again on each one of the m machines. However, some of these processing times may be zero. There are no restrictions with regard to the routing of each job through the machine environment. The scheduler is allowed to determine a route for each job, and different jobs may have different routes.

**Job Shop**
In a job shop with m machines, each job has its own predetermined route to follow. A job typically consists of a number of tasks that needs processing on different machines. Job shop configuration is a generalization of a flow shop in which each and every job has the same route. Job shops are prevalent in industries that make customized industrial hardware, but they also occur frequently in the service industries.

A distinction is made between job shops in which each job visits each machine at most once and job shops in which job may visit each machine more than once. In the latter case, the classification notation contains the entry recrc for recirculation which significantly increases the complexity of the model.

In the general job shop model, there is a set of machines indexed by k. Jobs are indexed by j, and tasks are indexed by j. Each task on a machine is indicated by a set of three indices: i, the job that the task belongs to, j, the number of the task itself, and k, the machine that this particular task needs to use.

**Flexible Job Shop**
A flexible job shop is a generalization of the job shop and parallel machine environments. Instead of m machines of the job shop, the flexible job shop has m workcentres that have multiple machines in parallel. Each job had its own route to
follow through the shop; job j requires processing at each work centre on only one machine and any machine can do. If a job on its route through the shop may visit a work centre more than once, then it is said to be subject to recirculation. From a combinatorial point of view, the flexible job shop with recirculation is the most complex machine environment.

### 7.2.5 Performance Measures

Many different types of objectives are important in operating scheduling. In practice, the overall objective is often a composite of several basic objectives. In this section, the most important of these basic objectives are explained.

It is useful to make the distinction between regular and irregular performance measures. Regular performance measures are nondecreasing in completion time. Their value is completely determined by the job completion times. But an objective function such as the total weighted earliness plus the total weighted tardiness which is defined as:

\[
\sum_{j=1}^{n} w_j E_j + \sum_{j=1}^{n} w_j T_j
\]

is not regular.

A schedule for which the value of a particular performance measure $\gamma$ is at its minimum is called optimal, and the corresponding value of $\gamma$ is denoted by $\gamma^*$.

**Throughput**

The throughput of a system is equivalent to the output rate. For many facilities maximizing the throughput is very important, and managers are often measured by how well they do so. Throughput is frequently determined by bottleneck machines which are the machines having the lowest capacity relative to their demand. Thus maximizing a facility’s throughput rate is often equivalent to maximizing the throughput rate at these bottlenecks. To achieve this, the scheduler must try to ensure that the bottleneck machine is never idle. This may require always having some jobs in queue waiting for the machine. And also, if there are sequence-dependent setup times on the bottleneck machine, the scheduler has to sequence the jobs in a way that minimizes the sum of the setup times, or equivalently, the average setup time.

**Makespan**

The makespan is important when the number of jobs is finite. The makespan or schedule length is denoted by $C_{\text{max}}$ and is defined as:

\[
C_{\text{max}} = \max\{C_1, \ldots, C_n\}
\]

where $C_j$ is the completion time of job $j$. 
$C_{\text{max}}$ is equivalent to the completion time of the last job to leave the system. Minimizing schedule length leads to both, the maximization of the processor utilization factor within schedule length $C_{\text{max}}$, and minimization of the maximum in processing time of the scheduled set of tasks. The makespan objective is very closely related to the throughput objective. For example, minimizing the makespan in a parallel machine environment with sequence dependent setup times forces the scheduler to balance the load over the various machines and to minimize the sum of all the setup times. Heuristics that tend to minimize the makespan in a machine environment with a finite number of jobs also tend to maximize the throughput rate when the flow of jobs is constant over time.

**Maximum Lateness**

The maximum lateness is defined as:

$$L_{\text{max}} = \max\{L_1, \ldots, L_n\}.$$  

where $L_j$ is the lateness of job $j$.

It measures the worst violation of the due dates. Minimizing maximum lateness is in a sense equivalent to minimizing the worst-case performance of the schedule. Performance measures involving due date are of great importance in manufacturing systems, especially in those that produce to specific customer orders.

Some of the problems involving lateness are:

1. Minimizing the maximum lateness, $L_{\text{max}}$.

2. Minimizing the total lateness. The total lateness is defined as:

$$\sum_{i=1}^{n} L_i$$

3. Minimizing the total weighted lateness. The total weighted lateness is defined as:

$$\sum_{j=1}^{n} w_j L_j$$

where $w_i$ denote the weight assigned to the $j^{th}$ job.

4. Minimizing the mean lateness. The mean lateness is defined as:

$$\frac{1}{n} \sum_{j=1}^{n} L_j$$

**Number of Tardy Jobs**

Another due date related objective is the number of tardy jobs.

$$U = \sum_{j=1}^{n} U_j ,$$

This objective is not concerned with how tardy a job actually is, but rather with weather it is tardy at all. The number of tardy jobs is a statistic that is very easy to track in a database, so managers are often measured by the percentage of on time
shipments. However, minimizing this objective may result in schedules where some jobs are very tardy, which is often unacceptable in practice.

Another objective that addresses this concern is the total tardiness or, equivalently, the average tardiness.

Total tardiness objective function is defined as:

$$\sum_{j=1}^{n} T_j$$

Suppose different jobs carry different priority weights, where the weights of job $j$ is $w_j$. The larger the weight of the job, the more important it is. Then a more general version of the objective function is the total weighted tardiness

$$\sum_{j=1}^{n} w_j T_j$$

Some other performance measures involving tardiness are:

- mean tardiness:
  $$\bar{D} = \frac{1}{n} \sum_{j=1}^{n} D_j$$

- mean weighted tardiness:
  $$\bar{D}_w = \frac{1}{\sum_{j=1}^{n} w_j} \sum_{j=1}^{n} w_j D_j$$

- Weighted number of tardy tasks:
  $$U_w = \sum_{j=1}^{n} w_j U_j$$
  where $U_j=1$ if $C_j>d_j$, and 0 otherwise

Some of the problems involving tardiness are:

1. Minimizing the maximum tardiness

   $$\text{Min } (\max (T_i))$$

2. Minimizing the number of tardy jobs. The number of tardy jobs is defined as,

   $$N_T = \sum_{i=1}^{n} \delta(T_i)$$
   where $\delta(x) = 1$ if $x > 0$, and $\delta(x) = 0$ otherwise

**Total (Weighted) Completion Time**

The total completion time, which is the sum of the completion times of all the jobs, is defined as:

$$\sum_{j=1}^{n} C_j$$

where $C_j$ denote the completion time of the $j^{th}$ job in a batch of $n$ jobs given

The total weighted completion time is defined as:
\[
\sum_{j=1}^{n} w_j C_j
\]

where \( w_j \) denotes the weight assigned to the \( j^{th} \) job in a batch of \( n \) jobs given.

Minimizing the total (weighted) completion time allows one to find an indication to the total holding or inventory caused by the schedule. Total (weighted) completion time is also referred to as the (weighted) flow time as the sum of the completion times is equivalent to the flow time.

**Discounted Total Weighted Completion Time**

Discounted total weighted completion time is calculated as:

\[
\sum_{j=1}^{n} w_j (1 - e^{-rt_j})
\]

This is a more general cost function then the previous one; the costs are now discounted at a rate of \( r, 0 < r < 1 \), per unit time. Which means if job \( j \) is not completed by time \( t \), an additional cost \( w_j e^{-rt_j} dt \) is incurred over the period \([t, t+dt]\). If job \( j \) is completed at time \( t \), the total cost incurred over the period \([0, t]\) is \( w_j (1 - e^{-rt}) \). The value of \( r \) is usually close to 0, say 0.1 or 10%.

**Mean (Weighted) Flow Time**

The mean flow time is defined as:

\[
\bar{F} = \frac{1}{n} \sum_{j=1}^{n} F_j
\]

where \( F_j \) denotes the flow time of the \( j^{th} \) job in a batch of \( n \) jobs given.

The weighted mean flow time is defined as:

\[
\bar{F}_w = \frac{1}{\sum_{j=1}^{n} w_j} \sum_{j=1}^{n} w_j F_j
\]

where \( w_j \) denote the weight assigned to the \( j^{th} \) job in a batch of \( n \) jobs given.

The usual problem considered in scheduling is minimizing the (weighted) mean flow time. Minimization of the mean flow time criterion yields minimization of the mean response time and the mean in process time of the scheduled task. Note that it is equivalent to minimizing the mean (weighted) completion time.

**Mean Waiting Time**

The mean waiting time is defined as:

\[
\frac{1}{n} \sum_{j=1}^{n} W_j
\]

where \( W_j \) denotes the waiting time of the \( j^{th} \) job.
Some of the problems of interest are which involves waiting time are minimizing the mean waiting time or minimizing the maximum waiting time:

\[ \text{Min} \left( \max \left( W_j \right) \right) \]

**Mean (Weighted) Earliness**

None of the objectives described so far penalizes the early completion of a job. In practice, however, it is usually disadvantageous to complete a job early, as doing so may lead to storage costs and additional handling costs.

Mean earliness is defined as:

\[ \bar{E} = \frac{1}{n} \sum_{j=1}^{n} E_j \]

where \( E_j \) denotes the waiting time of the \( j^{th} \) job

Mean weighted earliness is defined as:

\[ \bar{E}_w = \frac{\sum_{j=1}^{n} w_j E_j}{\sum_{j=1}^{n} w_j} \]

**Equivalence of Performance measures**

Two performance measures are equivalent if a schedule, which is optimal with respect to one, is optimal with respect to the other too.

\( \bar{C} \sim \bar{F} \sim \bar{W} \) are equivalent because minimizing one guarantees minimization of others.

\[ C_i = F_i + \gamma_i = W_i + \sum P_i + \gamma_i = L_i + d_i \]

\( L_{\text{max}} \) and \( T_{\text{max}} \) are equivalent

\[ T_{\text{max}} = \max \{ \max \{ L_1, 0 \}, \ldots, \max \{ L_n, 0 \} \} \]

\[ T_{\text{max}} = \max \{ L_1, \ldots, L_n, 0 \} \]

\[ T_{\text{max}} = \max \{ L_{\text{max}}, 0 \} \]

\[ \min L_{\text{max}} \Rightarrow \min T_{\text{max}} \]

**7.2.6 Classes of Schedules**

**Nondelay Schedule**

A feasible schedule is called non-delay if no machine is kept idle while an operation is waiting for processing. No unforced idleness.

Requiring a schedule to be nondelay is equivalent to prohibiting unforced idleness. For many models, including all those that allow preemptions and have regular objective functions, there are optimal schedules that are nondelay. For many models the goal is to find an optimal nondelay schedule. However, there are models where it may be advantageous to have periods of unforced idleness.

A smaller class of schedules within the class of all nondelay schedules is the class of nonpreemptive, nondelay schedules.
Active Schedule

A feasible schedule is called active if it is not possible to construct another schedule by changing the order of processing on the machines and having at least one operation finishing earlier or no operation finishing later.

In other words, a schedule is active if no operation can be put into an empty hole earlier in the schedule while preserving feasibility. A preemptive, nondelay schedule has to be active, but reverse is not true.

Semi-active Schedule

A feasible schedule is called semi-active if no operation can be completed earlier without changing the order of processing on any one of the machines.

It is clear that an active schedule has to be semi-active. However, the reverse is not necessarily true.

7.3 Classification of Deterministic Scheduling Problems

In the past four decades, considerable amount of theoretical research has been done in the field of deterministic scheduling. This work has lead to a variety of specific problem instances and solution techniques. A taxonomy is always useful in fields with as diverse an application as scheduling problems. During this four decade period, a notation has evolved that concisely captures the structure of most of the deterministic models considered in the literature. In this section a basic classification notation based on a classification scheme widely used in literature (e.g. [155]) is explained.

The notation is composed of three fields $\alpha|\beta|\gamma$. They have the following meaning:

The first field $\alpha$ describes the processor environment.

The second field $\beta$ describes task and resource characteristics.

The third field $\gamma$ denotes an optimality criterion (performance measure).

$\alpha$ consists of two parameters $\alpha=\alpha_1, \alpha_2$:

- $\alpha_1$ characterizes the type of processor used.
- $\alpha_2$ denotes the number of processors in the problem.

$\beta$ consists of eight parameters $\beta=\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8$:

- $\beta_1$ indicates the possibility of task preemption.
- $\beta_2$ characterises additional resources.
- $\beta_3$ reflects the precedence constraints.
- $\beta_4$ describes ready times.
- $\beta_5$ describes task processing times.
• $\beta_6$ describes deadlines.
• $\beta_7$ describes the maximal number of tasks constituting a job in case of job shop systems
• $\beta_8$ describes a no-wait property in the case scheduling on dedicated processors.

These parameters can take the values explained below. $\phi$ denotes an empty symbol and is omitted in presenting problems.

$$\alpha_1 = \{\phi, P, Q, R, O, F, FF, J, FJ\}$$

- $\alpha_1 = \phi$: single processor
- $\alpha_1 = P$: identical parallel machines
- $\alpha_1 = Q$: uniform parallel machines
- $\alpha_1 = R$: unrelated parallel machines
- $\alpha_1 = O$: dedicated machines: open shop system
- $\alpha_1 = F$: dedicated machines: flow shop system
- $\alpha_1 = FF$: dedicated machines: flexible flow shop system
- $\alpha_1 = J$: dedicated machines: job shop system
- $\alpha_1 = FJ$: dedicated machines: flexible job shop system

$$\alpha_2 = \{\phi, m, s\}$$

- $\alpha_2 = \phi$: the number of processors is assumed to be variable
- $\alpha_2 = m$: the number of processors is equal to $m$ ($k$ is a positive integer).
- $\alpha_2 = s$: the number of processor stages is equal to $s$ ($s$ is a positive integer).

$$\beta_1 = \{\phi, pmtn\}$$

- $\beta_1 = \phi$: no preemption is allowed.
- $\beta_1 = pmtn$: preemption is allowed

$$\beta_2 = \{\phi, res\}$$

- $\beta_2 = \phi$: no additional resources exist
- $\beta_2 = res$: there are specified resource constraints

$$\beta_3 = \{\phi, prec, uan, tree, chains\}$$

- $\beta_3 = \phi$: independent tasks
- $\beta_3 = prec$: general precedence constraints
- $\beta_3 = uan$: uniconnected activity networks
- $\beta_3 = tree$: precedence constraints forming a tree
- $\beta_3 = chains$: precedence constraints forming a chain

$$\beta_4 = \{\phi, r_j\}$$

- $\beta_4 = \phi$: all ready times are zero
\( \beta_4 = r_j \): ready times differ per task
\[ \beta_4 \in \{ \phi, p_j = p, p \leq p_j \leq \bar{p} \} \]

\( \beta_5 = \phi \): tasks have arbitrary processing times
\( \beta_5 = (p_j = p) \): all tasks have processing times equal to \( p \) units
\( \beta_5 = (p \leq p_j \leq \bar{p}) \): no \( p_j \) is less than \( p \) or greater than \( \bar{p} \).
\[ \beta_5 \in \{ \phi, \bar{d} \} \]

\( \beta_6 = \phi \): no deadlines are assumed in the system (however, due dates may be defined if a due date involving criterion is used to evaluate schedules)
\( \beta_6 = \bar{d} \): deadlines are imposed on the performance of a task set.
\[ \beta_6 \in \{ \phi, n_t \leq k \} \]

\( \beta_7 = \phi \): the above number is arbitrary or the scheduling problem is not a job shop problem.
\( \beta_7 = (n_j \leq k) \): the number of tasks for each job is not greater than \( k \).
\[ \beta_7 \in \{ \phi, n \leq \text{no-wait, block, prmu, recrc, Mj, Sjk} \} \]

\( \beta_8 = \phi \): buffers of unlimited capacity are assumed
\( \beta_8 = \text{no-wait} \): buffers among processors are of zero capacity and a job after finishing its processing on one processor must immediately start on the consecutive processor.
\( \beta_8 = \text{Block} \): Blocking
\( \beta_8 = \text{Prmu} \): Permutations (FIFO queue ordering
\( \beta_8 = \text{Recrc} \): recirculation (visit a machine more than once)
\( \beta_8 = \text{Mj} \): Machine eligibility restriction
\( \beta_8 = \text{Sjk} \): Sequence dependent setup times

Any other entry that may appear in the \( \beta \) field is self-explanatory.

The third field \( \gamma \) denotes the performance measure.
\[ \gamma \in \{ C_{\text{max}}, \sum C_j, \sum w_j C_j, L_{\text{max}}, \sum D_j, \sum w_j D_j, \sum E_j, \sum w_j E_j, \sum U_j, \sum w_j U_j, \} \]

where
\[ \sum C_j = \bar{C}, \sum w_j C_j = \bar{E}_w, \sum D_j = \bar{D}, \sum w_j D_j = \bar{D}_w, \sum E_j = \bar{E}, \sum w_j E_j = \bar{E}_w, \sum U_j = \bar{U}, \sum w_j U_j = \bar{U}_w, \]

and "-" means testing for feasibility whenever scheduling to meet deadlines is considered.

Below a couple of examples are described to show how this notation is interpreted.
P||Cmax reads as "Scheduling of non-preemptable and independent tasks of arbitrary processing durations, arriving to the system at time 0, on parallel, identical processors in order to minimize the schedule length".

O3|pmtn,rj|ΣCj stands for "Preemptive scheduling of arbitrary length tasks in three machine open shop where operations arrive at different time moments, and the objective is to minimize mean flow time".

### 7.4 Computational complexity

Scheduling problems are optimisation problems defined on their finite set of active schedules. The cardinality of this set is usually so large that complete enumeration of all elements is not feasible within reasonable time.

Consider scheduling of three jobs on a single machine.

```
Job1, Job2, Job3 → Machine
```

There are 3! = 6 possible sequences to chose from:
- Job1-Job2-Job3
- Job1-Job3-Job2
- Job2-Job1-Job3
- Job2-Job3-Job1
- Job3-Job1-Job2
- Job3-Job2-Job1

But for twenty jobs the number of sequences becomes 20! = 2.4x10^{18}. To evaluate the performance of each sequence for the latter case would take years with even fastest processing speed available. Computation time increases factorially with number of jobs to be scheduled. For this reason, more subtle solution methods are required, taking into account the specific structure of scheduling problems. These solution methods are of varying quality. On one hand, combinatorial analysis may lead to very efficient algorithms that produce an optimal schedule in a predictable number of steps, with this number increasing at most polynomially with the size of the problem. On the other hand, it often seems necessary to resort to far less predictable enumerative methods such as branch and bound. The question here is whether the complexity of a problem is such that the latter step is unavoidable.

A partial answer to this question can be obtained by applying results from the area of computational complexity. One important finding is that if there is an efficient algorithm for any of the problems in the NP Complete class, it can provide a similarly efficient algorithm for all the other NP Complete problems as well (see section 6.3.4). But no such efficient algorithm has been found so far and existence of such an
algorithm is highly unlikely. Thus, if a particular scheduling problem can be shown to belong NP Complete class as well, the use of an enumerative method for it is justified reasonably well; no substantially better method is likely to exist. Many notorious problems such as the 0-1 programming problem, the knapsack problem and the travelling salesman problem are some of the NP Complete problems. Scheduling of soaking pits and rolling mill system is also an example of NP hard complexity. The results found on the class of scheduling problems that SPO belongs to is discussed further in the chapter 9.

From complexity analysis results, it appears that efficient algorithms exist only for a very limited class of problems. Hence many of the algorithms developed are of an enumerative nature.

7.5 Discussion

This section shows that scheduling problems have many aspects. Although there are many other important issues not included here, some of the most fundamental scheduling problems have been explained in detail. The information provided should provide sufficient information to understand any literature written in this field and especially to grasp the work done to develop the SPO system.
8 Dynamic Programming

Dynamic programming is the method used to allocate pit loads to the pits and to decide the start and end times of the heating processes. This section gives the necessary background on dynamic programming to understand the developed application. It starts with an overview which contains a concise definition and explanation of the underlying principles and a summary of the background of dynamic programming. In the following sections a simple example is explained to clarify the idea, the computational difficulties that are faced in applying dynamic programming are presented and methods to overcome these difficulties are explained.

8.1 Overview

Dynamic Programming is the name of the mathematical theory for multistage decision processes. Decision making involves model building and then solving the model to determine an optimal solution. The representation of a problem in an abstract or symbolic form is known as a mathematical model. Optimisation means finding a best solution for the model among several feasible alternatives. Dynamic Programming is an approach to optimisation rather than optimisation itself. Many models, encompassing different disciplines and areas of application, can be solved by dynamic programming approach. These models contain many decision variables and have a mathematical structure in which calculations of the optimal decisions can be done sequentially. The way an optimal solution is determined depends on the form of the objective function and constraints, the number of variables and the kind of computational facilities available. Often, before performing the optimisation, it is desirable to make some changes of variables and transformations. In contrast to simplifying the model, these preparatory operations preserve the properties of the model completely. The transformed model has the same optimal solution as the original one, but it is in a form that can be optimised more easily. Basically, dynamic programming is such a transformation of a problem into a different form more suitable for optimisation. The name ‘Dynamic Programming’ is misleading in suggesting of a computer programming technique. The name was coined because in many cases Dynamic Programming leads to a solution in the form of a program for a digital computer. Dynamic programming takes a sequential or multistage decision process containing many interdependent variables and converts it into a series of single-stage problems, each containing only a few variables. For example, a problem with N decision variables can be transformed into N sub problems, each containing only one decision variable. With this transformation, number of feasible solutions and the value of the objective function associated with each feasible solution are preserved. This
transformation is based on the intuitively obvious principle known as principle of optimality [159]:

An optimal set of decisions has the property that whatever the first decision is, the remaining decisions must be optimal with respect to the outcome which results from the first decision.

In an optimisation algorithm for a multistage decision problem, the computations increase exponentially with the number of variables, but only linearly with the number of subproblems. Thus there can be great computational saving by using dynamic programming. Often this saving makes the difference between an insolvable problem and one requiring only a small amount of computer time.

Dynamic Programming can also be described as a modified form of recursion. The essential difference is that Dynamic Programming keeps its intermediate results whereas recursion does not.

It would be misleading to suggest that the method is new since it relies heavily on mathematical induction. It turns out that it is natural to treat a sequence of decisions by reversing the order. For this reason, the analysis is also called backward induction.

Dynamic programming was practiced long before it was named. Wald's work on sequential decision theory contains the idea of dynamic programming approach [160]. The two papers by Dvoretzky, Kiefer, and Wolfowitz on inventory theory are certainly in the spirit of dynamic programming [161, 162].

The work of great importance carried out by the research groups under the direction of R. Bellman in the United States and L.S. Pontryagin in the U.S.S.R. revealed the value of dynamic programming as economic studies or in advanced technological programs such as those associated with space flights.

R. Bellman is known as the father of dynamic programming. He invented the not descriptive but alluring name Dynamic Programming for the approach. His research in the 1950's led to the publication of a large number of significant papers on dynamic programming. His first book on the theory of dynamic programming was published in 1957 [159]. His other works on dynamic programming include numerous articles, a book on the application of dynamic programming to control theory and a book on applied dynamic programming, written in collaboration with S. Dreyfus [163, 164]. These references contain many insights into dynamic programming and some fascinating exercises, but for non-mathematicians they are hard going. [165, 166, 167 and 168] are introductory books that give a more readable presentation.

Certain problem areas, such as inventory theory, allocation, control theory, and chemical engineering design, have been particularly fertile for dynamic programming
applications. The property basic to these problems is that decisions can be calculated sequentially.

Consider a chemical process consisting of a heater, reactor, and distillation tower connected in series. It is desired to determine the optimal temperature in the heater, the optimal reaction rate, and the optimal number of trays in the distillation tower. All of these decisions are interdependent. But whatever temperature and reactor rate are chosen, the number of trays must be optimal with respect to the output from the reactor. So the optimal number of trays must be determined as a function of the reactor output. Since the optimal temperature or reaction rate is not known yet, the optimal number of trays and return from the tower must be found for all feasible reactor outputs. Continuing sequentially, whatever temperature is chosen, the reactor rate and number of trays must be optimal with respect to the heater output. To choose the best reaction rate as a function of the heater output, the dependence of the distillation tower on the reactor output must be accounted for. But the optimal return from the tower is already known as a function of the reactor output. Hence, the optimal reaction rate can be determined as a function of the reactor input, by optimising the reactor together with the optimal return from the tower as a function of the reactor output.

In making decisions sequentially as a function of the preceding decisions, the first step is to determine the number of trays as a function of the reactor output. Then the optimal reaction rate is established as a function of the input to the reactor. Finally, the optimal temperature is determined as a function of the input to the heater. Finding decision functions, the chemical process can be optimised one stage at a time.

8.2 Principle of Optimality

Dynamic Programming is based on the Principle of Optimality. Just as in calculus the fundamental idea of differentiating a function and equating it to zero (and also evaluating the end points) is used to find its minima and maxima, dynamic programming uses the principle of optimality expressed in the functional equation.

The key idea of the principle of optimality is that optimisation over time can often be regarded as optimisation in stages. The desire to obtain the lowest possible cost at the present stage is traded off against the implication this would have for costs at future stages. The best action minimizes the sum of the cost incurred at the current stage and the least total cost that can be incurred from all subsequent stages, consequent on this decision. Equivalently, once a particular state is reached in a sequential decision process, the remaining decisions must be optimal with respect to that state.
The principle of optimality can be verified quickly enough using a proof by contradiction but it may not be easy to grasp. It is therefore worthwhile to give a simple example to illustrate the idea it expresses.

8.3 Example

The following example from [167] presents a simple illustration of the dynamic programming approach. The entries in the matrix (6.1) represent the costs associated with the positions in a rectangle. It is required to find an optimal route from the top left-hand corner to the bottom right-hand corner which consists of steps, either to the right or downwards, at each stage. The cost of following any particular route is the sum of all the entries encountered on the way. For example, the cost of moving down the first column and along the bottom row is $2 + 4 + 5 + 0 + 3 + 8 + 5 + 0 = 27$. An optimal route is one which the total cost is a minimum.

$$
\begin{pmatrix}
2 & 5 & 3 & 8 & 6 \\
4 & 2 & 9 & 4 & 1 \\
5 & 3 & 2 & 6 & 9 \\
0 & 3 & 8 & 5 & 0
\end{pmatrix}
$$

In order to find such a route, we construct another matrix in which each entry represents the minimum cost, for that position, of reaching the bottom right-hand corner. The complete minimum-cost matrix is (6.2) and the construction proceeds by examining the columns in reverse order. Thus, for the position in row 1, column 5, there is only one admissible path, as indicated by the arrows, and the minimum cost is $6 + 1 + 9 + 0 = 16$. We can now deal with positions in column 4, starting at the bottom where is the minimum cost is $5 + 0 = 5$. The next entry in row 3, column 4, is obtained by noting that it is preferable to make the initial move downwards, giving a total coast $6 + 5 = 11$, rather than moving to the right. It does not take long to work backwards through the matrix in this way; the minimum cost for any position can be found by comparing the two neighbouring entries on the right and below it. The last entry to be obtained is that in the top left-hand corner and this is based on the fact that the smallest total of $2 + 22 = 24$ is achieved by making the first move downwards. Notice that the optimal route through the matrix is determined by following the arrows and, finally, we can verify the minimum total cost by adding up the appropriate entries in the original matrix: $2 + 4 + 2 + 3 + 2 + 6 + 5 + 0 = 24$.

$$
\begin{pmatrix}
24 & 23 & 25 & \to & 22 & 16 \\
\downarrow & \downarrow & \downarrow & & \downarrow \\
22 & \to & 18 & 22 & 14 & \to & 10 \\
\downarrow & \downarrow & & \downarrow \\
21 & \to & 16 & \to & 13 & \to & 11 & 9 \\
\downarrow & \downarrow & \downarrow \\
16 & \to & 16 & \to & 13 & \to & 5 & \to & 0
\end{pmatrix}
$$

(6.2)
8.4 Computational Problems

In tabular dynamic programming calculations, at each stage, for each feasible value of the state variables, the return is computed for every feasible value of the decision variables. Then, for each feasible value of the state variables, an optimal value for the decision variables is recorded to trace the optimal solution. For this reason, it might not be possible to solve a problem with a large number of stages and state variables even on a high-speed computer because of the huge memory requirements. This problem, which is also referred to as "curse of dimensionality", is one of the biggest stumbling blocks in the widespread use of dynamic programming.

Computational aspects of dynamic programming concern the solution of recursive equations [168]:

\[ f_n(X_n) = \max_{D_n} Q_n(X_n, D_n) \quad n = 1, \ldots, N \]  
\[ Q_n(X_n, D_n) = r_n(X_n, D_n) \cdot f_{n-1}(X_{n-1}) \]

where

- \( n \) is the index referring to stage,
- \( X_n \) is the current state,
- \( D_n \) is the decision taken in the current stage,
- \( r \) is the stage return,
- \( f \) is the optimal return function.

In successively evaluating the \( f_n(\cdot) \), storage locations are needed in the computer for \( \{f_{n-1}(X_{n-1})\}, \{f_n(X_n)\}, \) and \( \{D^*_n(X_n)\} \), which denotes the optimal decision for each state \( X_n \) at stage \( n \).

Consequently if there are \( N \) states altogether, \( 3N \) words of computer storage is needed, apart from those required to store the program itself. Also, in order to backtrack at the end, after \( f_N(\cdot) \) has been evaluated, all the \( D^*_n(X_n) \) for all the stages are needed. Thus the number of words of storage required is of the same order as the number of states.

If a dynamic program has ten stages, one decision variable and four state variables per stage, each state variable having one hundred feasible values. The total storage requirement for optimal return and decision functions alone is \( 10(1+2)\times(100)^4 = 3\times10^9 \) storage cells. To this must be added memory space for initial data on returns and transformations, for other intermediate results, and for the program instructions.

Consider the SPO example. There are 10 soaking pits for allocating pit loads into them using a dynamic programming approach. The state of pit allocation can be denoted by a state description \((P_1, P_2, \ldots, P_{10})\) where \( P_m \) is the occupation level of the \( m \)th pit. If a
A schedule of 9 hours period is planned in one hour segments (i.e. each pit can take 9 values from 0 to 8), then it is natural to define \( f_n(P_1, P_2, \ldots, P_{10}) \) as the expected cost associated with having occupation levels \( P_m \) of the \( m^{th} \) pit \((m=1, \ldots, 10)\) and proceeding over the next \( n \) periods. It is disconcerting to think of the number of states. The states can be anything between 0000000000 to 8888888888, so there are \( 9^{10} \) states altogether.

The number of states is given by the number of values each variable can take to the power or number of variables. Thus if there are \( M \) variables and each can take \( S \) levels, the total number of states is \( S^M \). Doubling the number of levels increases the number of states by a factor \( 2^M \), whereas doubling the number of variables increases the number of states by a factor of \( S^M \).

Storage requirements depending on the number of decision variables per stage is insignificant when compared with storage requirements depending on the number of state variables. The number of entries in the lists of optimal functions increases linearly with the number of decision variables. The factor that has the most significant influence on the total storage requirement is the number of state variables per stage.

To appreciate the storage problem we must have at least a little understanding of how a digital computer operates [168, 169]. Basically, there are two time-consuming processes in the computer: the calculations in the processing unit and the transfer of information to and from the storage units. Thus the time for the total operation depends almost entirely on speed of calculation and transfer of information. A large scientific computer contains several kinds of storage mechanisms (see Figure 68). The most general distinction is between internal and external storage. From internal storage there are direct links to the processing and control units, but all communication to and from external storage must go through internal storage.

![Figure 68. Operation of a digital computer](image)

Advanced internal storage mechanisms, give rapid access to data and instructions. The times to locate and transfer information to and from the processing unit are comparable to unit calculation times. However, the amount of high speed memory included in any computer system is limited. To augment the high speed memories, there are cheaper forms of internal storage, such as using a part of the hard disk as
virtual memory. However, these memory systems are much slower than the rapid access memories. When storage requirements for data and instructions exceed the rapid access memory, a bottleneck forms. This bottleneck can become acute when the amount of data and instructions exceeds the entire capacity of the internal storage and it becomes necessary to use external storage. But retrieval of information from external storage can be more than a thousand times slower than that from rapid access memory.

Sensible programming can partially alleviate delays caused by transfer of data from external to internal storage, by carrying out these transfers while other operations are going on in the processing unit. Nevertheless, when there is a great deal of data to be transferred from auxiliary to internal units, the transfer time can exceed the time required in the internal units, and this causes a serious bottleneck. The easiest remedy may be expanding the memory of the computer.

Since the optimal decision functions are not used until the optimal policy is traced sophisticated programming will allow them to be stored externally with almost no penalty. In particular, by reversing internal storage space for only the optimal decision function currently being computed (Dn(Xn)) and for fn(Xn) and fn-1(Xn-1), we find that the internal storage requirement for optimal functions is reduced to 3SM. For the numbers just given this would reduce storage requirements by a factor of ten, but would probably not be decisive. It is possible to reduce storage requirements to SM by reserving internal storage only for fn-1(Xn-1). Putting Dn(Xn) and fn(Xn) into external storage immediately will probably not cause a large delay. It is absolutely crucial to keep fn-1(Xn-1) in the internal memory because when computing fn(Xn), we cannot anticipate in advance the sequence in which the values of fn-1(Xn-1) will be needed. However, SM may exceed internal storage capacity, which is the case in the SPO example.

Methods must be found that require less storage. Even methods that reduce requirements at the expense of increasing the number of computations might be highly desirable. At some point we might be willing to trade one transfer from internal to external storage for a million calculations, as the latter would take less time. Note how the computer has influenced our idea of what constitutes a desirable method of calculation. The concept of the number of calculations being the paramount is replaced by another in which routineness and storage requirements have an important role in determining the efficiency of a method. Dynamic programming calculations are certainly routine enough, but may not always satisfy the restrictions on storage space.

The most obvious way to reduce storage requirements is to reduce the number of state variables. In the storage formula N(D+2)SM, where M is the number of state
variables., reduction of $M$ from 4 to 2, in case $N=10$, $D=1$, $S=1000$, would be decisive. In fact, it would be so decisive that we might be willing to solve thousands of problems with $M=2$ rather than solve one with $M=4$. Most state variable reduction methods have originated with Bellman [159, 163, 163].

8.5 Computational Refinements

Computational refinements related to dynamic programming are associated with the state variables. There are two basic strategies. The first is to reduce the number of feasible values of a state variable. The second, when there is more than one state variable, is to eliminate some state variables altogether.

8.5.1 Coarse Grid Approach

Out of the number of values and number of variables, it is latter which is the more restricting from the computational point of view, since it is often possible to use a coarser grid size to reduce the number of values each variable can take. Under certain circumstances the coarse grid approach can determine the optimal decisions without having to determine the optimal return for all feasible values of the state variables.

The coarse grid approach is used to reduce the number of feasible values of each state variable. It is based on the notion of solving a series of problems, beginning with only a few widely spaced values for each state variable. The solution to the first problem yields an approximation to the true solution. Based on this approximation some previously feasible values of state decision variables can be eliminated. In the new and smaller feasible region, finer spacing is used on the state and decision variables to obtain a better approximation to the true solution. This procedure continues until the desired accuracy is attained. However, one must be aware of pitfalls that can lead to entirely erroneous results in some situations. The grid may not be fine enough to detect a steep, narrow peak. The danger of the coarse grid approach is that the optimum may be missed because of a narrow ridge. However, functions with quite narrow ridges are not very common. If one is willing to take the risk of missing a narrow ridge, the coarse grid approach can be of considerable computational help. One can, to a certain extent, control the risk by choosing an appropriate grid size. But there is a conflicting objective; the coarser the initial grid, the fewer the computations, but the greater the chance of missing a narrow ridge.

Local exploration, on a very fine grid around the final solution, may be used to test the validity of the solution. However, generally the optimal integer solution may not even be close to the optimal solution without integer constraints.

The role of the state variable is very different from the role of the decision variable, so a fundamentally different approach must be taken to reduce the number of values of
the state variable. Consider the basic recursion equations (6.3) and (6.4). Since we are interested in the maximum values of $Q_n(X_n,D_n)$ only as a function of $X_n$, it is justifiable and certainly advisable to calculate $Q_n(X_n,D_n)$ for a few values of $D_n$ as possible, provided that these few calculations yield the maximum. On the other hand, the recursion equation stipulates that $f_n(.)$ must be calculated for all feasible values of $X_n$, since the optimal return function $f_n(X_n)$ is used in its entirety to calculate $f_{n+1}(X_{n+1})$. We are well aware that $f_n(X_n=A) > f_n(X_n=B)$ gives no indication whatsoever about the optimality or even preferability of $X_n=A$, or $X_n=B$. In fact, it is not until the optimal return has been calculated from the whole process that we can begin to determine the optimal values for intermediate state variables $X_n$, $n=N-1,...,1$. The problem is simply that to determine the optimal states and decisions, $f_n(X_n)$ must be known. But to calculate $f_n(X_n)$ we must know $f_{n-1}(X_{n-1})$ for all feasible $X_{n-1}$. Under certain circumstances the coarse grid approach can be used to determine the optimal decisions without having to determine the optimal return for all feasible values of the state variable.

8.5.2 Lagrange Multipliers

A suggestion frequently made for overcoming the curse of dimensionality is the use of Lagrange multipliers; however they are of limited applicability. The aim of Lagrange Multipliers is to reduce a dynamic programming problem to a smaller problem in which the state description has at least one less dimension. It must be emphasized that the sufficient condition for successful use of Lagrange multipliers may not hold. Great care needs to be taken in using the Lagrange multipliers to ensure that optimal policies are not missed. Unless by varying $\lambda$ we can find a solution which satisfies the constraint exactly, we cannot take it for granted that one of the solutions which has been determined is optimal. In practice, we might possibly try to find the second best, third best, fourth best policy, and so on until the constraint is satisfied exactly, but the derivation of these policies is not trivial. Consequently the power of Lagrange multipliers in reducing the burden of dimensionality is limited.

If it appears difficult to reduce the number of values of a state variable, it seems to be against the grain of dynamic programming altogether to eliminate state variables. The state variable is the basic element that allows us to treat optimisation problems recursively. However, there are computational procedures for reducing the number of state variables. When we solve a problem on a computer, and extremely important benefit from state variable elimination is storage reduction. The optimal decision functions $D_n(X_n)$ that must be recorded to trace the optimal policy will, of course, have fewer entries when there are fewer state variables. In most cases the computational refinements associated with state variable reduction yield only approximate solutions, and in some circumstances may not work at all.
The use of Lagrange multipliers and grid size reduction are only two of a number of methods of solving computational problems. Bellman and Dreyfus consider various other methods and give a number of helpful flow charts related to specific problems [163]. The best advice is to try to take account of the particular characteristics of each problem, since it is on them the most appropriate computational method depend.

8.6 Summary

The SPO systems which is discussed in detail in the next chapter first decides in which order pit loads should be rolled and then allocates them to the soaking pits which deliver them to the rolling mill in the specified order. A dynamic programming algorithm determines this allocation. In this chapter a concise review of the dynamic programming technique is presented. The chapter also contains a discussion on computational difficulties in applying this technique and possible solutions to these problems. The background provided in this chapter is used in the next chapter to discuss the SPO's soaking pit allocation module.
This chapter presents the work done on the development of the soaking pit optimisation system (SPO) which synthesises a schedule for the soaking pits/rolling mill process to increase the throughput and decrease the energy consumption by the process.

The schedule focuses on the process where groups of aluminium slabs are loaded into a number of soaking pits to be heated and after the content of a pit is soaked at the required temperature, the slabs are drawn out one by one and are rolled by the hot reversing mill. In Figure 69, an ideal allocation of resources is shown. Each colour in the figure represents an activity. The main characteristics of the ideal schedule is that the rolling mill is never idle and no pit load is ready for rolling waiting for the rolling mill to become available.

The continuous operation of the rolling mill is the prime concern in a schedule, because of the high opportunity cost (profits forgone by not using the equipment) lost when the mill remains idle unnecessarily. Preventing unnecessary energy consumption by avoiding job waiting (waiting of a pit load for rolling although it has completed its soaking process because the rolling mill is busy with rolling another pit's load) and preventing rejection of the slabs for metallurgical quality reasons (because they have been heated too long) are the other targets of the SPO algorithm proposed.

In the previous chapters 6, 7, and 8, the background theory of the methods to be used to synthesize an optimal schedule is summarised. In this chapter, an overview of the development process and results obtained are presented. The chapter starts with a brief overview of the work done on similar problems by scheduling researchers. Then
the design of the SPO system is explained and the details of each component that makes up SPO are described. The chapter concludes with results and analysis from trials of the SPO.

9.1 Background

A two-stage flexible flow-shop environment can be defined as follows. Let $J$ be a set of $n$ jobs, and a job $j_i \in J$ consist of two tasks, $t_{i1}$ and $t_{i2}$ with corresponding processing time requirements $p_{i1}$ and $p_{i2}$. The first stage of the machine environment consists of $m$ identical parallel machines, $m_1, m_2, \ldots, m_m$. And in the second stage there exists only one machine. The first task $t_{i1}$ of job $j_i$ can be processed by any of the $m$ stage-1 machines. The second task of all the jobs has to be carried out by the only stage-2 machine. Figure 70 shows an illustration of a two stage flexible flowshop environment.

![Figure 70. Two stage flexible flowshop](image)

The soaking pits/rolling mill process can be classified as a two stage flexible flowshop too. The soaking pits corresponds to the parallel stage-1 machines and the rolling mill the single stage-2 machine. Every pit load can be seen as a job consisting of two tasks, heating and rolling. The first task of any job, heating, can be carried out in any of the stage-1 machines, the pits; and the second task of all the jobs has to be carried out at the only stage-2 machine, the rolling mill. The only difference with the definition above is that the soaking pits are not identical at Alcoa Europe FRP, Kitts Green. There are 4 small pits, 5 large pits and a very small soaking pit. And the efficiency of each pit varies also.

Based on the classification scheme defined in [155], minimising the makespan (time between the start of the first job and the finish of the last job) in a two stage flowshop will be represented by the notation $F2(Pm, 1)||C_{max}$.

A list of benchmarks of scheduling flowshops and job shops, their best existing solution to date and a lower bound on the optimal value are available at (http://www.ecn.purdue.edu/labs/uzsoy).
F2(Pm, 1)||Cmax has look-behind scheduling characteristics (i.e. optimal performance of the soaking pits/rolling mill system depends on eliminating idle time from the rolling mill or, equivalently, scheduling the soaking pits so that the rolling mill does not starve). For this reason it is also referred to as a look behind flow shop (LBFS). [170] gives more information on LBFS and its dual LAFS (look ahead flow shop).

[171] shows that F2(Pm, 1)||Cmax and its inverse (F2(1, Pm)||Cmax) are equivalent. Numerous studies have been reported for the F2(Pm, 1)||Cmax configuration in the literature. [172] develops a branch and bound procedure for F2(P2, 1)||Cmax. It has been shown that the minimum makespan is attained at least for one member of the set of schedules called preferred schedules. Considerable reduction in search has been obtained by restricting attention to the preferred schedules. [173] develops another branch and bound algorithm to solve the same problem in a simplified manner. [171] develops a branch and bound algorithm for F2(Pm, 1)||Cmax in which a dominance rule is derived to restrict the size of the search tree, a branching rule is adopted and heuristic methods such as Johnson [152], list scheduling, descent algorithm, etc. are used to generate an initial upper bound.

[174] shows that F2(Pm, 1)||Cmax is NP-hard for m≥2. Therefore, much of the effort to find a solution for this problem is devoted to developing heuristic algorithms. A complete list of scheduling problems and their complexities is available at (www.mathematic.uni-osnabrueck.de/research/OR/class).

[175, 176, 177] and others have developed heuristic algorithms without worst-case error bound analysis. These heuristics are evaluated by establishing a lower bound and computing the average relative gap of the heuristic solution from the lower bound. [178] develops a heuristic for the F2(1, Pm)||Cmax, which has a tight worst-case error bound of 3-(1/m). [179] presents a heuristic with a 2-(1/m) error bound and further improves this heuristic in [180] by means of a dynamic programming algorithm developed by [181]. [182] classifies the heuristics for F2(Pm, 1)||Cmax into three classes and performs some empirical comparisons among selected heuristics in the three classes. [183] presents five heuristics. They have experimented with problems of size 50, 150, and 300 parts. The reported relative deviation from the lower bound ranges from 0.75 to 4.27% on average, depending on the heuristic and the range of the task processing times. [184] presents two cases to minimize the maximum tardiness (the amount of time a job is late). In the first case, machines are always available. Two heuristic methods are given and compared. The first heuristic uses the result of minimizing the maximum lateness in the parallel machines problem when preemption is allowed and the results of the minimization of the maximum tardiness in a one-machine problem with earliest starting times. The second one
determines a schedule at the second stage so that a feasible schedule exists at the first stage. In the second case, the authors consider that at every time, there are at least $m_k-1$ available machines. They propose a method based on the second heuristic.

Another case of a two-stage flow shop problem involves parallel non-identical machines at stage 1 and a single machine at stage 2. [185] develops a heuristic to find approximate solutions for the problem with two uniform machines at first stage to minimize the makespan for two kinds of jobs. It applies Johnson's algorithm to the two types of jobs at the first stage and finds two schedules. For the second stage, jobs are sorted in the increasing order of their completion time and assigned to their positions.

The utilization of classical scheduling theory in most production environments is minimal and the consequences of poor scheduling strategies on overall company performance are generally not appreciated. Although classical scheduling theory has matured, the theoretical methods which have been developed are still far from being widely used in practice. Scheduling theory has tended to develop a mathematical momentum of its own. Too little emphasis has been placed on practical application. There are two main reasons to this. Theory and solution methods are unknown or not properly understood by practitioners, and the ideal situations assumed by the theory are not sufficiently close to those found in practice. The work presented in this chapter aims to address this gap in theory and application.

Because even the simplified flow shop problem has been shown to be NP-hard, our efforts to find a solution algorithm for the scheduling of the soaking pits/rolling mill process have focussed mainly on heuristic methods. In the rest of this chapter the method used to synthesise a schedule is explained. This method was first mentioned in [180] where Lee and Vairaktarakis compare the throughput performance of several flexible flowshop and job shop designs. We adapt this method to solve a real industrial problem and also modify the method to solve a two-stage flexible flowshop problem in which the parallel stage-1 machines are not identical. The method uses the algorithm developed by Johnson [152], which has inspired many heuristic methods proposed to solve similar problems, and combines it with a dynamic programming method proposed by Rothkopf [181] to solve the parallel-machines problem.

9.2 The SPO Algorithm

The Soaking Pit Optimisation System (SPO) has two main tasks; prediction and scheduling. It uses artificial neural networks for predicting the durations of heating and rolling operations and a heuristic based procedure that combines two scheduling algorithms. Figure 71 shows the relation between the two modules and their
interaction with the other plant systems. The neural network module is explained in detail in chapters 4 and 5. The main components of the SPO are

- **Grouping**: Forming pit loads
- **Estimating processing times**
- **Resource allocation**.
  - Forming the processing sequence of the pit loads
  - Allocations to the pits

**NEURAL NETWORK TRAINING**

past data (slab groups, their properties, corresponding heating and rolling times)

**Furnace Control System**

available slabs & their properties (dimensions, target gauges, heat practice, due dates, etc)

**Shop Floor Reporting (SFR)**

current pits state

**SCHEDULING**

groups of slabs that will be soaked together
rolling order of the groups
pit loading program
rolling order of the slabs in the group

**Figure 71. How does SPO work?**

available slabs & their properties (dimensions, target gauges, heat practice, due dates, etc)

**Jobs: pit loads**

**Grouping**

**Estimated rolling time & heating time of jobs**

**Sequencing**

**Allocating to Pits**

groups of slabs that will be soaked together
rolling order of the groups
pit loading program
rolling order of the slabs in the group

**Figure 72. Inside Scheduling**
Figure 72 shows the relation between grouping and resource allocation modules.

9.2.1 Grouping

The slabs that are going to go through the same recipes can be soaked together in the same pit. We form groups of slabs that fill the capacity of the pits and can be heated efficiently (i.e. slabs with similar dimensions). A group of small slabs and a group of large slabs would be heated in a short period and a long period respectively whereas two groups of mixed dimensions of slabs would take a long time for both groups.

Deciding which slabs or batches will go into the same pits is similar to the two-dimensional bin-packing problem. The bin packing problem is NP-complete. [186] surveys the approximation algorithms for bin packing. The SPO Algorithm uses a combination of work instructions and best-fit heuristics to determine the groups of slabs. Dividing the pits into zones simplifies the problem and makes it possible to obtain a good solution by the BFD (Best Fit Decreasing) heuristic in a reasonable time.

Figure 73 shows the view of a pit from the top where the slaps are loaded in three predetermined zones. The slabs are placed in parallel to the slabs in the same zone and there is a minimum gap between slabs [25]. These practices are used to maximize the efficiency of the heating process.

![Minimum distance between slabs](image)

Slabs are grouped to utilize the pit capacity best within the following constraints:

- Each pit can take a certain volume and weight of slabs.
- A batch of slabs cannot be divided into different pits.
- All the batches in the same pit must have compatible soaking recipes.

Other considerations that are taken into account are:

- Pit capacity
- Process compatibility
- Experience
What is meant by BFD is basically sorting the slabs according to their dimensions and choosing the ones that will be soaked together starting from the largest to the smallest. The computational requirement of this algorithm is very small. But during the course of the project the practice of using different zones in the pit was abandoned. This resulted in a considerable amount of effort to develop an algorithm with similar performance. The research on developing a bin packing method for deciding the pit load is continuing and it is left out of this thesis. The existing Kitts Green planning system advises the pit loads that are scheduled by the present SPO system. When the work on the grouping module is completed, the SPO will be using the results from this module.

9.2.2 Estimation of processing times

Once the slabs are grouped into pit loads, the processing times of each pit load is required. Each pit load taken as a job has to be heated first and then rolled. Neural network models are trained by a significant amount of past processing data do the estimation of the duration of these two operations. We use feedforward back-propagation networks of nine neurons in the hidden layer with tansig transfer functions and one neuron in the output layer with a purelin transfer function. ANNs are trained by batch training with automated regularisation using the Bayesian framework. Before the training of the neural network models starts, a pre-processing step is included in which the input and target vectors are scaled to values between [-0.5, +0.5]. A post-processing step converts output back into original units.

Chapter 4 provides a concise review of artificial neural networks and chapter 5 explains SPO ANNs in detail.

9.2.3 Sequencing

The continuous operation of the rolling mill is the prime concern in any schedule because of the high opportunity cost. A desired plan would schedule the pit loads to guarantee the continuous running of the rolling mill using minimum number of soaking pits. So we first look for a good sequence in which the pit loads should arrive to the rolling mill. We use a heuristic similar to Johnson's algorithm to find the sequence in which the pit loads have to be rolled.

Johnson's algorithm gives a minimum makespan (time between the start of the first job and the end of the last job) processing schedule for a two machine flow shop (one stage-1 machine and one stage-2 machine; each job has to be processed on the stage-1 machine first then on stage-2 machine) in $n \log n$ time where $n$ is the number of jobs to be scheduled [152].
Johnson observed that processing the job with the shortest operation-1 first and the job with shortest operation-2 last; and scheduling the shortest Machine-1 operations and longest Machine-2 operations as early as possible would give the optimal schedule. This observation can be verified by studying Figure 74, below.

Consider scheduling of three jobs on two machines where each job has to be processed on machine M1 first, then on machine M2 with the following processing times:

<table>
<thead>
<tr>
<th>Machines/Jobs</th>
<th>J1</th>
<th>J2</th>
<th>J3</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>6</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>M2</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

There are 6 possible sequences: J1-J2-J3, J1-J3-J2, J2-J1-J3, J2-J3-J1, J3-J1-J2, and J3-J2-J1 where J2-J1-J3 gives the minimum makespan.

For any sequence, machine M1 will start working at T=0, and work until T=6+2+8=16. Therefore, all scheduling must concentrate on is trying to make the Gantt chart for machine M2 as compact as possible. Total makespan will be 16 plus the duration of last job on machine two. And, M2 will not work at the initial period when M1 is doing its first scheduled job. So it makes sense to put the job with shortest duration on M1 first and the job with the shortest duration on machine M2 last.

If the second task of a job is very short, M2 will finish this part, while M1 is still working on the first operation of the next job. This will make M2 idle for some time. So we should try to place jobs that have long second operations in the beginning.

Another example:

<table>
<thead>
<tr>
<th>Machines/Jobs</th>
<th>J1</th>
<th>J2</th>
<th>J3</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>30</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>M2</td>
<td>25</td>
<td>28</td>
<td>20</td>
</tr>
</tbody>
</table>
Using logic developed above, the shortest task on Machine M1 is J2, so we schedule it first (the second task of J2 is the longest, so that is also in accordance with Johnson's second observation). Also, the shortest operation on machine M2 is J3, so we schedule it last. Hence we get the sequence J2-J1-J3

Johnson's algorithm can be applied to the N jobs problem as follows.

If each job Jj has two operations, of duration $P_{j1}$, $P_{j2}$ to be done on Machine M1, M2 in that sequence.

Step 1. List $A = \{1, 2, ..., N\}$, List $L_1 = \{\}$, List $L_2 = \{\}$.

Step 2. Form all available operation durations, pick the minimum.

If the minimum belongs to $P_{k1}$, remove K from list A and add to the end of $L_1$.

If the minimum belongs to $P_{k2}$, remove K from list A and add to the start of $L_2$.

Step 3. Repeat Step 2 until list A is empty.

Step 4. Join List $L_1$, List $L_2$. This is the optimum sequence.

Johnson's method only works optimally for two machines. However, since it is optimal and easy to compute, some researchers have tried to adopt it for other flexible flow shop problems (including $F2(Pm, 1) || C_{\text{max}}$). Most of the research done on flowshop problems to find a good heuristic solution has been inspired by Johnson's algorithm.

In the case of soaking pits/rolling mill problem, if we combine m stage-1 machines into an (imaginary) Machine Centre, $MC_1$, and assume the stage-2 machine as the Machine Centre $MC_2$, and apply Johnson's Algorithm by taking the processing times as ($P_{1i}/m$, $P_{2i}$), we can find a near optimal processing sequence in nlogn time where $i=\{1,..n\}$. [179] shows that this sequence would have an error bound of $(2-1/m)$.

The sequence in which the pit loads will be rolled is found by applying Johnson's algorithm with respect to processing times:

$$\{(\frac{P_{1i}}{m}, P_{2i}); i=\{1, 2, ..., n\}\} \quad (1)$$

where m is the number of soaking pits and n is the number of pitloads.

9.2.4 Allocating soaking pits

After deciding the rolling sequence we need to allocate the pit loads to the pits and determine the load, unload, heat start and roll start times.

Once we have decided the rolling sequence by the modified Johnson's algorithm as explained in the previous section, for each pit load, we appoint the starting time of rolling operation as deadline for completion of the heating operation. We allocate the pit loads to the soaking pits so that the maximum lateness of soaking pits' schedule is
So the soaking pits/rolling mill problem reduces to scheduling of n jobs with deadlines on m parallel machines to minimize maximum lateness.

Rothkopf observed that, in the problem of scheduling n immediately available tasks on m parallel processors with the assumptions [181]:

- each task must be processed by exactly one processor,
- each task i has an integer service time $T_{ij}$ if it is processed by processor j,
- each processor, j, becomes available for processing at a nonnegative integer time $t_j^*$ and is continuously available thereafter,
- each task, i, incurs a special cost $B_{ij}$ if it is processed by processor j;
- there is a cost $G_0(t_1, t_2, \ldots, t_m)$, associated with a schedule in which the processor j completes the processing of the tasks assigned to it at time $t_j$. $G_0$ is a monotonic nondecreasing function of each of its arguments for any set of values for the others.
- each processor must process the tasks assigned to it in the reverse numerical order

The following dynamic programming relationships hold:

$$G(t) = \min_{1 \leq j < m} [G_{i-1}(t + e_j T_{ij}) + B_{ij} + C_i(t_j + T_{ij})],$$

where $t = (t_1, t_2, \ldots, t_m)$, $e_j$ is the vector with the $j^{th}$ component equal to one and all other components equal to zero and $G_i(t)$ is the minimum cost schedule for units one through i if processor j first becomes available at time $t_j$ ($j = 1, 2, \ldots, m$). The $j^{th}$ choice above corresponds to the assignment of unit i to processor j.

An example:

Consider the allocation of the following jobs on two identical parallel machines:

<table>
<thead>
<tr>
<th>Job Number (i)</th>
<th>Task duration ($T_i$)</th>
<th>Deadline ($d_i$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Cost of exceeding the deadline is proportional to lateness and is measured with $\text{Cost} = \text{sgn}(T_i - d_i)(T_i - d_i)$ where $\text{sgn}(t)$ is zero for $t < 0$ and 1 at all other times.

First note that since the machines are identical $B_{ij} = 0$ and $T_{ij} = T_i$. Moreover $G_i(t_1, t_2) = G_i(t_2, t_1)$ and $t_2 = \sum_{k=i+1}^{m} T_k - t_1$. We assume that $G_0(t) = 0$.

Since we have already determined the processing order and numbered the tasks in reverse numerical order, we can immediately use equation 2 to make calculations:
\[ G_1 = \min_j \left[ \frac{G_0(t_1 + 4, t_2) + \text{sgn}(t_1 + t_2 - d)}{G_0(t_1, t_2 + 4) + \text{sgn}(t_2 + t_1 - d)} \right] \]

\[ G_2 = \min_j \left[ \frac{0 + \text{sgn}(t_1 + 4 - 6)(t_1 + 4 - 6)}{0 + \text{sgn}(t_2 + 4 - 6)(t_2 + 4 - 6)} \right] \]

\[ G_3 = \min_j \left[ \frac{0 + \text{sgn}(t_1 + 4 - 6)(t_1 + 4 - 6)}{0 + \text{sgn}(t_2 + 4 - 6)(t_2 + 4 - 6)} \right] \]

The allocation of jobs to the processor vacancies that gives the optimum cost of 0 is:

<table>
<thead>
<tr>
<th>Task No (i)</th>
<th>Machine allocated</th>
<th>Task finished at</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>(1, 0)</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>(1, 2)</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>(4, 2)</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>(6, 2)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>(6, 6)</td>
</tr>
</tbody>
</table>

So with this allocation all five deadlines are met.

Usually it is possible to obtain a zero maximum lateness for all the pit loads on soaking pits, where \( p_{i,1} \) is the initial idle time of the mill because of waiting for the first rolling task [187]. If zero maximum lateness cannot be obtained, then the number of soaking pits to be used can be increased.

When used online the processing time estimation errors can be taken into account by shifting the deadlines as much as the worst case errors and updating the plan after completion of each job.

9.3 Implementation

A document view architecture using Microsoft Foundation Classes (MFC) has been used to develop the SPO system in C++ [188]. An ODBC connection to the plant database SFR (shop floor reporting) has been established. Nucleus (soaking pit control system) connection is not yet completed. The ANN models developed in C++ are connected to the SPO system. However, the ANN models developed in Matlab have not been used in this architecture. The communication between the SPO system and the plant database system is in progress. The communication is done via an ODBC connection. The connection may enhance the performance of the system.
been used in this architecture yet. The communication between the Expert system and
the SPO system is not established either, but it would not require much extra work if
the communication is done via text files. An expert system connection may enhance
the produced schedules by utilising expert knowledge about the process. Also the
expert system could improve its supervisory control of the rolling mill by utilising the
data that SPO feeds to it.

In the initial implementation of the SPO scheduling module the system was able to
produce a schedule for 4 soaking pits very quickly. However, to get a schedule for 5
pits required very long processing times. The main reason for this is the amount of
memory required by the dynamic programming algorithm. Since internal storage
capacity is not adequate, the computer allocates virtual memory from the external
storage. From internal storage there are direct links to the processing and control
units, but all communication to and from external storage must go through internal
storage. Thus transfer of information becomes a bottleneck for the algorithm. With
addition of extra memory and use of a coarser grid in the search space this problem
has been solved. At the moment we can create a schedule for 10 soaking pits and a
release version can produce the result in less than 20 seconds. The cost function
explained in section 9.2.4 is stored in a multidimensional array. An 11 dimensional
array was developed by expanding the 2 dimensional array of [189]. The array was
then tested cell-by-cell to identify any faults. Initially the program was written just to
verify the dynamic programming algorithm. Since coding the appropriate exception
handlers and constraints in the program, it has become very robust and fast. Further
research in using sparse matrices for storing a record of the cost function of the
dynamic programming algorithm may enhance this system [190, 191].

9.4 Results

In this section, the effectiveness of the SPO algorithm in synthesising schedules for
the soaking-pit/rolling-mill process is analysed. First, a manually produced schedule is
presented and then the data of the manual schedule is used to produce an optimal
schedule using the SPO system. These two schedules are compared to demonstrate
the advantages of using the SPO system. The manual schedule, whose details are
presented in Table 19, was used at Alcoa's Kitts Green plant between 6-9 December
2001 [Thursday-Sunday]. It is produced for 16 pit loads and nine soaking pits are
considered but pit 7 is kept empty because it is reserved for a homogenising process.
In the SPO algorithm the cost of allocating a pit load to pit 7 is increased to 100 to
prevent allocating any pit loads to pit 7. A cost of 100 means SPO will allocate a pit
load to pit 7 only if this will reduce rolling mill's waiting time 100 hours, but because
the maximum makespan of this schedule is 96 hours, no pit loads will be allocated to
pit 7. Pit loads and pits are divided into four groups according to their sizes. The sizes of the pit loads and pits used in this study are presented in Table 16 and Table 17 respectively.

**Table 16. Sizes of the pit loads**

<table>
<thead>
<tr>
<th>Size</th>
<th>Pit Load Sequence Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>very small</td>
<td>796 805</td>
</tr>
<tr>
<td>small</td>
<td>787 788 794 795 797 798</td>
</tr>
<tr>
<td></td>
<td>799 801 802 804</td>
</tr>
<tr>
<td>large</td>
<td>792 793 803</td>
</tr>
<tr>
<td>very large</td>
<td>800</td>
</tr>
</tbody>
</table>

**Table 17. Sizes of the pits**

<table>
<thead>
<tr>
<th>Size</th>
<th>Pit Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>very small</td>
<td>S furnace</td>
</tr>
<tr>
<td>small</td>
<td>1 2 4 5</td>
</tr>
<tr>
<td>large</td>
<td>3 6 7</td>
</tr>
<tr>
<td>very large</td>
<td>8</td>
</tr>
</tbody>
</table>

SPO system allows allocating a pit load to a pit which is larger than the pit load's size, if this reduces the waiting time at the rolling mill. But it does not allow a pit load to be allocated to a pit which is smaller than pit load's size because this is physically not possible. Table 18 shows the size difference costs in allocating pit loads to pits. A cost of 5 means the pit load will be allocated on a pit of the corresponding size only if this reduces the rolling mill's waiting time by 5 hours.

**Table 18. Costs due to size in allocating a pit load on a pit.**

<table>
<thead>
<tr>
<th>Pit Size</th>
<th>Very Small</th>
<th>Small</th>
<th>Large</th>
<th>Very Large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Small</td>
<td>0</td>
<td>5</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>Small</td>
<td>N/A</td>
<td>0</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>Large</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Very Large</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
</tr>
</tbody>
</table>

In order to make a reasonable comparison, the SPO algorithm uses the heating and rolling times used in the manual schedule to produce the SPO schedule. This would avoid confusion with the savings due to estimating the process durations more accurately by using the ANN models explained in chapter 5.

Figure 75 presents the Gantt chart of the manually prepared schedule for the soaking-pits/rolling-mill process. The Gantt chart shows rolling and heating operations and job waiting with blue, red and black bars respectively. The makespan of the schedule is 92 hours (3 days 20 hours). The soaking pits are waiting a total of 80 hours for the rolling mill to become available to process the pit load they contain and the rolling mill is interrupted for 15 hours after starting rolling because of waiting for a pit load to become ready for rolling. In an optimal schedule, the minimum time required to process the 16 jobs of this manual schedule would be 69 hours (2 days 21 hours). If the rolling mill starts its operation as soon as possible and rolls until the end of the schedule without any interruption, the shortest schedule would be achieved which is 14 (shortest heating time) + 55 (total rolling time) = 69 hours. However, finding the optimal schedule manually is very difficult. For this particular example, there are $16! = 2.0923\times10^{13}$ orders in which the pit loads can be rolled and for each one of
these sequences there are \(8^8=16,777,216\) possible allocations of the soaking pits. It is not only impossible to check manually all the possible schedules, but also checking even a few of them needs a lot of concentration and time. However, the schedule presented in Figure 76, whose makespan is equal to the minimum possible makespan, is produced by the SPO system in less than 20 seconds. The details of the SPO schedule can also be seen Table 20. SPO algorithm first uses the modified Johnson's algorithm to find the order in which the pit loads will be rolled. It must be noted that the duration of the first task of a pit load is taken as heating+rolling time because a pit does not become available until all its content is rolled. Table 21 shows the resulting rolling order that is used in the SPO schedule shown in Figure 76. The SPO system then uses this sequence in the dynamic programming algorithm to allocate pit loads onto pits. The makespan of the resulting SPO schedule is 23 hours shorter than that of the manual schedule and the waiting times (apart from the 14 hours needed to start rolling of the first job) are completely avoided. In this example, these improvements corresponds to 25% reduction in the makespan which is a significant saving in the costs due to opportunity lost because of not using the rolling mill for more manufacturing and 4.71% reduction in the total running time of the soaking pits which is a significant saving in the costs due to fuel consumption and opportunity lost because of not using soaking pits for more manufacturing. In chapter 5, it was noted that by using the ANN models it is possible to improve the accuracy of predicting the rolling time of a pit load by half an hour. An optimal schedule produced by rolling times reduced by half an hour would result in 8 hours reduction in the makespan for the example studied in this section. This would correspond to 8.7% reduction in the makespan which is a significant saving in the costs due to opportunity lost because of not using the rolling mill for more manufacturing. In addition to these promising results, SPO system provides flexibility in the management of the soaking-pits/rolling-mill process. At present the schedules are produced manually and updated daily during the week and not updated during the weekend. In the case of a need for change in the schedule, to produce a new schedule manually there needs to be an experienced scheduler present. SPO system can produce a schedule whenever required and can update the schedule in response to events that may take place during the course of a schedule. For example, if a process takes significantly longer than predicted, the rest of the jobs can be scheduled again.
Table 19. Forward Plan for 6-9/12/01 [Thursday-Sunday]

<table>
<thead>
<tr>
<th>Sequence No</th>
<th>Rolling Order</th>
<th>Pit No</th>
<th>Start Time</th>
<th>Finish Time</th>
<th>Heating Time</th>
<th>Rolling Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>787</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>25</td>
<td>17</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>796</td>
<td>2</td>
<td>9</td>
<td>3</td>
<td>26</td>
<td>22</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>797</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>29</td>
<td>18</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>788</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>32</td>
<td>17</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>792</td>
<td>5</td>
<td>3</td>
<td>17</td>
<td>35</td>
<td>14</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>793</td>
<td>6</td>
<td>6</td>
<td>11</td>
<td>40</td>
<td>24</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>794</td>
<td>7</td>
<td>5</td>
<td>10</td>
<td>44</td>
<td>24</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>795</td>
<td>8</td>
<td>4</td>
<td>35</td>
<td>63</td>
<td>24</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>801</td>
<td>9</td>
<td>1</td>
<td>43</td>
<td>65</td>
<td>16</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>798</td>
<td>10</td>
<td>5</td>
<td>46</td>
<td>70</td>
<td>17</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>800</td>
<td>11</td>
<td>8</td>
<td>40</td>
<td>75</td>
<td>16</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>805</td>
<td>12</td>
<td>9</td>
<td>50</td>
<td>77</td>
<td>16</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>803</td>
<td>13</td>
<td>3</td>
<td>49</td>
<td>82</td>
<td>16</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>799</td>
<td>14</td>
<td>2</td>
<td>58</td>
<td>86</td>
<td>17</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>802</td>
<td>15</td>
<td>4</td>
<td>63</td>
<td>88</td>
<td>16</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>804</td>
<td>16</td>
<td>1</td>
<td>65</td>
<td>92</td>
<td>16</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

**TOTAL** | **290** | **55** | **50** |

Table 20. SPO Schedule.

<table>
<thead>
<tr>
<th>Sequence No</th>
<th>Rolling Order</th>
<th>Pit No</th>
<th>Start Time</th>
<th>Finish Time</th>
<th>Heating Time</th>
<th>Rolling Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>792</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>17</td>
<td>14</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>801</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>19</td>
<td>16</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>802</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>21</td>
<td>16</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>805</td>
<td>4</td>
<td>9</td>
<td>5</td>
<td>23</td>
<td>16</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>804</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>27</td>
<td>16</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>788</td>
<td>6</td>
<td>5</td>
<td>10</td>
<td>30</td>
<td>17</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>787</td>
<td>7</td>
<td>6</td>
<td>13</td>
<td>33</td>
<td>17</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>800</td>
<td>8</td>
<td>8</td>
<td>17</td>
<td>38</td>
<td>16</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>803</td>
<td>9</td>
<td>3</td>
<td>22</td>
<td>43</td>
<td>16</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>799</td>
<td>10</td>
<td>1</td>
<td>26</td>
<td>47</td>
<td>17</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>797</td>
<td>11</td>
<td>2</td>
<td>29</td>
<td>50</td>
<td>18</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>798</td>
<td>12</td>
<td>4</td>
<td>33</td>
<td>55</td>
<td>17</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>794</td>
<td>13</td>
<td>5</td>
<td>31</td>
<td>59</td>
<td>24</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>796</td>
<td>14</td>
<td>6</td>
<td>35</td>
<td>63</td>
<td>24</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>793</td>
<td>15</td>
<td>8</td>
<td>39</td>
<td>68</td>
<td>24</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>796</td>
<td>16</td>
<td>9</td>
<td>46</td>
<td>69</td>
<td>22</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 21. Johnson Alg. for 9 machines

<table>
<thead>
<tr>
<th>Seq No</th>
<th>(H+R)/9</th>
<th>Rolling Time</th>
<th>Rolling Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>792</td>
<td>1.89</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>801</td>
<td>2.00</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>802</td>
<td>2.00</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>805</td>
<td>2.00</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>804</td>
<td>2.22</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>788</td>
<td>2.22</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>787</td>
<td>2.22</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>800</td>
<td>2.33</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>803</td>
<td>2.33</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>799</td>
<td>2.33</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>797</td>
<td>2.33</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>798</td>
<td>2.44</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>794</td>
<td>3.11</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>795</td>
<td>3.11</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>793</td>
<td>3.22</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>796</td>
<td>2.56</td>
<td>1</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 22. Johnson Alg. for 8 machines

<table>
<thead>
<tr>
<th>Seq No</th>
<th>(H+R)/8</th>
<th>Rolling Time</th>
<th>Rolling Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>792</td>
<td>2.125</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>804</td>
<td>2.500</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>788</td>
<td>2.500</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>787</td>
<td>2.500</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>800</td>
<td>2.625</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>803</td>
<td>2.625</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>799</td>
<td>2.625</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>797</td>
<td>2.625</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>798</td>
<td>2.750</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>794</td>
<td>3.500</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>795</td>
<td>3.500</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>793</td>
<td>3.625</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>801</td>
<td>2.250</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>802</td>
<td>2.250</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>805</td>
<td>2.250</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>796</td>
<td>2.875</td>
<td>1</td>
<td>16</td>
</tr>
</tbody>
</table>
Figure 75. Manual Schedule

- Duration of the schedule: 69 hours = 2 days and 21 hours.
- Total waiting time of Pit Loads for the Rolling Mill: 0 hours.
- Total waiting time of the Rolling Mill for Pit Loads: 14 hours.

Figure 76. SPO Schedule

- Duration of the schedule: 69 hours = 2 days and 21 hours.
- Total waiting time of Pit Loads for the Rolling Mill: 0 hours.
- Total waiting time of the Rolling Mill for Pit Loads: 14 hours.
Two more schedules are produced by the SPO system to demonstrate the principles that the SPO system is based on. The first of these two schedules is produced by using the rolling order used in the manually produced schedule and the second schedule uses a rolling order produced by Johnson's algorithm for 8 machines (not 9 as was the case for the SPO schedule in Figure 76) in the first stage of the flow shop model.

Figure 77 presents the Gantt chart of the first schedule based on the manually produced rolling order. The details of this schedule are presented in Table 23. It can be seen that by shifting the starting time of the heating processes, it is possible to eliminate the waiting time of the pit loads. However, the rolling mill operation is still interrupted for 11 hours. A reduction of 9 hours is achieved in the makespan by reducing it from 92 hours to 83 hours. This schedule demonstrates the fact that a trial and error method may produce improved schedules but it is not as straightforward as it might seem. However, further work on examining different rolling orders may help in deriving heuristics to improve Johnson's algorithm, which is used in the SPO system.

As mentioned earlier, in the example studied in this section the schedules are considered for 9 soaking pits but because pit 7 is reserved for a different process, they are effectively prepared for 8 pits only. In deciding the rolling order for the SPO schedule in Figure 76, Johnson's algorithm takes the number of soaking pits as 9. Figure 78 shows a schedule where to determine the rolling order in Johnson's algorithm the number of soaking pits is taken as 8. The calculations leading to this order are presented in Table 22. Table 24 presents the details of the schedule of this rolling order. It can be seen that this schedule has deviated by 3 hours from the makespan of the "optimal" schedule because the rolling mill has to wait for 3 hours for pit load 796 to become ready for rolling after completing rolling of 805. This waiting can be avoided if 805 and 796 swap the pits they are heated in. This problem has occurred due to the costs related in allocating a pit load onto a pit with larger size. For this reason it might be necessary to do a local search within the neighbourhood of the SPO schedule if it is not optimal initially. A basic heuristic that seems to be promising is to group the pit loads according to their sizes and then to use Johnson's algorithm to find a rolling sequence for each group and combine the sequences by inserting the pit loads of a sequence in to another sequence. For the last schedule studied above, this heuristic would have produced an optimal schedule by rolling 805 before 797.
## Table 23. Manual Schedule’s rolling order

<table>
<thead>
<tr>
<th>Sequence No</th>
<th>Rolling Order</th>
<th>Pit No</th>
<th>Start Time</th>
<th>Finish Time</th>
<th>Heating Time</th>
<th>Rolling Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>787</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>20</td>
<td>17</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>796</td>
<td>2</td>
<td>9</td>
<td>0</td>
<td>23</td>
<td>22</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>797</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>26</td>
<td>18</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>788</td>
<td>4</td>
<td>4</td>
<td>9</td>
<td>29</td>
<td>17</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>792</td>
<td>5</td>
<td>3</td>
<td>15</td>
<td>32</td>
<td>14</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>793</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>37</td>
<td>24</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>794</td>
<td>7</td>
<td>5</td>
<td>13</td>
<td>41</td>
<td>24</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>795</td>
<td>8</td>
<td>8</td>
<td>17</td>
<td>45</td>
<td>24</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>801</td>
<td>9</td>
<td>1</td>
<td>29</td>
<td>47</td>
<td>16</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>798</td>
<td>10</td>
<td>2</td>
<td>30</td>
<td>52</td>
<td>17</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>800</td>
<td>11</td>
<td>8</td>
<td>45</td>
<td>66</td>
<td>16</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>805</td>
<td>12</td>
<td>9</td>
<td>50</td>
<td>68</td>
<td>16</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>803</td>
<td>13</td>
<td>3</td>
<td>52</td>
<td>73</td>
<td>16</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>799</td>
<td>14</td>
<td>4</td>
<td>56</td>
<td>77</td>
<td>17</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>802</td>
<td>15</td>
<td>6</td>
<td>61</td>
<td>79</td>
<td>16</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>804</td>
<td>16</td>
<td>5</td>
<td>63</td>
<td>83</td>
<td>16</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

## Table 24. Schedule for the rolling order of Johnson’s algorithm for 8 machines

<table>
<thead>
<tr>
<th>Sequence No</th>
<th>Rolling Order</th>
<th>Pit No</th>
<th>Start Time</th>
<th>Finish Time</th>
<th>Heating Time</th>
<th>Rolling Time</th>
<th>Wait Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>792</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>17</td>
<td>14</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>804</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>21</td>
<td>16</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>788</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>24</td>
<td>17</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>787</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>27</td>
<td>17</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>800</td>
<td>5</td>
<td>8</td>
<td>11</td>
<td>32</td>
<td>16</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>803</td>
<td>6</td>
<td>6</td>
<td>16</td>
<td>38</td>
<td>16</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>799</td>
<td>7</td>
<td>5</td>
<td>20</td>
<td>41</td>
<td>17</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>797</td>
<td>8</td>
<td>3</td>
<td>23</td>
<td>44</td>
<td>18</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>798</td>
<td>9</td>
<td>1</td>
<td>27</td>
<td>49</td>
<td>17</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>794</td>
<td>10</td>
<td>2</td>
<td>25</td>
<td>53</td>
<td>24</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>795</td>
<td>11</td>
<td>4</td>
<td>29</td>
<td>57</td>
<td>24</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>793</td>
<td>12</td>
<td>8</td>
<td>33</td>
<td>62</td>
<td>24</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>801</td>
<td>13</td>
<td>3</td>
<td>46</td>
<td>64</td>
<td>16</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>802</td>
<td>14</td>
<td>5</td>
<td>48</td>
<td>66</td>
<td>16</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>805</td>
<td>15</td>
<td>9</td>
<td>50</td>
<td>68</td>
<td>16</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>796</td>
<td>16</td>
<td>1</td>
<td>49</td>
<td>72</td>
<td>22</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
These results illustrate the potentially significant advantages of using the SPO system for scheduling the soaking pits' and the rolling mill's operations. By using the SPO system it has been possible to schedule the processes, which were planned for almost a 4 days period, to be carried out in 2 days 21 hours. Moreover machine utilization is improved by avoiding any waiting. By using the ANN models to predict the rolling time of the pit loads more accurately, further reduction in the makespan is certain because the makespan of an optimal schedule is the sum of all rolling times and the heating time of the first job. Estimating the heating times more accurately would decrease the energy consumption and free soaking pits from unnecessary heating. Using the SPO system on plant may reduce the cost of the soaking-pits/rolling-mill process significantly and give more flexibility in managing this process. Further work on development of the SPO system will initially be to implement the system to be used on site, then to develop the algorithms used in SPO further to account for all possible constraints that may be encountered.

9.5 Summary

This chapter has presented the work done on the development of the soaking pit optimisation (SPO) system which synthesises a schedule for the soaking pits/rolling mill process to increase the throughput and decrease the energy consumption by the process. SPO merges the ANN technology and classical scheduling theory to manage the operations of the soaking pits and the rolling mill. The results obtained have illustrated the benefits of using the system in terms of better utilisation of machines and energy conservation.
10 Discussion and Directions for Future Work

The research carried out on the application of expert systems to supervisory control of rolling mills has concluded that the expert system methodology is practicable and can produce flat product. The system is stable, timely, transparent and flexible. It offers major benefits in the ease of leveraging knowledge contained within data, plant and human resources, to complement existing model-based systems.

Two further successful online trials of the expert system have been carried out by the rest of the Leicester team since the work reported in this thesis. It has been decided to continue expanding the knowledge base and finalise industrial implementation. In the next phase of this work, research is aimed at enhancing the interaction with the present model based adaptation system used for level II control. The biggest difficulty of model based systems is having to cope with sudden changes in the operating point. Managing the adaptation system in such situations would decrease the time needed to reach the new operating point. This can be achieved with a knowledgebase module that manages either the operation of adaptation (e.g. setting long term and short term forgetting factors) or directly setting the parameters that are adapted.

The research carried out on the development of the SPO system which combines ANNs with scheduling theory to produce schedules for the soaking-pits /rolling-mill process has concluded that this approach can produce optimal schedules and offers major benefits in terms of flexibility in managing the process, reducing the operational costs and improving utilization of resources. The research on industrial implementation and enhancing the SPO system is ongoing. At this stage, the highest priority is to implement the system for onsite use. This will make further assessments easier and can help to identify more opportunities to improve the system. Research in developing the system further can be carried out in a number of directions.

In the implementation stage, the initial phase will be exploring the possibility of incorporating the SPO system into a commercial scheduling software package. This would reduce the workload in implementing a graphical user interface and provide additional tools.

Another direction is trying alternative scheduling methods. Since the algorithms used by SPO are based on heuristic approaches, although it may produce good schedules, these schedules will not necessarily be optimal. Further improvements by trying alternative heuristic methods or local search algorithms that look for the optimal schedule within the neighbourhood of the SPO schedule should be investigated.

Work done on the grouping module that decides how slabs form pit loads must be carried on. Optimisation methods that are developed to solve similar grouping
problems exist in the literature. They can be implemented by taking into account the characteristics of the problem of forming the pit loads. Distribution of slabs within pits and variance of slab dimensions within a pit load must be taken into account for heating efficiency concerns.

Once the connection with the expert system is established the use of a knowledge-based integrated approach can also be explored. This may have the potential to represent the interactions within the production planning and control function. An overview of this area is given by [192]. Further discussion of scheduling systems using knowledge-based approaches can be found in the survey by Noronha and Sarma [193].

The dynamic programming algorithm used by the SPO system to decide allocation of the pit loads onto pits requires a large amount of memory. If this memory requirement exceeds the internal memory storage of the computer used, the computation time required to find a schedule increases because of the computation power required to transfer data to and from external memory storage. The amount of memory required to keep a record of the cost calculations for every step of the allocations increases exponentially with the number of pits. In the present SPO system the memory requirement was significantly reduced by using a coarser grid of the search space. The present SPO system can produce schedules for 10 soaking pits within a fraction of a minute. Another possible avenue to tackle the problem of memory requirement is to keep this information in a sparse matrix format [190, 191]. At present the records are kept in a matrix format. Because the cost calculations result in zero most of the time and nonzero only if an allocation causes completion time of processing of the pit load to exceed its deadline, most of the elements of the records matrix form blocks of zeros. Sparse matrixes can keep such information in a smaller amount of memory by representing the data in a more compact format.

To estimate the heating and rolling times with better accuracy and resolution than the manual estimations used at the moment, ANN technology is used to develop estimation models. It is concluded that artificial neural networks can constitute a powerful tool for nonlinear modelling and identification of aluminium heating and rolling times. Using ANN models to predict the duration of processes offers major improvements in accuracy of estimations. Using ensemble models also provides a measure of confidence for these estimations and this additional information can be used to make decisions on adjusting the SPO schedules. The performance of the ANN ensemble model developed for estimating the rolling process durations is suitable for including it in the SPO system being implemented to be used onsite. However, further work must be done to improve the performance of the heating time estimators. More
data must be collected to be used in training the heating time estimation model. Data collection and cleaning is the most important part of developing an ANN model. It is important to have correct samples with an even distribution over the whole operation range. Further data collection would make it possible to improve the performance of the rolling time estimation model too.
APPENDIX A – Trial results of the expert system

A.1 Trial UL1

A.1.1 Entry Temperature

<table>
<thead>
<tr>
<th>Plate</th>
<th>Entry Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>UL1-1</td>
<td>465</td>
</tr>
<tr>
<td>UL1-2</td>
<td>439</td>
</tr>
<tr>
<td>UL1-3</td>
<td>416</td>
</tr>
</tbody>
</table>

A.1.2 Gauge Check

The gauge check:

<table>
<thead>
<tr>
<th>Plate</th>
<th>Inter-aim (mm)</th>
<th>Inter-got (mm)</th>
<th>Final-aim (mm)</th>
<th>Final-got (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UL1-1</td>
<td>22.275</td>
<td>21.47</td>
<td>15.00</td>
<td>14.75</td>
</tr>
<tr>
<td>UL1-2</td>
<td>29.889</td>
<td>28.80</td>
<td>20.00</td>
<td>19.48</td>
</tr>
<tr>
<td>UL1-3</td>
<td>35.905</td>
<td>34.90</td>
<td>25.35</td>
<td>24.40</td>
</tr>
</tbody>
</table>

Here: Inter-aim and Final-aim are the ExGauge of the Ipthmas.out;

Inter-got and Final-got are the gauge checks from the Hmas screen;

The inter gauge check:

<table>
<thead>
<tr>
<th>Plate</th>
<th>Inter-aim (mm)</th>
<th>Inter-got (mm)</th>
<th>Error = IA - IG</th>
<th>Inter-RGSet (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>22.275</td>
<td>21.47</td>
<td>0.805</td>
<td>21.087</td>
</tr>
<tr>
<td>2</td>
<td>29.889</td>
<td>28.80</td>
<td>1.089</td>
<td>28.468</td>
</tr>
<tr>
<td>3</td>
<td>35.905</td>
<td>34.90</td>
<td>1.005</td>
<td>34.396</td>
</tr>
</tbody>
</table>

The final gauge check:

<table>
<thead>
<tr>
<th>Plate</th>
<th>Final-aim (mm)</th>
<th>Final-got (mm)</th>
<th>Error = IA - IG (mm)</th>
<th>Final-RGSet (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15.00</td>
<td>14.75</td>
<td>0.25</td>
<td>13.785</td>
</tr>
<tr>
<td>2</td>
<td>20.00</td>
<td>19.48</td>
<td>0.52</td>
<td>18.567</td>
</tr>
<tr>
<td>3</td>
<td>25.35</td>
<td>24.40</td>
<td>0.95</td>
<td>23.955</td>
</tr>
</tbody>
</table>

Here: Error = ExGauge-GaugeCheck = Final-aim-Final-got;

Inter-RGSet and Final-RGSet determined by the expert system.
A.2 Trial UL1-1

A.2.1 Operation Data

<table>
<thead>
<tr>
<th>InGauge (mm)</th>
<th>InLength (mm)</th>
<th>InWidth (mm)</th>
<th>FGMax (mm)</th>
<th>FGMin (mm)</th>
<th>Flength (mm)</th>
<th>Fwidth (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80.0</td>
<td>2778.146</td>
<td>1588.089</td>
<td>15.8</td>
<td>14.2</td>
<td>13638.17</td>
<td>1588.089</td>
</tr>
</tbody>
</table>

A.2.2 ES Data

<table>
<thead>
<tr>
<th>NP</th>
<th>PassType</th>
<th>ARLoad</th>
<th>ExGauge</th>
<th>ExLength</th>
<th>ExTemp</th>
<th>ExWidth</th>
<th>MotorCurr</th>
<th>MillSpeed</th>
<th>RGSet</th>
<th>GCheck</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1079.664</td>
<td>0.067690</td>
<td>3.251</td>
<td>454.56</td>
<td>1.604</td>
<td>921.421</td>
<td>0.5</td>
<td>0.066958</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1132.997</td>
<td>0.057198</td>
<td>3.847</td>
<td>444.64</td>
<td>1.604</td>
<td>2250.211</td>
<td>1.3</td>
<td>0.056355</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1296.599</td>
<td>0.047506</td>
<td>4.632</td>
<td>440.541</td>
<td>1.604</td>
<td>2409.415</td>
<td>1.3</td>
<td>0.046325</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1354.852</td>
<td>0.039554</td>
<td>5.563</td>
<td>435.78</td>
<td>1.604</td>
<td>2231.802</td>
<td>1.3</td>
<td>0.038253</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1304.241</td>
<td>0.032877</td>
<td>6.693</td>
<td>429.599</td>
<td>1.604</td>
<td>1950.506</td>
<td>1.3</td>
<td>0.03168</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1287.706</td>
<td>0.027182</td>
<td>8.095</td>
<td>422.083</td>
<td>1.604</td>
<td>1791.221</td>
<td>1.3</td>
<td>0.026019</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1299.902</td>
<td>0.022275</td>
<td>9.878</td>
<td>413.221</td>
<td>1.604</td>
<td>1728.795</td>
<td>1.3</td>
<td>0.021087</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1340.151</td>
<td>0.018188</td>
<td>12.098</td>
<td>402.925</td>
<td>1.604</td>
<td>1218.127</td>
<td>0.917</td>
<td>0.016917</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1312.926</td>
<td>0.015000</td>
<td>14.669</td>
<td>385.322</td>
<td>1.604</td>
<td>1158.876</td>
<td>0.917</td>
<td>0.013785</td>
<td></td>
</tr>
</tbody>
</table>

Table A1. Trial UL1-1 Lpthmas Output (ES: lpthmas_trial-out1-fig.xls)

Fig A1. Trial UL1-1: Motor Current & ARLoad (ES: lpthmas_trial-out1-fig.xls)
A.3 Trial UL1-2

A.3.1 Operation Data

<table>
<thead>
<tr>
<th>InGauge (mm)</th>
<th>InLength (mm)</th>
<th>InWidth (mm)</th>
<th>FGMax (mm)</th>
<th>FGMin (mm)</th>
<th>Flength (mm)</th>
<th>Fwidth (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80.0</td>
<td>2776.502</td>
<td>1587.149</td>
<td>21.0</td>
<td>19.0</td>
<td>10601.189</td>
<td>1587.149</td>
</tr>
</tbody>
</table>

A.3.2 ES Data

<table>
<thead>
<tr>
<th>NP</th>
<th>PassType</th>
<th>ARLoad</th>
<th>ExGauge</th>
<th>ExLength</th>
<th>ExTemp</th>
<th>ExWidth</th>
<th>MotorCurr</th>
<th>MillSpeed</th>
<th>RGSet</th>
<th>GCheck</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>997.369</td>
<td>3.094</td>
<td>429.798</td>
<td>1.603</td>
<td>725.027</td>
<td>0.5</td>
<td>0.070508</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1050.418</td>
<td>3.581</td>
<td>421.032</td>
<td>1.603</td>
<td>2019.809</td>
<td>1.3</td>
<td>0.060731</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1302.8</td>
<td>4.225</td>
<td>417.775</td>
<td>1.603</td>
<td>2404.66</td>
<td>1.3</td>
<td>0.050848</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1387.668</td>
<td>5.078</td>
<td>414.23</td>
<td>1.603</td>
<td>2418.752</td>
<td>1.3</td>
<td>0.041934</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1457.651</td>
<td>6.101</td>
<td>410.2</td>
<td>1.603</td>
<td>2279.953</td>
<td>1.3</td>
<td>0.034528</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1412.666</td>
<td>7.357</td>
<td>404.871</td>
<td>1.603</td>
<td>2030.259</td>
<td>1.3</td>
<td>0.028468</td>
<td>0.02880</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1401.181</td>
<td>8.934</td>
<td>398.336</td>
<td>1.603</td>
<td>1898.181</td>
<td>1.3</td>
<td>0.023216</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1418.931</td>
<td>10.995</td>
<td>390.544</td>
<td>1.603</td>
<td>1324.169</td>
<td>0.917</td>
<td>0.018567</td>
<td>0.01948</td>
<td></td>
</tr>
</tbody>
</table>

Table A3. Trial UL1-2 LptHmas Output (ES: lpthmas_trial-out2-fig.xls)

![ARLoad graph][1]

ARLoad (Trial2)

![MotorCurrent graph][2]

MotorCurrent (Trial2)

Fig A2. Trial UL1-2: Motor Current & ARLoad (ES: lpthmas_trial-out2-fig.xls)
# A.4 Trial UL1-3

## A.4.1 Operation Data

<table>
<thead>
<tr>
<th>InGauge (mm)</th>
<th>InLength (mm)</th>
<th>InWidth (mm)</th>
<th>FGMax (mm)</th>
<th>FGMin (mm)</th>
<th>Flength (mm)</th>
<th>Fwidth (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80.0</td>
<td>2775.047</td>
<td>1586.318</td>
<td>26.11</td>
<td>24.59</td>
<td>8779.24</td>
<td>1586.318</td>
</tr>
</tbody>
</table>

## A.4.2 ES Data

<table>
<thead>
<tr>
<th>NP</th>
<th>PassType</th>
<th>ARLoad</th>
<th>ExGauge</th>
<th>ExLength</th>
<th>ExTemp</th>
<th>ExWidth</th>
<th>MotorCurr</th>
<th>MillSpeed</th>
<th>RGSet</th>
<th>GCheck</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1288.048</td>
<td>0.068345</td>
<td>3.216</td>
<td>408.302</td>
<td>1.602</td>
<td>1069.638</td>
<td>0.5</td>
<td>0.067180</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1345.824</td>
<td>0.058495</td>
<td>3.757</td>
<td>401.038</td>
<td>1.602</td>
<td>2594.179</td>
<td>1.3</td>
<td>0.057211</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1461.091</td>
<td>0.050267</td>
<td>4.372</td>
<td>398.725</td>
<td>1.602</td>
<td>2509.709</td>
<td>1.3</td>
<td>0.048747</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1383.871</td>
<td>0.042484</td>
<td>5.173</td>
<td>395.374</td>
<td>1.602</td>
<td>2267.693</td>
<td>1.3</td>
<td>0.041122</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1455.229</td>
<td>0.035905</td>
<td>6.121</td>
<td>391.667</td>
<td>1.602</td>
<td>2164.414</td>
<td>1.3</td>
<td>0.034396</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1413.572</td>
<td>0.030255</td>
<td>7.264</td>
<td>386.863</td>
<td>1.602</td>
<td>1947.214</td>
<td>1.3</td>
<td>0.028832</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1400.090</td>
<td>0.025350</td>
<td>8.670</td>
<td>381.041</td>
<td>1.602</td>
<td>1825.335</td>
<td>1.3</td>
<td>0.023955</td>
<td></td>
</tr>
</tbody>
</table>

Table A5. Trial UL1-3 LptHmas Output (ES: lpthmas_trial-out3-fig.xls)

![ARLoad (Trial3)](ARLoad_Trial3.png)

![MotorCurrent (Trial3)](MotorCurrent_Trial3.png)

Fig A3. Trial UL1-3: Motor Current & ARLoad (ES: lpthmas_trial-out3-fig.xls)
APPENDIX B - ANN Types

1 Supervised
   1. Feedforward
      - Linear
        - Adaline - Widrow and Hoff (1960), Faussett (1994)
        - Functional Link - Pao (1989)
      - Backprop - Rumelhart, Hinton, and Williams (1986)
      - Cascade Correlation - Fahlman and Lebiere (1990), Faussett (1994)
      - Quickprop - Fahlman (1989)
      - RPROP - Riedmiller and Braun (1993)
      - OLS: Orthogonal Least Squares - Chen, Cowan and Grant (1991)
      - CMAC: Cerebellar Model Articulation Controller - Albus (1975), Brown and Harris (1994)
      - Classification only
      - Regression only
        - GNN: General Regression Neural Network - Specht (1991), Nadaraya (1964), Watson (1964)
      - Recurrent time series
        - Backpropagation through time - Werbos (1990)
        - Elman - Elman (1990)
        - Jordan - Jordan (1986)
        - TDNN: Time Delay NN - Lang, Waibel and Hinton (1990)
   3. Competitive
      - ARTMAP - Carpenter, Grossberg and Reynolds (1991)
      - Fuzzy ARTMAP - Carpenter, Grossberg, Markuzon, Reynolds and Rosen (1992), Kasuba (1993)
      - Gaussian ARTMAP - Williamson (1995)
      - Adaptive resonance theory
        - ART 2-A - Carpenter, Grossberg and Rosen (1991a)
        - ART 3 - Carpenter and Grossberg (1990)
        - Fuzzy ART - Carpenter, Grossberg and Rosen (1991b)
   1. Competitive
      - Vector Quantization
        - Grossberg - Grossberg (1976)
        - Kohonen - Kohonen (1984)
        - Conscience - Desieno (1988)
      - Self-Organizing Map
        - GTM - Bishop, Svensén and Williams (1997)
        - Local Linear - Mulier and Cherkassky (1995)
      - Adaptive resonance theory
        - ART 2-A - Carpenter, Grossberg and Rosen (1991a)
        - ART 3 - Carpenter and Grossberg (1990)
        - Fuzzy ART - Carpenter, Grossberg and Rosen (1991b)
      - Oja - Oja (1989)
      - Sanger - Sanger (1989)
      - Differential Hebbian - Kosko (1992)
   3. Autoassociation
      - Linear autoassociator - Anderson et al. (1977), Faussett (1994)
      - BSB: Brain State in a Box - Anderson et al. (1977), Faussett (1994)
   3. Nonlearning
      2. various networks for optimization - Cichocki and Unbehauen (1993)
References


8. LF Siong, I Postlethwaite, Development of an expert system for supervisory control of a rolling mill, final year project report, University of Leicester, 1999.


17. F Sanfilippo, F Martini, M Oriati, Advanced integrated plant-product management system for rolling mills, Iron and Steel Engineer, 73/1, 17—21, Jan 1996.


22. TRS Rao, EA Upton, EA Rupar, R Ellis, Optimization of ingot heating in soaking pits, Iron and Steel Engineer, 61/10, 34-38, 1984


66 JD Birdwell, Future directions in computer-aided control system design software development, *IEEE Control System Magazine*, 3/1, 11-14, 1983.


68 HA Barker, IT Harvey, PW Grant, CP Jobling, Object-oriented data representation for computer aided control engineering, *Preprints of the IFAC World Congress*, 3, 281-284, 1993.


SPECIAL NOTE

THE FOLLOWING IMAGE IS OF POOR QUALITY DUE TO THE ORIGINAL DOCUMENT.

THE BEST AVAILABLE IMAGE HAS BEEN ACHIEVED.
http://www.spss.com/spssbi/clementine/
http://www.rulequest.com/see5-info.html

J Ross, Quinlan: C4.5: Programs for Machine Learning, Morgan Kaufmann 1993.
W McCulloch, W Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics, 5, 115-133, 1943.


152 SM Johnson, Optimal two- and three-stage production schedules with set-up times included, Naval Research Logistics Quarterly, 1, 61-68, 1954.


184 A Vignier, G Venturini, Resolution of some 2-stage hybrid flow shop scheduling problems. 5th International Workshop on Project Management and Scheduling, Poznan, Poland, 1996.


