SELECTING A MIX OF DISPATCHING RULES FOR A JOB SHOP BY USING ARTIFICIAL NEURAL NETWORKS

By

Pramit Shah

Bachelor of Production Engineering, Mumbai University, June 1997

A thesis presented to the Ryerson University

in fulfillment of the

thesis requirement for the degree of

Master of Applied Science

in

Mechanical and Industrial Engineering

Toronto,Ontario,Canada, 2004

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis.

I authorize the Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Pramit Shah

I further authorize the Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Pramit Shah

Borrower's page

Ryerson University requires the signature of all persons using or photocopying this thesis. Please sign below, and give address and date.

Name	Address	Date

ABSTRACT

Selecting a Mix of Dispatching Rules For a Job Shop by Using

Artificial Neural Networks

@ Publisher Pramit Shah, 2004

Master of Applied Science

In the program of Mechanical and Industrial Engineering

Ryerson University

Dispatching rules are a popular and commonly researched technique for scheduling tasks in job shops. Much of the past research has looked at the performance of various dispatching rules when a single rule is applied in common on all machines However, better schedules can frequently be obtained if the machines are allowed to use different rules from one another. This research investigates an intelligent system that selects dispatching rules to use on each machine in the shop, based on a statistical description of the routings, processing times and mix of the jobs to be processed. Randomly generated problems are scheduled using permutations of three different dispatching rules on five machines. A neural network is then trained by using a commercial package to associate the statistical description of each problem with its best solution. Once trained, a network is able to recommend for new problems a dispatching rule to use on each machine. Two networks were trained separately for minimizing makespan and the total flowtime in the job shop. Test results showed that the combination of dispatching rules suggested by the trained networks produced better results for both objectives than the alternative of using the one identical rule on all machines.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who contributed to this work directly as well as indirectly.

To work with Dr. Ahmed El-bouri was a lifetime opportunity. His trust in his students, continuous encouragement, strong support, insightful guidance makes a student perform far better than he is. I sincerely wish to thank him for all the help provided to me. I really enjoyed the exchange of ideas that helped me to formulate this work. Thank you. Sir!

The financial support through Professor Ahmed El-bouri of Department of Mechanical Engineering- Ryerson University is greatly appreciated.

My family had an indirect contribution to this research. Especially I would like to thank my wife, Miloni for her patience, understanding and emotional support. Thank you from the bottom of my heart!

I am also thankful to all my fellow graduate students specially Akram and Jayesh, for keeping my spirit up.

And above all, I thank God for giving me the strength and determination.

To my parents and Miloni

TABLE OF CONTENTS

AUTHOR'S DECLARATION----------*ii*

ABSTRACT----------*iv*

ACKNOWLEDGEMENTS----------*v*

TABLE OF CONTENTS----------*vii*

LIST OF FIGURES----------*xii*

LIST OF TABLES----------*xiv*

NOTATIONS----------*xvi*

GLOSSARY----------*xvii*

Chapter 1: Introduction and Background

1.1 Introduction----------1

1.2 Problem Definition----------5

1.3 Dispatching Rules----------7

1.4 Thesis Outline----------9

Chapter 2: Literature Review

2.1 Literature Review----------10

Chapter 3: Artificial Neural Networks----------22

3.1 Introduction to Artificial Neural Networks (ANN)----------22

3.1.1 Basic Concepts----------22

3.1.2 Architecture of ANN----------23

3.1.3 Classification of Neural Network Architectures ----------25

3.1.4 Feed Forward Neural Network----------25

3.1.5 Learning rule----------26

3.2 Neural Network Training by Back Propagation (BPNN)----------27

3.2.1 Scaling Function----------28

3.2.2 Transfer Function----------28

3.2.3 Delta Rule----------29

3.2.4 Type of Datasets----------29

3.2.5 Learning Rate and Momentum----------30

3.2.6 Initial Weights----------31

3.2.7 Training BPNN----------31

Chapter 4: Methodology and Design of BPNN----------33

4.1 Design of a BPNN for Job Shop Scheduling----------33

4.2 Problem Analysis----------33

4.3 Data Acquisition and Preparation ----------35

4.3.1 Input Variable Selection----------35

4.3.2 Construction of Input Vector----------37

4.3.3 Output Layer----------41

4.3.4 Data Collection----------41

4.3.5 Validation of Data Set Output----------43

4.3.6 Data Sorting----------46

4.4 Design of Neural Network----------49

4.4.1 Hidden Layer----------49

4.4.2 Hidden Units (Neurons)----------50

4.4.3 Training Stopping Criteria----------51

4.4.4 Training Procedure----------53

4.4.5 To Find Optimal Number of Hidden Neurons (Units)----------53

4.5 Training BPNN----------55

4.6 Implementation----------56

4.7 BPNN Model for Minimizing Mean Flowtime Criteria----------59

Chapter 5: Analysis and Discussion----------62

5.1 Analysis of Trained Network----------62

5.1.1 Neural Network Weights----------62

5.2 Test Problems ----------62

5.3 Test results for Minimizing Makespan ----------63

5.4 Test results for Minimizing Mean Flowtime----------64

5.5 Further Discussion on Results ----------68

5.6 Analysis of Variance (ANOVA) ----------69

5.6.1 Assumptions----------72

5.6.2 Data Representation----------72

5.6.3 General Logic - Analysis of Variance----------73

5.6.4 ANOVA Calculations ----------74

5.6.5 Hypothesis----------75

5.7 ANOVA Results for Makespan----------75

5.8 ANOVA Results for Mean Flowtime----------78

Chapter 6: Conclusion and Future research ----------80

6.1 Summary----------80

6.2 Conclusions----------81

6.3 Future Research ----------82

References----------84

Appendix A: C++ Programs

Appendix A1: Problem Generation----------A1

Appendix A2: Training Set Generation ----------A3

Appendix A3: Comparison of BPPNN Results with Other Dispatching Rule Results ----------A13

Appendix A4: Test Problem Generation ----------A17

Appendix B: Arena Simulation Model----------B1

Appendix C: Experimental Setup

Appendix C1: Determining Number of Hidden Neurons for Makespan----------C1

Appendix C2: Determining Number of Hidden Neurons for Mean Flowtime ---C2

Appendix D: Neural Weights

Appendix D1: Makespan Network----------D1

Appendix D2: Mean Flowtime Network----------D9

Appendix E: Comparison Results

Appendix E1: Comparison of BPNN with makespans from optimal rule combinations for n = 10 to 100 ----------E1

Appendix E2: Comparison of BPNN with mean flowtimes from optimal rule combinations for n = 10 to 100 ----------E8

LIST OF FIGURES

Figure 1.1 Schematic layout of a job shop, illustrating the routes followed by two different jobs ----------2

Figure 2.1 Degree of artificial intelligence required by different scheduling approaches ----------15

Figure 3.1 Basic features of a single biological neuron ----------23

Figure 3.2 Processing elements ----------24

Figure 3.3 Classification of common neural network architectures ----------25

Figure 3.4 Multi-layer feed forward neural network ----------26

Figure 4.1 Design methodology for building ANN----------34

Figure 4.2 Structure of ANN proposed for the 5-machine job shop ----------44

Figure 4.3 Details of a hidden node ----------50

Figure 4.4 Effect of number of hidden neurons on the minimum average error for makespan ----------54

Figure 4.5 BPNN generalization vs. minimum average error for makespan----------57

Figure 4.6 Effect of number of hidden neurons on the minimum average error for mean flowtime ----------60

Figure 5.1 BPNN makespans compared to makespans from optimal rule combinations for n=50 ----------65

Figure 5.2 BPNN mean flowtime compared to mean flowtime from optimal rule combinations for n=50----------67

Figure 5.3 Comparison of BPNN, SPT, LPT and MWKR with respect to makespan of optimal rule combinations ----------70

Figure 5.4 Comparison of BPNN, SPT, WINQ+PT and LWKR with respect to mean flowtime of optimal rule combinations ----------71

LIST OF TABLES

Table 1.1 Characteristics of the job shop problem considered----------9

Table 4.1 Raw data of example problem N10M265----------38

Table 4.2 Determining inputs 1 to 5 for example problem NM10265---------- ----------38

Table 4.3 Inputs 6 to 10 for example problem N10M265----------39

Table 4.4 Example of normalized value for inputs 11 through 15 for problem N10M265----------40

Table 4.5 15-unit input vector representing job shop problem of Table 4.1----------40

Table 4.6 Optimal rule combinations for example problem N10M265----------42

Table 4.7 Numerical representation for optimal rule combinations of Table 4.6----------43

Table 4.8 15-unit vector representing desired output for example problem N10M265----------43

Table 4.9 Sample of input and output pattern pairs ----------45

Table 4.10 Attributes used in job shop Arena model----------46

Table 4.11 Input and output patterns of Table 4.9 after sorting ----------48

Table 4.12 Total makespans using BPNN with various number of hidden neurons ------54

Table 4.13 Training and generalization results for minimizing makespan ----------56

Table 4.14 A 20-job example problem ----------57

Table 4.15 Input vector for example problem of Table 4.14 ----------58

Table 4.16 BPNN output for example problem of Table 4.14 ----------58

Table 4.17 Total mean flowtimes for BPNN with various numbers of hidden neurons ----------59

Table 4.18 Mean flowtime BPNN output for example problem of Table 4.14----------60

Table 5.1 Summary of total makespan in sets of 50 test problems for various n ----------64

Table 5.2 Results for individual problems for n=50 (set no. 9 in Table 5.1) ----------65

Table 5.3 Summary of total mean flowtime results for test sets consisting of 50 problems each ----------66

Table 5.4 Mean flowtime for individual test problems for n=50 (set no. 9 in Table 5.3) ----------67

Table 5.5 Experimental data for ANOVA (average makespan)----------76

Table 5.6 ANOVA results for the performance objective of minimizing makespan -----76

Table 5.7 LSD calculations for the minimization of makespan experiment ----------77

Table 5.8 ANOVA experimental data (average mean flowtime) ----------78

Table 5.9 ANOVA results for the performance objective of minimizing mean flowtime ----------78

Table 5.10 LSD calculations for the mean flowtime experiment ----------79

NOTATIONS

m	Number of machines
i	Subscript for jobs, i = 1,2,3,.....n.
j	Subscript for operation, j = 1,2,3,.....m.
k	Subscript for machines, k = 1,2,3....m.
λ_i	Time of arrival of job on the shop floor.
P_{ijk}	Processing time of operation j of job i on machine k.
Q_{ik}	Number of operations j required to complete job i
n_i	Number of input neurons.
n_o	Number of output neurons.
R_k	Mean routing order on machine k.
C_i	Completion time of job i.
C_{max}	Maximum completion time or makespan.
F_i	Flow time of job i.
σ	Standard deviation.
$Var\ (X)$	Variance
K	Number of treatments for ANOVA experiments.
$\overline{X_{..}}$	Grand mean.
$\overline{X_{j.}}$	Group mean.

GLOSSARY

Dispatching rules

Rules based on the smallest processing time or least work remaining

LWKR: - Select the job with the least total work remaining work

SPT:- Select the job with the least intermediate processing time

TPT: - Select the job with least total processing time

Rules based on the largest processing time or most work content

LPT:- Select the job with the most intermediate processing time

LOPNR: - Select the job whose least number of operations remaining

MOPNR: - Select the job whose most number of operations remaining.

MWKR:- Select the job with the most total work remaining work

MWKR-P:- Jobs ranked by MWKR after the present scheduling operation

MWKR/P: - Jobs ranked by the greatest ratio of total remaining work to processing time of schedulable operation

Rules based on due date

DS: - Select the job with the smallest slack, where slack is the due date less the total work remaining. This is also called as the dynamic slack.

DDT / EDD: - Jobs ranked according to their earliest due date

SLK/RO: - Select the job wit the smallest ratio of its dynamic slack to the number of remaining operations

Rule based on arrival

AT: - Select the job based on earliest arrival time

FIFO: - First in first out

LIFO: - Last in first out

RAN: - Select the job randomly

Rule based on queue status

WINQ: - Select the job based whose workload in the next queue is least

NINQ: - Select the job based on the number of jobs waiting in the queue of the next machine.

CHAPTER 1

Introduction and Background

1.1 Introduction

In real world applications, one of the most important aspects of competitiveness and success of an organization is efficient production management and timely decisions. Such decisions are often constrained by specific objectives and requirements. These constraints are frequently incompatible in nature, and as the number of such constraints increases, the decision process becomes more complex. Scheduling decisions are an important component of overall operational control in any manufacturing system.

The decision-making requirements for scheduling production systems have been a fascinating research topic for more than three decades. The goal of scheduling is to ensure that production objectives are met optimally. Researchers have developed numerous models and methodologies for the planning and scheduling control of production activities for different manufacturing layouts. Two of the most researched models are flow shop and job shop.

A flow shop is a model in which machines are arranged in series. In this system, jobs flow from an initial machine (resource), through several intermediate machines, and ultimately to a final machine. A job may skip a particular machine, but all jobs must be processed according to a fixed route. The job shop differs from the flow shop in one important aspect: the flow of work is not unidirectional, meaning that jobs need not visit the machines in the same order. The problem in the job shop is to schedule n jobs on m

machines such that one or more performance objectives is optimized. Each job requires *j* number of operations (tasks) that are to be processed on these machines. Generally, a job may not require all the machines and it may visit the same machine more than once.

A scheduling problem in a job shop is defined by several attributes, namely, the number of jobs; the number of machines; the processing order of the jobs on the machines (or routing); processing time of each job on different machines; job arrival times, and last but not the least, the performance objective to be achieved. Figure 1.1 displays a general job shop model. It illustrates the routes taken by two different jobs through the shop. In this instance both the jobs visit machine number *three* after completion of two operations as per their respective job routes. Supposing Job 1 completes both its operations prior to job 2, then job 2 may have to wait in the machine buffer (or queue) if the machine is still busy with job 1.

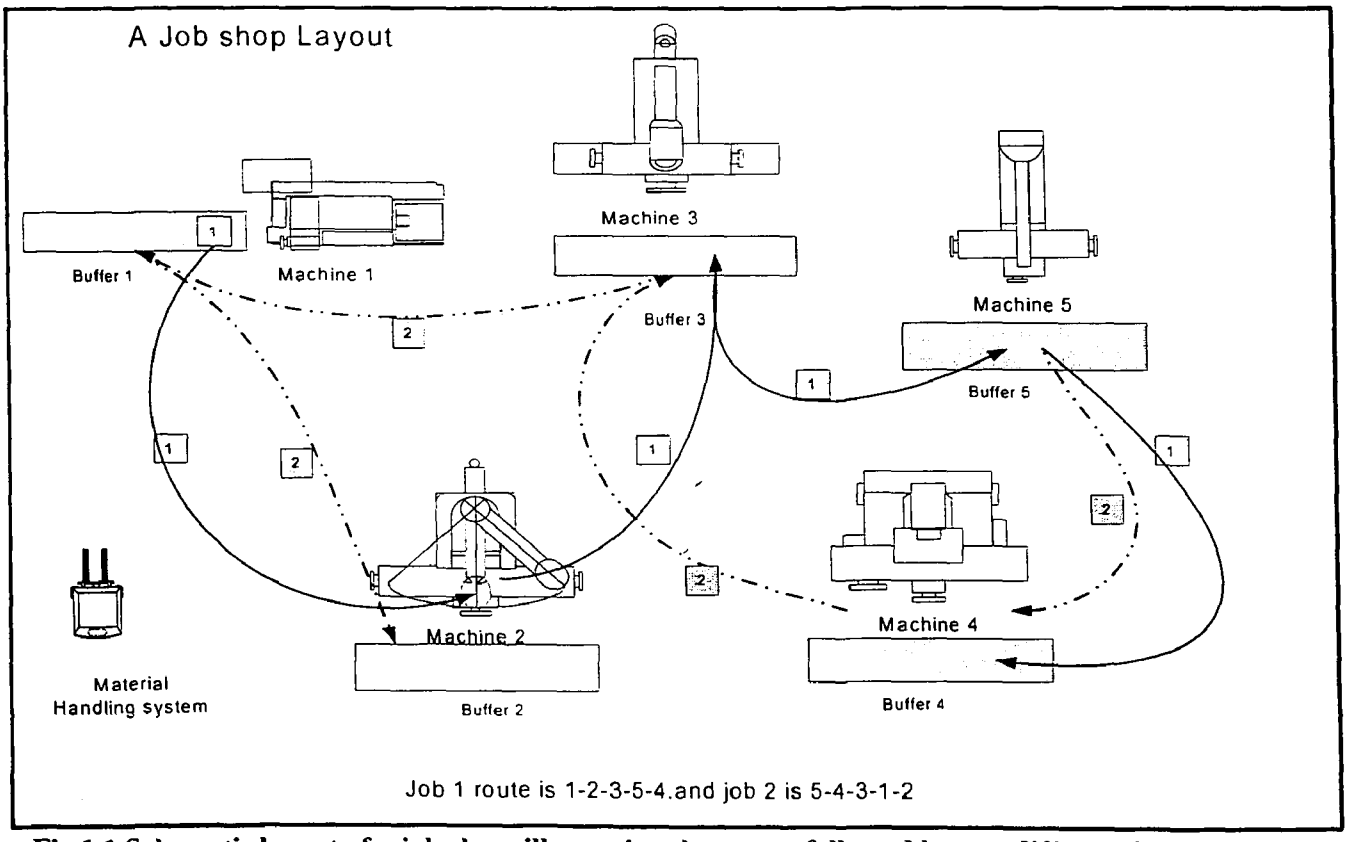


Fig 1.1 Schematic layout of a job shop, illustrating the routes followed by two different jobs.

For scheduling n jobs on a single machine there are n! possible schedules, but in the case of a job shop the number of possible combinations is $(n!)^m$, when each job has one and only one operation on each machine (Conway, 1967). For example with 10 jobs on 5 machines there are 6.92×10^{32} possible number of schedules. Thus, job shop scheduling is a NP-hard problem (Pinedo, 1995) and, consequently, it is not practical to search for a global optimal solution other than for very small problems.

A general job shop problem can be either static or dynamic. The static job shop assumes that all jobs are available at the beginning of the planning phase. On the other hand, the dynamic job shop problem assumes that the arrival times of the jobs are unknown, and that jobs arrive at random intervals throughout the production cycle. A schedule can be evaluated by its ability to meet production objectives like minimizing the make span (or completion time), the mean flow time and mean tardiness. A schedule that results in minimum completion time for a particular problem does not necessarily minimize the mean flowtime in that problem. Hence, the aim of the scheduling is to find a schedule for processing all jobs, such that one or more specified performance measures are optimized.

A number of job shop scheduling techniques have been developed for solving static and dynamic problems. Among these are the dispatching rules, which are a popular and commonly researched technique for scheduling tasks in a job shop. A dispatching rule (DR) is used to select the next job to be processed from a set of jobs waiting in the queue (or buffer) when a machine becomes free. The simplicity and ease in application of dispatching rules (DRs) have made them a practical tool for scheduling in the real world. However, there are some shortcomings in the use of dispatching rules. First, none of the dispatching rules dominates the others for the important performance criteria like mean

flow time, mean tardiness, etc. This implies that DRs are problem dependent, and one DR that gives a good result for one problem may not give an equally good result in another. Due to the dynamic and changing characteristics of the jobs as they are processed, a job shop may be able to meet the performance objectives better by judiciously changing the DR on individual machines over time, or by using a combination of different rules for the machines.

As a result of the scheduling complexity in modern manufacturing systems, Artificial Intelligence (AI) techniques have been considered as a scheduling decision tool; one technique that has shown promise is the Artificial Neural Network (ANN). In simple terms, Artificial Neural Networks apply knowledge gained from past experience to new problems or situations. An ANN looks for patterns in "training" sets of data, learns these patterns and develops the ability to correctly classify new patterns. This approach in a job shop that uses dispatching rules requires the neural network to be prepared by selecting a set of training examples for different performance measures, and finding from simulation (or other) studies the optimal rules to use in these examples. Information from these optimal solutions is then used to "teach" the network to select the most appropriate dispatching rule from several candidate / rules. Once this procedure is completed, the trained neural network is capable of providing faster solutions to new problems which were previously unseen by the network.

This research addresses the problem of an appropriate selection of dispatching rules (in this case from three alternatives) to use for each of the machines in a job shop by using artificial neural networks. Two separate artificial neural networks are developed for the performance criteria of minimizing makespan and minimizing mean flowtime. Each of these criteria have a different set of competing DRs to choose from.

1.2 Problem Definition

A job shop problem of interest is a 5-machine job shop. In actual manufacturing systems, particularly those that follow a group technology concept, the number of machines used can be reasonably expected to be between 3 to 10. Also, it seems to be a consensus among researchers that a four-machine job shop is adequate to represent the complexity involved in a large job shop (Kiran et al, 1984). Thus, the number of machines chosen for this study is 5 machines, with number of jobs ranging from 10 jobs to a maximum of 100.

In this study, a static job shop is considered with arrival time $\lambda_i = 0$ for all the jobs. Thus, all the jobs that are to be processed are available at the start of the scheduling period. It has been assumed that all the jobs have two sets of attributes. The first one is the work flow Q_{ik} pattern (or route), that is machine k's order in job i's route. At the beginning of the schedule, each job's flow pattern is defined with the condition that each job visits all machines once. In other words, each job has a specific precedence order of operations, which has to be followed before exiting from the system. For example, a specific job i may have the precedence order $4 \longrightarrow 5 \longrightarrow 3 \longrightarrow 2 \longrightarrow 1$, indicating that

job i's first operation must be done on machine 4, its second operation on machine 5, and so on.

The second characteristic is the processing time (P_{ijk}) representing the j^{th} operation of job i on machine k. The time needed by each operation j of job i on machine k is known in advance. In this study, deterministic processing times for these operations are generated from the discrete uniform distribution U (10, 99) integer time units. For

example, a specific job on 5 machines may have processing times of 45-95-61-20-35 corresponding to the machines on the job's route.

Once operation j of job i has been completed, it will be transferred with the help of a material handling system to the next machine, as per the job's route for the next operation $j+1$ if that machine is free, or to a buffer for that machine otherwise. It has been assumed that the transfer time between machines is negligible and that the material handling system is always available whenever required. Hence, jobs are either in process on a machine, or waiting in a buffer for processing.

"Schedules are generally evaluated by aggregate quantities that involve information about all jobs, resulting in one-dimensional performance measures" (Baker, 2002). The following two performance measures *or* criteria *or* objectives are considered in this study.

a) Makespan: The makespan measures the total time taken by a given schedule to complete the set of available jobs. In other words, it is the time at which the last job exits from the shop. This measure is defined, for a sequence of n jobs, by

$$C_{max} = \max_i\{C_i\} \quad \ldots. \quad (2.1)$$

Where C_i = Completion time of job i = 1,2,3.... n

The objective of scheduling the jobs in a way that minimizes the makespan is an important one, because it reduces throughput time for processing a batch of different jobs.

b) Mean Flowtime: Flowtime represents the total time spent in a job shop. This includes actual machine processing time plus time spent waiting in buffers.

The mean flowtime is given by:

$$\overline{F} = \frac{1}{n}\sum_{i=1}^{n} F_i \quad \ldots. \quad (2.2)$$

Where F_i = Flow time or the time spent by job i in the system

The objective of minimizing the mean flowtime is a common objective in scheduling, because it acts to reduce work-in process and inventory levels.

The job shop model considered in this study is based on the following explicit assumptions:

- Jobs are independent and consist of strictly ordered operational sequences (or job routes). Furthermore, all jobs have equal weights (importance).
- Job pre-emption or cancellation is not allowed. Once the processing of any operation has started on a machine, it cannot be interrupted before its completion, and then resumed at a later time.
- There is only one machine of each type in the shop, and operations for two different jobs cannot be processed simultaneously on the same machine.
- Set up time is negligible.
- An operation may not begin until its predecessor is complete.
- Each machine is continuously available for production; machine breakdown or downtime is not considered.
- There are no alternate routes for any job.
- The buffer capacity is unlimited and machine blocking does not occur.

1.3 Dispatching Rules (DRs)

Dispatching rules or (priority rules) refer to the procedure used to prioritize the jobs that are waiting in queue for a machine. The dispatching rule therefore picks the next job

to load on the resources or machine. There are more than 100 dispatching rules surveyed by Panwalkar and Iskander, (1977). Dispatching rules use job specific information such as processing time, due date, remaining number of operations, etc. The rules considered in this study are:

a) **Rules based on shortest processing time**

- Shortest processing time (SPT): - Priority is given to the job with the shortest immediate processing time (i.e., smallest processing time on the current resource).
- Least work remaining (LWKR): - Priority is given to the job with the least sum of remaining operation processing times.

b) **Rules based on Longest processing time**

- Longest processing time (LPT): - Priority is given to the job with the longest immediate processing time (i.e., largest processing time on the current resource).
- Most work remaining (MWKR) – Priority is given to the job with the total work remaining processing times.

c) **Rule based on Queue status**

- Work in the next queue (WINQ + PT): - Priority is given to the job with the least workload in the next queue it will visit, *plus* the processing time P_{ijk} on the current resource.

The Following Table1.1 summarizes the job shop problem parameters considered in this research.

Number of jobs	From 10 to 100
Number of machines	5
Job arrival times	All jobs are available at the start of schedule
Flow pattern	Process routes
Processing time	Deterministic integers drawn from U (10,99)
Performance measures	1) Minimizing makespan 2) Minimizing mean flowtime
Dispatching rules	SPT, LPT, MWKR, LWKR, WINQ+PT
Job shop scheduling technique	Selection of dispatching rules by a trained neural network

Table 1.1 Characteristic of the job shop problem under consideration.

1.4 Thesis outline

The rest of the thesis is organized as follows: Chapter 2 provides a detailed literature review on the dispatching rules, neural networks for scheduling, and past research in selecting combinations of dispatching rules by different techniques. Chapter 3 presents a background introduction to neural networks. Chapter 4 provides a step-by-step guideline on building a feed forward back-propagated supervised neural network for selecting dispatching rule combinations. Chapter 5 analyses the output generated from the trained network and examines the generalization capability of the trained network. Chapter 6 provides a summary and a conclusion, and discusses further research scope.

CHAPTER 2

Literature Review

2.1 Literature Review

Scheduling jobs is an important aspect of a job shop manufacturing system environment, for it can have a deep impact over the system's performance efficiency. Job shop scheduling has been studied extensively over the last three decades. Many approaches have been developed to solve the static job shop scheduling problem, and some of the well-known approaches can be found in French, (1982) and Pinedo, (1995).

However, for a practical application, scheduling decisions are usually taken in real time considering existing constraints. Some of such constraints are the state of the shop floor (e.g., availability of resources), characteristics of the production program (e.g., part routing, due date of jobs) and production objectives (e.g., minimizing makespan) to be achieved. Nevertheless, some of the uncertain variables such as breakdown / failures of resources, new jobs prompted during a production cycle, etc. also need to be considered. For these reasons many researchers tend to approach the job shop scheduling issue through the acceptance of heuristic dispatching rules rather than seeking a deterministic optimal solution to the problem.

This literature review reports on various job shop scheduling approaches. Both Jones and Rabelo, (1996) and Blazewicz et al, (1996) have done detailed surveys on various job shop scheduling techniques, which can be categorized mainly as either exact or approximation methods.

The category of exact methods includes all the mathematical models. These guarantee optimal solutions for a job shop problem. However, their application is limited to a smaller numbers of jobs and resources. This is due to some of the limitations like computational requirements for obtaining an optimal solution, difficulty in formulation of material flow constraints, etc. Also, the development in computational power of the computer has sharply improved the use of such approaches, but nevertheless its utilization remains limited.

Branch and bound is one such mathematical technique that deals with NP- hardness of scheduling problems by decomposition into smaller sub problems that may be solved for optimality. Blazewicz et al, (1996) presented a detailed discussion on the success and the limitations of this method in job shop problems.

The category of approximation methods includes numerous algorithms and techniques that are developed for producing good solutions, which can be reasonably close to optimal results. These techniques can be further categorized as dispatching / priority rules, Artificial Intelligence (AI) techniques and other heuristic methods. They are used either to obtain a best sequence of jobs for the desired performance objective, or to select from various dispatching rules ones to apply on the machines, based on the current or prevailing conditions.

One of the most common approaches to dynamically schedule jobs is to use dispatching rules (DRs). These rules are sometimes called scheduling rules, or priority rules. They are defined by Blackstone et al, (1982) as a "Rule used to select the next job to process from jobs awaiting service." Dispatching rules are widely used in practice and a considerable body of research exists because of their ease of implementation and their substantially reduced computational requirements.

The most well known and comprehensive survey of scheduling heuristics is by Panwalker and Iskander (1977) where more than 100 dispatching rules were presented, reviewed and classified based on their processing time, arrival time, queue status, etc. A survey of 34 dispatching rules could also be found in Blackstone et al, (1982). A common conclusion found in both surveys is that no single priority rule dominates and provides consistently best results for different job shop situations. There have also been many instances where combinations of rules have been successfully used in job shop scheduling (Blackstone et al, 1982). This approach has two or more dispatching rules dynamically selected for each of the machines based on the shop floor's prevailing conditions.

Review of literature related to dispatching rules in job shop schedules reveals a focus either to introduce new dispatching rules to optimize the shop floor performance or to review and test the existing ones, both for different shop configurations and performance objectives. Advanced simulation tools have been widely adopted for this purpose. For instance, in the simulation study done by Waikar et al, (1995), ten different dispatching rules (FIFO, SPT, DDT, LWKR, MWKR, MWKR-P, MWKR/P, MOPNR, SLK/RO and RAN) were tested for different shop loads ranging from 70 to 85 %, with job arrival and processing times following exponential and normal distributions respectively. Waikar et al, (1995) considered two different sets of performance criteria based on the flow-time and tardiness. The results of this study showed that both SPT and LWKR perform well under different shop loads not only for mean flowtime, which resulted in lowering in process inventories, but also for total queue time and time spent in the system.

On the other hand, a comparative study done by Chang et al, (1996) evaluated the performance of 42 different dispatching rules using a linear programming model. These rules were broadly categorized in six different categories based on the smallest processing time, the largest processing time, due date, number of operations, random rule and lastly on the queue status. In order to evaluate all the dispatching rules, seven different performance objectives were considered, which were further divided into two sets, based on completion time and tardiness. Their analysis indicated that the shortest processing time (SPT) related rules consistently performed well, while the longest processing time based rules consistently performed badly.

Similarly, Rajender and Holthaus, (1999) carried out two different types of comparative studies of dispatching rules in both flow shops and job shops. In the first study, operations were performed on all the machines, but missing operations were allowed for both the shops in the second study. The job shop studied had *ten* machines, number of operations ranging from (2 ...10), processing times uniformly distributed between (1,49), shop loads ranging from 80 to 95% and exponentially distributed interarrival time. In their study, *thirteen* different dispatching rules such as (FIFO, AT, EDD, SPT, PT+WINQ, etc) consisting of both existing and new rules proposed by Rajender and Holthaus, (1999) were evaluated for *seven* different performance objectives. The objectives were based either on flow time or on tardiness. The results showed that for the mean flowtime criterion SPT, PT+WINQ and RR (a rule by Raghu and Rajendran, 1993) performed consistently well under different shop loads. For the same criteria with missing operations, PT+WINQ rules emerged on average to be the best. The study also showed that for a higher shop load PT+WINQ performs significantly better than the SPT rule.

Almost all the papers cited above either introduced new dispatching rules or tested existing rules based on a validated simulation model developed to test the rule itself. Thus, simulation is one of the most common tools widely used by many researchers for testing new or existing dispatching rules.

Although for many years the only practical approximation methods were priority/ dispatching rules, the introduction of more powerful computers, as well as an emphasis on carefully designed, analyzed and implemented algorithms has allowed more novel approaches to be developed for solving job shop problems. One example of such an approach is the use of artificial intelligence (AI). AI is a sub field of computer science that is concerned with integrating biological and computer intelligence. It has fundamental origins from biological understanding and uses principles in nature to find solutions for various complex problems. There are a number of classes of AI techniques, some of which are expert/knowledge-based system, neural networks (training and learning), fuzzy logic, genetic algorithm search, etc.

As the development in solving job shop scheduling problems increased, the degree of intelligence and the knowledge required for solving such problems also increased. Figure 2.1 presents the degree of artificial intelligence required by different scheduling approaches (Sim et al, 1994).

Expert / knowledge based systems mainly consist of knowledge and an inference engine to operate on that knowledge base. The application of such systems can be seen in

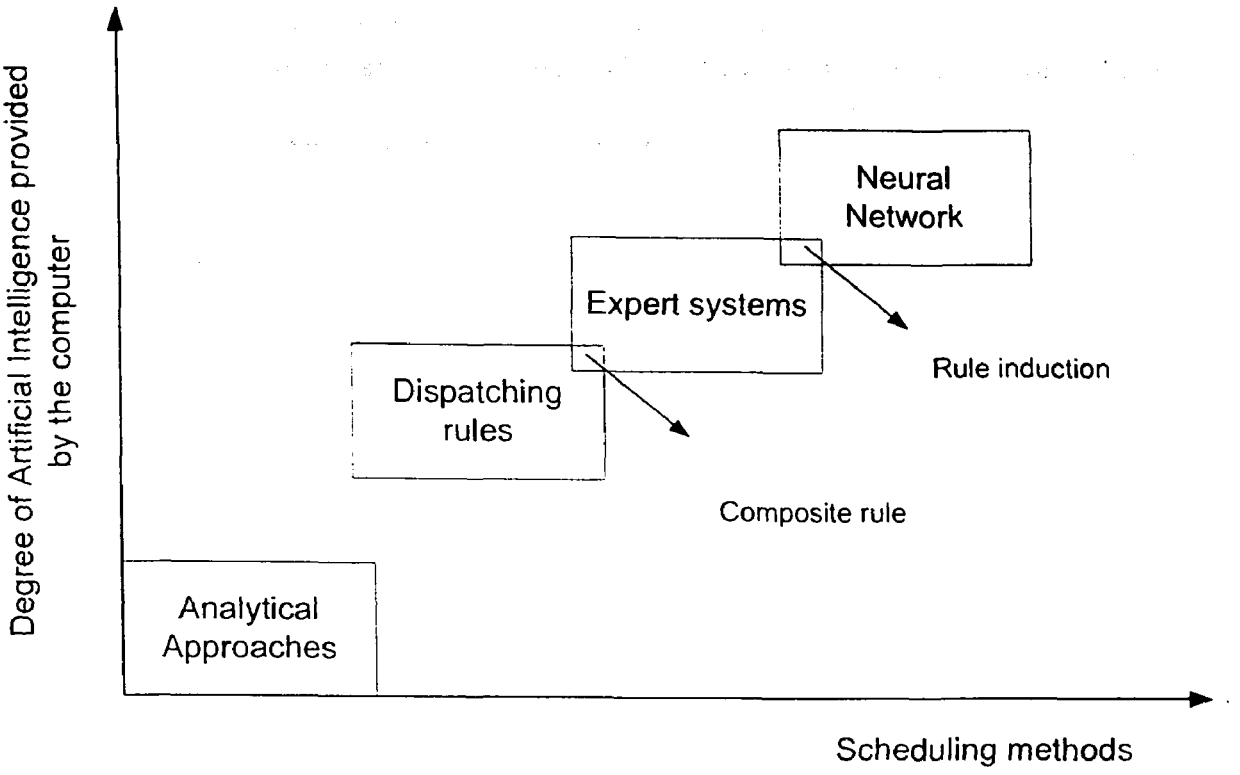


Fig. 2.1 Degree of artificial intelligence required by different scheduling approaches (Sim et al, 1994).

Pierreval, (1992), who demonstrated its use for selecting priority rules in a *two* machine flow shop model of a flexible manufacturing system, with the shop load of 83%. *Two* different dispatching rules (SPT and EDD) for *five* different criteria (based on flow time and tardiness) were examined. The results showed that expert system (ES) provided on average good results for the performance criteria of mean flowtime and average waiting time as compared to SPT on both machines. Also, ES achieved the best performance among SPT-EDD and SPT-SPT combinations on the two machines for mean tardiness criteria.

Sim et al, (1994) developed an expert neural system in order to overcome some of the limitations with the expert system and for solving job shop problems for objectives related to tardiness. The artificial neural network was based on the back propagation feed forward neural network (BPNN) with the generalized delta rule as a learning algorithm and the sigmoid curve as the activation function. This model consisted of an input array

of 14 neurons, sixteen sub-networks embedded in expert system and a single output node, which determined jobs to be processed first, based on the lowest output value. The job shop had *nine* resources, job arrivals following a Poisson distribution with an average rate of 0.6 to 0.775. Each job had 3-6 operations and processing time of 1-4 time units per operation and the performance objectives of minimizing lateness and tardiness. The first 10 input neurons represented different dispatching rules (such as SPT, LPT, EDD, etc) and the remaining neurons represented the arrival rate of jobs in the neural network. The results showed that the performance of this system was able to match the performance of the best dispatching rule used in training for both the performance objectives. Although the BPNN required a lengthy training process, once trained, the network only requires a single forward pass of computation (from the input nodes to the output node) to schedule.

Production demands are often cyclic in nature, and if the pattern of such demands can be recognized, then systems can respond to seasonal and sudden changes. Thus, a system that is able to recognize such patterns is in a position to update scheduling decisions effectively. Artificial neural networks, fuzzy logic and genetic algorithm are some of the methods that can be used in order to develop such systems.

Fuzzy logic application in dynamic selection of dispatching rules was examined by Subramaniam et al, (2000) for the performance objective of minimizing makespan in *ten*-job, *ten*-machine job shop problems. The proposed approach carefully selects *three* normalized input units, based on the conditions prevailing in the job shop, and a fuzzy scheduler selects an appropriate dispatching rule from the available candidates SPT, WINQ, MWKR to use for the individual machines. The results showed that the best makespan was obtained with the use of a combination of dispatching rules as compared to using a single dispatching rule such as SPT, LPT, MWKR, LWKR, FIFO, LIFO,

MOPNR, LOPNR, NINQ, WINQ. The fuzzy scheduler is a one-pass approach and requires the same order of computation time as the simple dispatching rule.

On the other hand, both Kumar & Srinivsan, (1996) and Chryssoluris & Subramaniam, (2001) used genetic algorithms (GA) for dynamic selection of dispatching rules in job shop scheduling problems. In the former, a genetic search procedure for a real world combinatorial optimization problem was considered. The job shop had *eighty* jobs and *fifty-nine* machines, with the number of operations ranging from (2, 37). Some jobs could visit a machine more than once. Genetic algorithms use the idea of survival of the fittest by progressively accepting better solutions to the problems. In this case randomly generated strings of dispatching rules had a length of *ten*. The GA method generated a better makespan as compared to using any one of the seven different dispatching rules (SPT, LPT, TPT, RPT, DS, EDD, RS, FIFO) on all machines. The proposed algorithm yielded an improvement of about 3% in makespan over the best (SPT) among the seven rules tested, but the computation time required by the genetic algorithm (998 sec) was very large as compared to generating makespan by using a single dispatching rule (3.32 sec).

In the GA study by Chryssoluris & Subramaniam, (2001), the dynamic job shop had *six* machines, fixed job arrival times, from *three* to *six* operations and processing time ranging uniformly from U [1,100] and uniformly distributed due date of U [-100,1500] (loose) and U [-100,500] (tight). The study also considered machine breakdowns and repair, and jobs could visit the same resource more than once. The performance objectives were minimizing cost and tardiness. The proposed GA outperformed the other dispatching rules. Varying due dates did not seem to have any effect on the relative

variation of the results. The computation time for this GA approach was about 2 orders of magnitude larger than for a simple dispatching rule.

An artificial neural network (ANN) has the capability to recognize and learn new patterns to generalize for any measurable function. ANN has been employed in a number of real world applications in manufacturing, finance, stock market, medical field, national security, etc. Zhang and Hung, (1995) provided a detailed survey on the neural networks in manufacturing with the applications in the areas of process planning, quality assurance, engineering design, scheduling, process control, etc.

There are various types of neural networks proposed and developed for solving scheduling problems. It has been observed in the literature that job shop scheduling problems were solved by neural networks either to obtain optimal sequence of jobs or to make a dynamic selection of dispatching rules based on the prevailing conditions of shop floor; so that the desired performance objective could be satisfied. For the purpose of research review, the various types of neural network that are of interest, viz:

1) Hopfield network and other optimizing networks

2) Multi-layer preceptrons (Back propagation networks)

Sabunguoglu, (1998) presented a detailed survey on using neural networks exclusively for scheduling applications. He proposed two different classifications, based on the types of neural networks used and the application area.

Hopfield networks were the first type of artificial neural network used for solving job shop scheduling problems. Foo and Takeji, (1988a, 1988b) proposed a Hopfield network with only input and output neurons. They mapped the problem on a two dimensional matrix of neurons with (nm +1) rows and (mn) columns, where m is the number of machines and n is the number of jobs. A Simulated annealing process was then

applied to the model in order to force the network out of local minima. The method was successful for a 4-job, 3-machine problem, but it had several limitations, such as the number of jobs must be greater than the number of machines. The proposed network in this method is not practical for large size problems and there is no guarantee of an optimal solution. Furthermore, computation time, even for a small problem is excessive.

Satake et al, (1994) used a Hopfield network for minimizing the makespan in a job shop by obtaining an optimal sequence of jobs. In this case, a Boltzmann mechanism was used in order to avoid local minima. Various problems were considered ranging from 4-jobs/3-machines to 10-jobs/10-machines. The proposed network produced optimal or near optimal schedules within a reasonable amount of time. Further development in the application of Hopfield networks for solving a job shop problem can be obtained from (Sabunguoglu, 1998) and (Jain and Meeran, 1998).

The conclusions drawn from the review of Hopfield networks are that they require an excessive number of neurons and interconnections and can get easily trapped in local minima. For these reasons, they are suitable only for small size problems.

It has been observed in the literature that back propagation (BP) networks have drawn the attention of many researchers. One of the reasons for this interest is that BP network provides an increased speed for the selection process that may be needed in real world applications. Jones and Rableo, (1990) were the first to use the back propagated neural network in a proposed integrated (expert and ANN) scheduling system for ranking a set of dispatching rules based on the current shop status and job characteristics such as job types, arrival patterns, process plans. The output of the neural network was evaluated by an expert system, which then generated a schedule for the performance objective of

minimizing tardiness. In their results, the BP network was able to predict correct results 90% of the time.

Pierreval et al, (1992) used both neural networks and simulation for selecting dispatching rules dynamically in a *two*-machine flow shop. In this study, *five* different dispatching rules and *five* different performance criteria were considered based on DR were considered. In this proposed back propagated neural network, there were 4 input units representing mean arrival rate of jobs, mean expected processing time on each machine and processing time variance, and a total of 19 output units representing a combination of *five* dispatching rules on both the machines. A trial and error approach determined 16 hidden neurons for a single hidden layer. A total of 500 training sets were used for training the network. The trained network was capable of selecting the dispatching rules based on the performance criteria. In a comparison of results between neural network (NN) and simulation, the authors highlighted that NN had the advantage of computational time over simulation for decision making in real time scheduling and, also, memory required by NN was less as compared to the simulations using SIMAN IV and GPSS packages. In a comparison between NN and expert system, the expert system required expertise to develop larger knowledge bases, and this was difficult to obtain in the case of selection of scheduling heuristics. The learning capabilities of NN avoid these problems.

During the literature review it has been observed that although neural networks were used in selection of dispatching rules in flow shops, there have been no investigations of their use in dynamic selection of combinations of dispatching rules in job shops.

Thus, in the current research an artificial neural network to select combinations of dispatching rules is investigated. The task of the neural network is to pick an appropriate DR from a number of alternatives, given an instantaneous environment in the job shop. (Pierreval, 1992) Also, it is desired to test the suggestion of and Sabunguoglo, (1998) regarding the generalization property of back propagated networks for solving large size problems, having learned to solve small size problems.

CHAPTER 3

Artificial Neural Networks

The objective of this chapter is to provide background information on artificial neural network, types of neural networks and a detailed description of a back propagated neural network (BPNN).

3.1 Introduction to Artificial Neural Networks (ANN)

A job shop problem can be solved by several methodologies including mathematical programming, simulation, priority/dispatching rules, expert system, artificial intelligence (AI) etc. An Artificial Neural Network (ANN), which is one of the AI techniques, is an information-processing paradigm. The inspiration of using neural networks lies in its ability to extract information from complex data, similar to the biological nervous systems, and how the brain processes information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in harmony to solve specific problems. ANN systems, like people, learn by examples and dynamically modify themselves to fit the data presented.

3.1.1 Basic Concepts

The basic building block of any ANN is the neuron, which is the fundamental cell of the brain or simply the processing unit of our brain. These neurons have three principal

components: dendrites, the cell body and an axon, which are presented in Fig 3.1. The dendrites are tree like receptive networks of nerve fibers that carry electrical signals into the cell body. The cell body effectively sums and thresholds these incoming signals. The axon is a single long fiber that carries the signal from the cell body out of other neurons. The point of contact between axon of one cell and a dendrite of another cell is called a synapse. Learning process in the brain occurs due to the strengthening and weakening of synapses (Reinhardt, 1990).

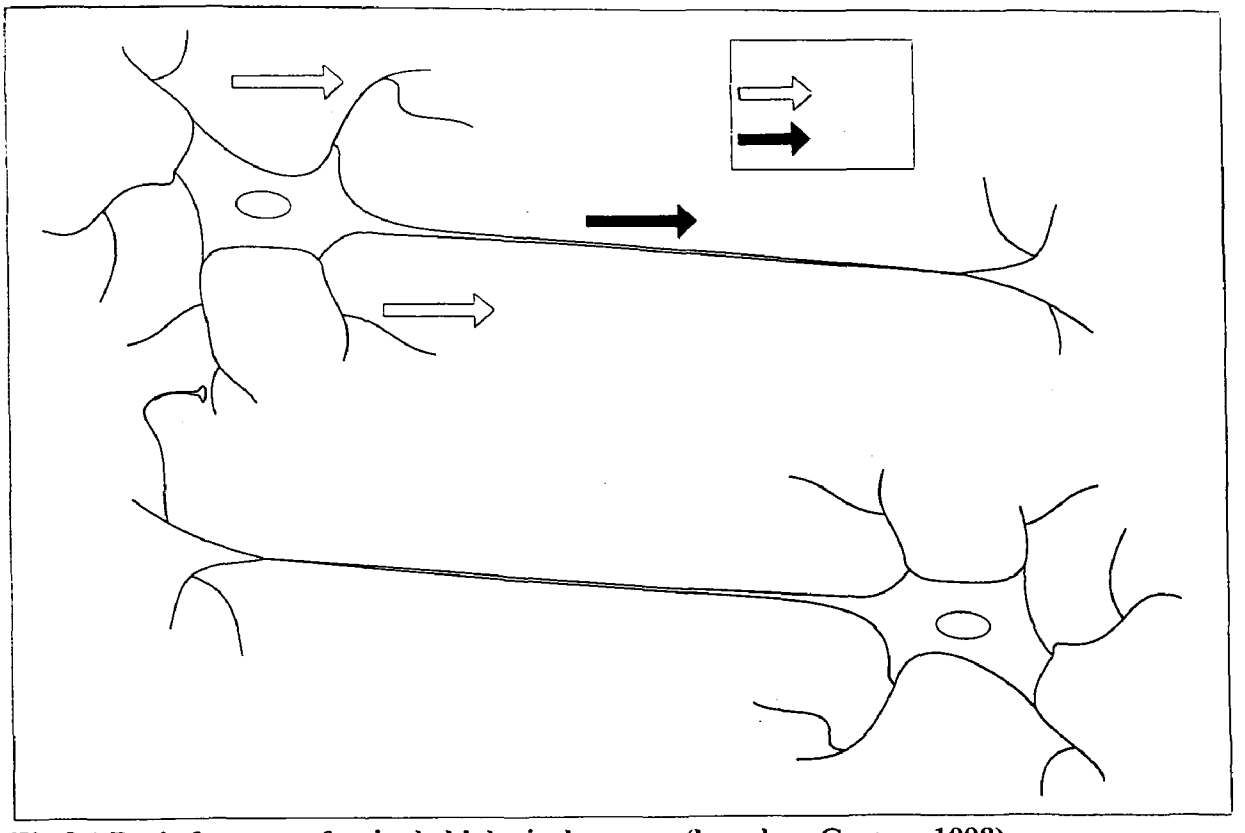


Fig 3.1 Basic features of a single biological neuron (based on Garson, 1998).

3.1.2 Architecture of ANN

ANNs are composed of basic units called processing elements (PE), which are also called as nodes or neurons. For example, in the back propagation (feed forward) network the PE are arranged in layers referred to as input, hidden and output layers. Each PE

receives an input X_a from every other neuron a, which is multiplied by corresponding weights. The aggregate input signal to the PE, Net, is calculated as the sum of these weighted inputs. (Smith, 1999).

The resulting signal is then passed through an activation function, which could be linear, logistic, step, ramp, sigmoidal or hyperbolic tangent depending on the problem to be solved. The output O_b of the PE neuron is therefore $O_b = f(\text{Net})$.

In simple terms, neurons multiply an input by a set of weights, then combine these weighted inputs into an internal activity level by adding them together. The resulting signal is then modified by the transfer function of the PE to produce the output. Figure 3.2 shows the structure of a PE. The power of neural computation comes from the immense number of interconnections between the PEs, which share the load of the overall processing task, and also from the adaptive nature of the parameters (weights) that interconnect the PEs. (Garson, 1998).

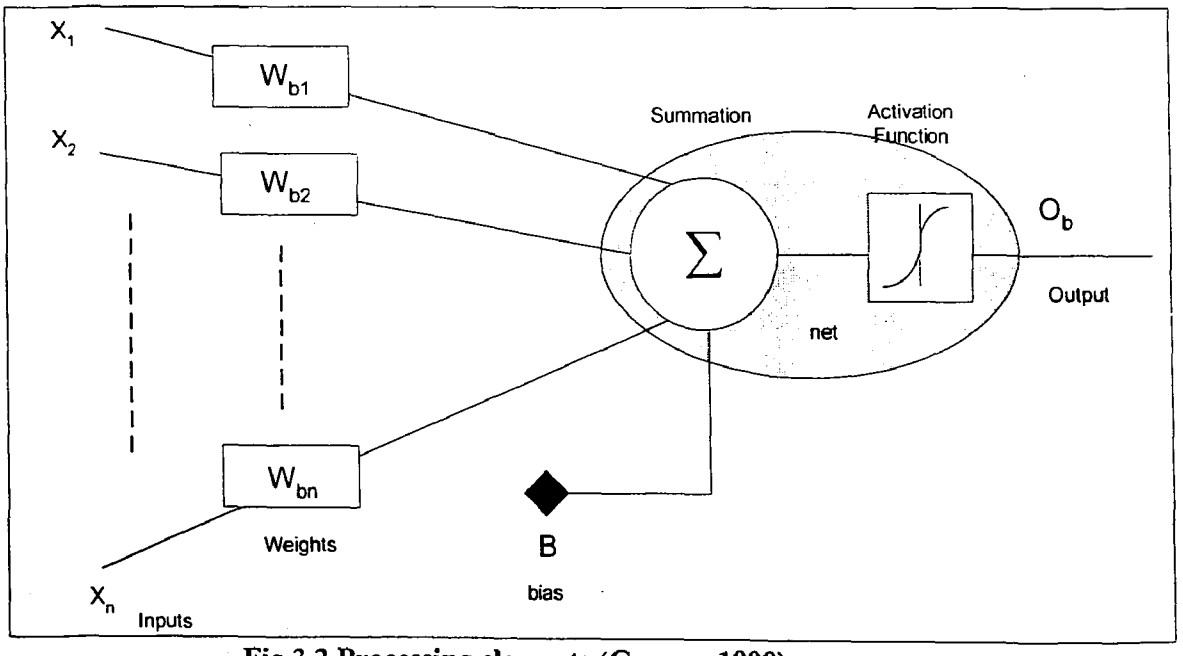


Fig 3.2 Processing elements (Garson, 1998).

3.1.3 Classification of Neural Network Architectures

There are several neural network architectures, which are used not only for solving job shop scheduling, but also for several other applications. Figure.3.3 provides classifications within the field of neural computing. Jain and Meeran (1998) have presented details about each of the different types of neural networks. There are several different neural architectural models, but most of these can be divided into two main categories viz. feed forward and feedback.

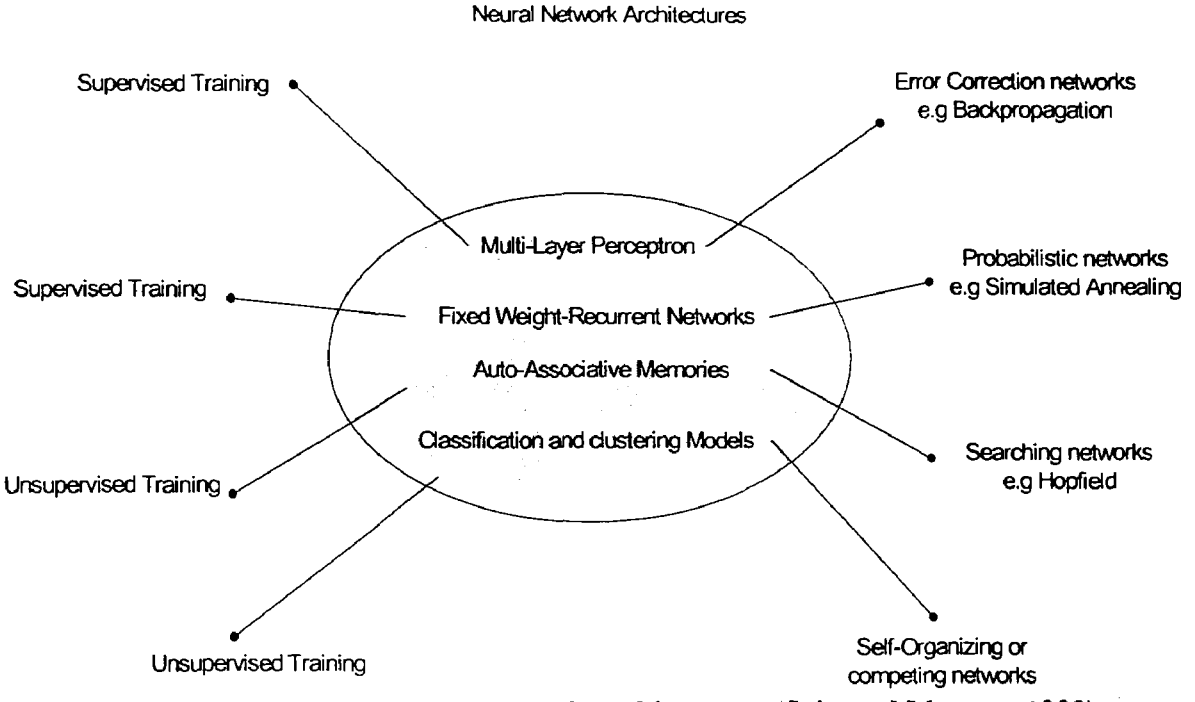


Fig.3.3 Classification of common neural network architectures (Jain and Meeran, 1998).

3.1.4 Feed Forward Neural Network

Processing elements are usually organized in layers. These layers can be connected in a number of different ways. A neural network with more than one layer is called a multi-layer neural network. Figure 3.4 shows the architecture of a multi-layer neural network. As can be seen in this network, the output from each layer feeds the next layer

of units in a forward direction and there is no feedback connection between the layers, that is, the information is processed from left to right only. Garson (1998) describes this network as one that "has one or more inputs which are propagated through one or more number of hidden layers. Each layer contains a variable number of nodes, which finally reaches the output layer containing one or more output nodes".

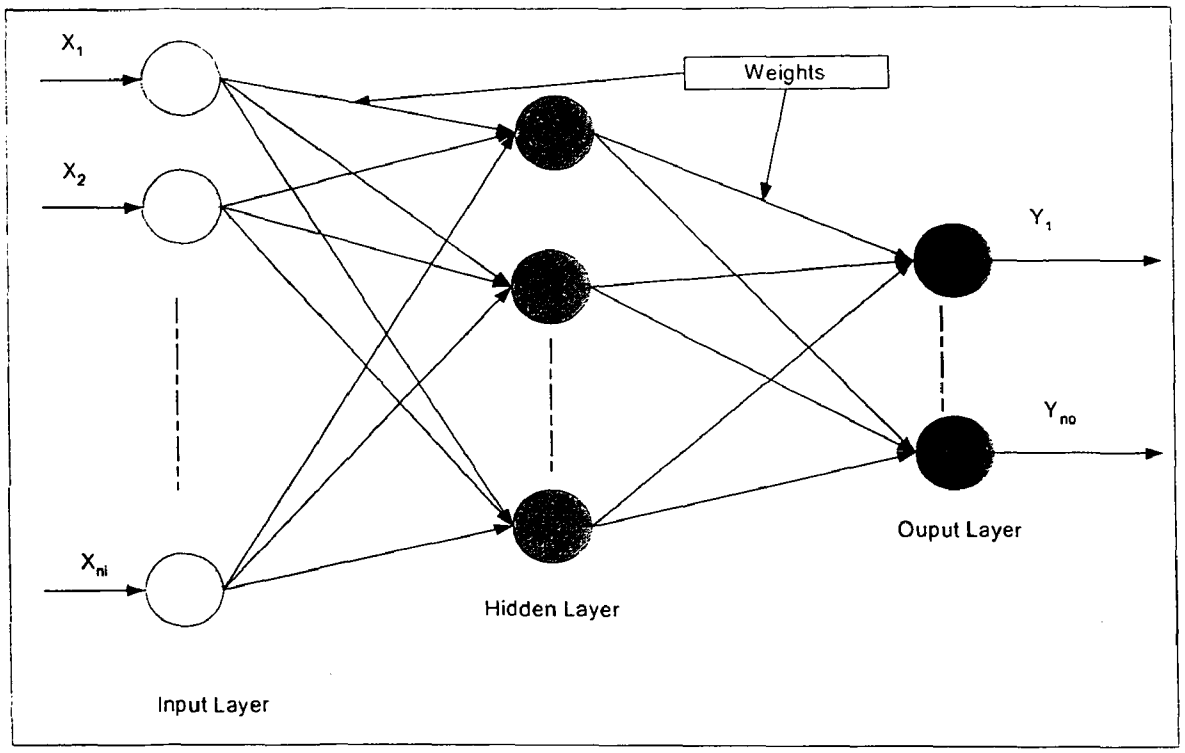


Fig 3.4 Multi-layer feed forward neural network (based on Garson, 1998).

3.1.5 Learning Rule

There are several learning rules, such as Hebb's rule, the Delta rule, Gradient Descent rule, etc. The choice of learning rule depends to some extent on the chosen architecture (Smith, 1999). For instance, in case of back propagation, the Delta rule is mostly used for generalization of errors. Learning rules can be divided into two main types: supervised and unsupervised learning (Smith, 1999).

a) Supervised learning in simple terms is defined as 'learning with teacher' (Smith, 1999). In this case, the network is trained with the correct/desired responses for given inputs. During the learning process global information may be required. Usually, supervised learning is done off-line i.e. the network can be trained separately and later on the trained network is used for obtaining solutions to new problems (Garson, 1998).

b) Unsupervised learning. In this situation there is no external teacher used by the neural network for training. In other words, the desired output for the input is either unknown or does not exist, and learning is based on local information. Usually unsupervised learning is performed on-line (Smith, 1999).

3.2 Neural Network training by Back Propagation (BPNN)

Back propagation Neural Networks are a class of feed forward neural networks with supervised learning rules. In order to train a neural network to perform a certain task the weights of each unit must be adjusted in such a way that the error between the desired output and the actual output is reduced. The actual response of the network is subtracted from a target response to produce an error signal. The derivates of the output error are passed back to the hidden layer using the original weighted connections. Each hidden node then calculates the weighted sum of the back-propagated errors to find its indirect contribution to the known output errors. After each output and hidden node finds its error values, the weights are adjusted to reduce the errors. This process requires that the neural network compute the error derivate of the weights.

The back propagation algorithm is the most widely used method for determining error derivates of the weights. For the present research different BPNN networks are built for different performance objectives. There are a few important parameters and

guidelines required while building the BPNN network, the details of which are provided below:

3.2.1 Scaling Function

Scaling function is used in the input layer to scale the data from their numeric range into a range that the network deals with efficiently. In case of supervised feed forward back propagation neural network the output patterns also require normalized or scaled value in the same range as of input. Common ranges for scaling are either 0 to1 or –1 to 1. In this study both the input and output patterns for neural network training are scaled between 0 and 1(Neuroshell 2 manual; Swingler, 1999).

3.2.2 Transfer Function

In simple terms, the transfer function, which is also called as transformation, squashing, activation or threshold is a mathematical formula that determines the output of a processing neuron. In most areas of research the Sigmoid or S-shaped function is more popular as compared to other functions such as hyperbolic, tangent, step, ramping, arc tan or linear. For sigmoid function, the output varies continuously but not linearly as the input changes. According to Swingler (1999), the activation functions are "necessary to introduce non-linearity in the network. This non-linearity makes it possible to learn non-linear functions". Sigmoid functions are more inclined to vary an output than threshold functions, because threshold functions usually are not sensitive to small changes in weights. For this reason, sigmoid functions are used in neuron activations for the proposed BPNN for job shop dispatching rules selection model (Garson, 1998).

3.2.3 Delta Rule

"For a given input vector, the output vector is compared to the correct answer. If the difference is zero no learning takes place; otherwise, the weights are adjusted to reduce this difference" (Garson, 1998). The change in weight w for output layer neuron *a* with respect to input layer neuron *b* is the learning rate r times the activation function (f) for neuron *a* times *c*, the correct desired output of neuron *a*:

$$\Delta w_{ab} = r f_{ab} c_a \quad \ldots \quad (3.1)$$

C_a is the difference between the expected and actual output. During this learning the delta weight as shown in the equation is added to the existing weight w_{ab}. More details about the rule can be obtained from (Garson, 1998).

3.2.4 Type of Datasets

It is important to define and discuss the various types and sets of data. Data for neural networks are an input-output model. "Inputs are the presumed predictor variables, while outputs are the neural model's estimates of the dependent variable or variables" (Garson, 1998). In simple terms the data set is a set of examples for learning, which is nothing, but a "training set". The neural modeling software uses this set to compute model weights. Garson, (1998) defined the test data set as "the set of data to which the final neural model, and its associated weights, is applied for purposes of generalization."

There is a distinction in types of data sets between test sets and validation sets. According to Garson (1998), a test set is a collection of data that is used for testing the performance measure without changing any of the parameters, while a validation data set is used to tune the parameters. However, Garson, (1998) also mentioned that the test data

sets are used for validation in the final stage of development of a neural model and thus, test data sets are sometime referred to as validation data sets. In this study only training and testing data sets are considered.

3.2.5 Learning Rate and Momentum

The learning rate (η) controls the magnitude of the adjustments to the network's weights in response to the error. Each time a pattern is presented to the network, the weights leading to an output node are modified so as to produce a smaller error the next time the same pattern is presented. The magnitude of the weight adjustments is determined by the relationship: learning rate times the error. In most neural packages the learning rate is between 0.1 and 1. In Neuroshell 2 the default value is 0.1. A high learning rate might lead the local minimum to be overstepped constantly, causing oscillation from side to side, and never reaching convergence to the lower error state (Garson, 1998).

One way to allow faster learning without oscillation is to make the weight change a function of the previous weight change in order to provide a smoothing effect. The momentum (α) factor determines the proportion of the last weight change that is added into the new weight change (Garson, 1998). The Momentum, which causes the weight changes to be affected by the size of previous weight changes, is used to avoid local minima. Typically α is selected in the range of $0 \leq \alpha \leq 1$" (Swingler, 1999). The default momentum value in Neuroshell 2 is 0.1.

3.2.6 Initial Weights

All network weights must be set to initial values before training starts. If weights are too large then the network might become unstable and the nodes saturated, and if weights are too small then weight changes will be slow. The choice of initial weights is dependent upon the problem and normalization of variables (Swingler, 1999). Generally, weights are chosen randomly. The default initial weight value in Neuroshell 2 is 0.3.

3.2.7 Neural Network Training

Garson, (1998) defines training as, "the process of refining the weights in a neural model through a process in which training data set are fed into the model, analyzed and reprocessed through a number of iterations." The objective of training a neural network is to adjust the weights so that application of a set of inputs produces the desired set of outputs. Training assumes that each input vector is associated with an output vector.

In brief, training a feed-forward neural network with supervised learning consists of the following procedure, adapted from Garson (1998).

- Select a training pair from the training set and apply the input vector to the network input.
- Calculate network output using a forward pass. Calculation of network output is accomplished by using a feed-forward process and application of an activation function for each layer in the network.
- Compute the difference between network output and the desired target value from the training pair output value.
- Change the network weights in a way that minimizes the error.

This chapter has provided a detailed description of BPNN along with different parameters to be considered while designing BPNN for solving the job shop problem for two different performance objectives. The next chapter describes building a BPNN for selecting appropriate dispatching rules in a 5-machine job shop.

CHAPTER 4

Methodology and Design of BPNN

This chapter introduces the methodology for constructing a back propagation neural network (BPNN) that selects a combination of dispatching rules to use in a job shop problem.

4.1 Designing a BPNN for job shop scheduling

In order to build a BPNN, there are some common steps that should be followed for achieving desired objectives. Baily and Thompson, (1993) provided detailed guidance on how to build neural networks along with the design decisions to be considered for commonly used neural paradigms like BPNN, Boltzman machines, Hopfield network, etc. Likewise, Kaastra and Boyd, (1996) provided a step-by-step guide for building a BPNN forecasting model. Based upon both these references, Figure. 4.1 is constructed to highlight the main steps required in designing a BPNN. The procedure is not usually a single pass process and some of the steps like training and testing may need to be carried out several times.

The material in this section is organized based on the design steps of Figure. 4.1. The detailed explanation is presented for a BPNN with regards to the performance criterion of minimizing makespan. This same procedure can be used for other performance criteria, as will be illustrated for minimizing the mean flowtime in section 4.7 of this chapter.

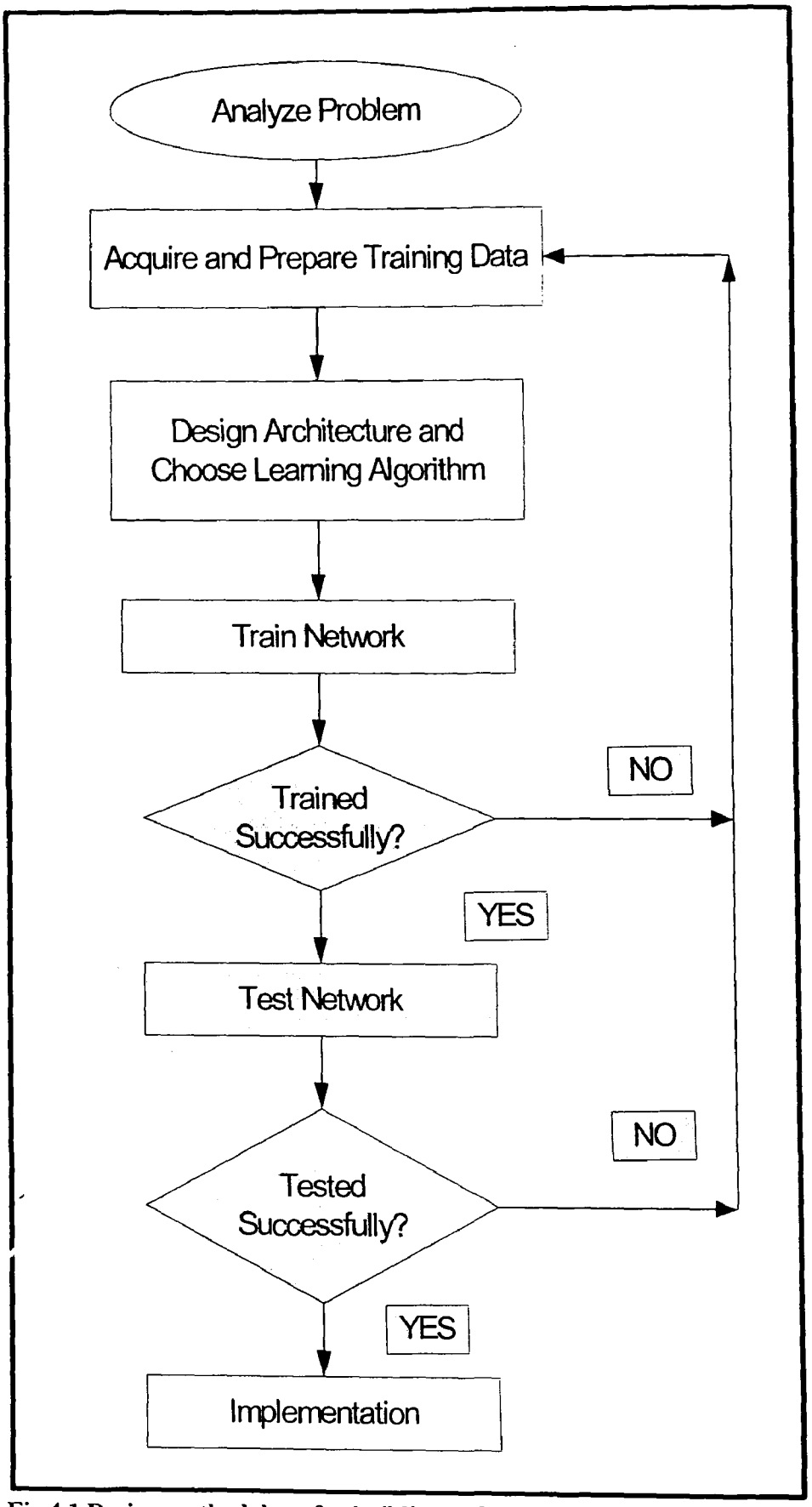


Fig 4.1 Design methodology for building ANN.

4.2 Problem Analysis

The definition, nature and scope of the job shop problem along with the type of neural network recommended for solving it have been discussed in chapter one. The case of a 5-machine job shop is considered, although a similar approach can be used for job shops having different numbers of machines. The objective is to select for each machine, one of three alternative dispatching rules so that a given performance criterion is optimized for the problem on hand.

4.3 Data Acquisition and Preparation

In building an effective ANN, the researcher has to decide what kind of data are to be used, namely either historical or constructed data. If historical data are being used, then they must be sorted or filtered from noisy data, if any. A format and range for each input and output are then selected and the data are expressed in a manner that can be presented to the BPNN. This usually takes form of a data vector as follows:

<Input 1, input 2, input 3,.....input N_i, output 1, output 2, output 3.....output N_o>

Where N_i and N_o represents number of input and output nodes respectively.

Before the data are collected, it is necessary to decide which of the job shop attributes will be used in the input layer and what output information is desired of the network.

4.3.1 Input Variable Selection

Input variables must be selected very carefully, taking into account the given performance objective to be achieved. Here, the objective considered is to minimize

makespan, which mainly depends on the job's processing times (P_{ijk}) and the routes (Q_{ik}). However, in a case where every job visits each machine once, the subscript j can be dropped and the processing time represented as (P_{ik}). The number of units in the input vector is dictated by the specific representation adopted for the schedule of the job shop problem. In the proposed representation, it is desired to characterize the job shop in terms of machine loads, dispersion of machine processing times and mean routing order on each machine. This is achieved by a total of 15 input units for a 5-machine job shop. The information carried in each of these units is organized as follows:

Input 1: - Total processing time on machine 1 $= \sum_{i=1}^{n} P_{i1}$ .... (4.1)

Input 2: - Total processing time on machine 2 $= \sum_{i=1}^{n} P_{i2}$ .... (4.2)

Input 3: - Total processing time on machine 3 $= \sum_{i=1}^{n} P_{i3}$ .... (4.3)

Input 4: - Total processing time on machine 4 $= \sum_{i=1}^{n} P_{i4}$ .... (4.4)

Input 5: - Total processing time on machine 5 $= \sum_{i=1}^{n} P_{i5}$ .... (4.5)

Input 6: - Variance of processing time for machine 1 $= \frac{n\sum_{i=1}^{n} P_{i1} - (\sum_{i=1}^{n} P_{i1})^2}{n(n-1)}$ .... (4.6)

Input 7: - Variance of processing time for machine 2 $= \frac{n\sum_{i=1}^{n} P_{i2} - (\sum_{i=1}^{n} P_{i2})^2}{n(n-1)}$ .... (4.7)

Input 8: - Variance of processing time for machine 3 $= \frac{n\sum_{i=1}^{n} P_{i3} - (\sum_{i=1}^{n} P_{i3})^2}{n(n-1)}$ .... (4.8)

Input 9: - Variance of processing time for machine 4 $= \dfrac{n\sum_{i=1}^{n} P_{i4} - (\sum_{i=1}^{n} P_{i4})^2}{n(n-1)}$ .... (4.9)

Input 10: - Variance of processing time for machine 5 $= \dfrac{n\sum_{i=1}^{n} P_{i5} - (\sum_{i=1}^{n} P_{i5})^2}{n(n-1)}$ .... (4.10)

Input 11: - Mean routing order of machine 1 = $R_{k=1}$ $= \dfrac{1}{n}\sum_{i=1}^{n} Q_{i1}$ .... (4.11)

Input 12: - Mean routing order of machine 2 = $R_{k=2}$ $= \dfrac{1}{n}\sum_{i=1}^{n} Q_{i2}$ .... (4.12)

Input 13: - Mean routing order of machine 3 = $R_{k=3}$ $= \dfrac{1}{n}\sum_{i=1}^{n} Q_{i3}$ .... (4.13)

Input 14: - Mean routing order of machine 4 = $R_{k=4}$ $= \dfrac{1}{n}\sum_{i=1}^{n} Q_{i4}$ .... (4.14)

Input 15: - Mean routing order of machine 5 = $R_{k=5}$ $= \dfrac{1}{n}\sum_{i=1}^{n} Q_{i5}$ .... (4.15)

4.3.2 Construction of an Input Vector

A single input vector represents a job shop problem. This section gives detailed illustration of how an input vector is computed for a randomly selected 10-job, 5-machine example. Table 4.1 depicts the raw data for this problem, identified as N10M265 - 10 denotes a 10-job problem; M represents the makespan criterion and 265 the problem number. (Note this example problem is taken from a set of 2494 random problems generated for training the network). The details of how each of the input units is constructed is explained hereafter.

Job No	Routes					Processing time (P_{ijk}) time units				
1	3	1	4	5	2	47	67	43	74	57
2	3	4	5	2	1	47	44	72	74	60
3	4	1	2	5	3	46	85	84	69	30
4	3	4	5	2	1	43	31	55	51	85
5	1	4	5	3	2	77	24	55	40	88
6	1	4	5	3	2	95	17	78	52	63
7	1	4	5	3	2	64	48	53	33	49
8	4	1	2	5	3	41	66	66	53	37
9	3	5	1	2	4	52	64	95	60	21
10	3	4	5	2	1	46	52	64	53	85

Table 4.1 Raw data of sample problem N10M265.

Input units 1 through 5 (Total processing times):

The total processing time on a given machine is the summation of processing time requirements on that machine. This measure helps to identify which machine is a bottleneck and to what degree. Total load on each of the machines is obtained by adding the processing time of all the jobs visiting that machine. The bottleneck machine is that which has the highest load. Applying equations 4.1 through 4.5 for the data of Table 4.1 results in total machine loadings given in the third column in Table 4.2. Here, machine 1 with a total load of 780 (time) units, is the bottleneck machine.

Input unit	Machine	Total load	Normalized value
1	1	780	1
2	2	645	0.827
3	3	427	0.547
4	4	367	0.471
5	5	637	0.817
Max Value (H) =		780	

Table 4.2 Determining inputs 1 to 5 for example no. NM10265.

The next step is to convert the machine loads in the range (0, 1). This is necessary because the input layer must have continuous values between 0 and 1 whenever the sigmoid activation function is used. This can be achieved by calculating a relative percentage for each load with respect to the highest load. Thus, for a given range of data with maximum values= H, a specific normalized value V is calculated by V / H. The

normalized machine loads for the example job shop of Table 4.1 are given in the last column of Table 4.2

Input units 6 through 10 (Variance of processing time):

As different jobs have different processing times on a given machine, it is useful to have an indication of the spread (dispersion) of these times about the mean. This is achieved by calculating *variance*, which is a measure of this spread for each machine. The variances for the sample problem in Table 4.1 are computed by using equations 4.6 to 4.10. The data are formatted in the range 0 to 1 by dividing them by the maximum variance among the five machines. Table 4.3 shows the variance for the five machines of the problem given in Table 4.1 and the normalized values used for input units 6 through 10.

Input unit	Machine	Variance	Normalized value
6	1	169.56	0.92
7	2	184.28	1.00
8	3	57.34	0.311
9	4	154.23	0.837
10	5	87.56	0.475
Max Value =		184.28	

Table 4.3 Inputs 6 to 10 for example problem N10M265.

Input Units 11 through 15 (Mean order of routing (R_k):

The purpose of this measure is to characterize the prevailing flow pattern, if any, due to the combination of job routes for the problem to be scheduled. The mean routing order (R_k) for each machine k is calculated as follows: -

$$R_k = \frac{1}{n}\sum_{i=1}^{n} Q_{ik} \qquad \ldots. \quad (4.16)$$

Where, Q_{ik} = machine k's order in job i's route.

k =1,2,..,5

The minimum value for Q_{ik} is 1 and the maximum is 5 (because 5 machines are visited by each job). A lower value for Q_{ik} indicates that machine k is visited predominantly by jobs during the earlier stages in their routes, while higher values suggest that the machine is visited more towards the later operations of the job's processing orders. The mean routing order for five machines based on the problem of Table 4.1 is given in column 3 of Table 4.4

Input unit	Machine	Mean routing order	Normalized Value
11	1	2.7	0.54
12	2	4.2	0.84
13	3	2.7	0.54
14	4	2.2	0.44
15	5	3.2	0.64
Max Value =		4.2	

Table 4.4 Example of normalized value for inputs 11 through 15 for example problem N10M265.

From Table 4.4 it is seen, for example, that the mean routing order for machine 2 is 4.2. This indicates that machine 2 is predominately visited towards the end of the routes in most of the jobs. The mean routing orders are normalized between 0 and 1 by dividing Q_{ik} for each machine by the number of machines (5 in this case).

Once all the input data computations are done, the input vector can be prepared. The 15 input units representing the 10-job problem in Table 4.1 is constructed by using the normalized values from the last columns of Table 4.2, 4.3 and 4.4. This result is shown in Table 4.5.

Data Set No.	Total Processing time					Variance of Processing time					Routing Complexity				
Input units	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	1.0	0.827	0.547	0.471	0.817	0.92	1.0	0.311	0.837	0.475	0.54	0.84	0.54	0.44	0.64

Table 4.5 15 Unit input vector representing job shop problem of Table 4.1.

The input vector in Table 4.5 can be interpreted to describe the job shop problem, for instance, as one where the machine 1 is a strong bottleneck. Machines 1, 2 and 4 have

a wider range of processing times as compared to the other two machines; and finally machine 2 is visited last or near last by most of the jobs.

4.3.3 Output Layer

For solving the makespan minimization problem the three dispatching rules SPT, LPT and MWKR are considered. Past research by Kumar & Srinivsan, (1996) and by Subramaniam et al, (2000) has showed rules SPT, LPT and MWKR are among a few rules effective in minimizing makespan. The neural network's task is to select one of these dispatching rules for each of the five machines. Therefore, the output layer has been designed with 15 units (3 rules x 5 machines). Each unit represents a dispatching rule for a machine. The higher the value of an output unit, the higher is the desirability of using the dispatching rule associated with that unit on the corresponding machine.

4.3.4 Data Collection

Now that the input units and the desired output have been defined, the next step is to collect data for training and testing the network. A total of 7500 job shop problems were generated randomly using n=10, 15 and 20 jobs in equal quantities. A C++ program (see Appendix A1) is used for this purpose. The processing times used for the problems are uniformly distributed between U (10,99) integer time units. The number of different job routes in each problem is between 5 and 11, selected randomly from the 120 (or 5!) possible routes in a 5-machine job shop. This range for the routes is used to ensure that several jobs have identical routes in order to simulate more realistic situations. It is

unlikely in a real world application, particularly within group technology settings, for most if not all jobs to have different routes from one another.

The optimal combination of dispatching rules that minimizes makespan for a given 5-machine job shop problem is found by enumerating all possible combinations, a total of 243 possibilities of dispatching rules (3^5 combinations) for the three dispatching rules SPT, LPT and MWKR. The neural output assumes values between 0 and 1.0; the maximum value "1.0" suggests undisputed preference for the specific dispatching rule represented by the unit. As the output value reduces, the preference for the corresponding dispatching rule diminishes proportionately. In supervised learning, desired (or target) outputs are needed in training the network. The target outputs for the proposed network are extracted from the optimal rule combinations in the following fashion, using data from Table 4.1 to illustrate the procedure.

The optimal selection of dispatching rules for the problem N10M265 gives a makespan of 826. Two different optimal combinations of the three rules exist for this problem as shown in Table 4.6

M/C 1	M/C 2	M/C 3	M/C 4	M/C 5
MWKR	SPT	MWKR	LPT	LPT
MWKR	MWKR	MWKR	LPT	LPT

Table 4.6 Optimal rule combinations for example problem N10M265.

In order to represent DR combination, the data of Table 4.6 are presented in a numerical form suitable for the output vector. The number of times a rule appears in an optimal combination is computed in each machine. For example, in machine 2 (Table 4.6) both SPT and MWKR (but not LPT) appear once in the optimal result. A similar calculation for the other machines is done and the final counts are given in Table 4.7.

DR	M/C 1	M/C 2	M/C 3	M/C 4	M/C 5
SPT	0	1	0	0	0
LPT	0	0	0	2	2
MWKR	2	1	2	0	0

Table 4.7 Numerical representation for optimal rule combination of Table 4.6.

In order to normalize the output results of Table 4.7 between 0 and 1, the entries in Table 4.7 are divided by the total number of optimal results (2) for the problem. The resulting output (o/p) vector is shown in Table 4.8, which represents the target output pattern that would be associated for training purposes with the input vector of Table 4.5.

Machine1			Machine2			Machine3			Machine4			Machine5		
SPT	LPT	MWKR	SPT	LPT	MWKR	SPT	LPT	MWKR	SPT	LPT	MWKR	SPT	LPT	MWKR
0.000	0.000	1.000	0.50	0.00	0.50	0.00	0.00	1.00	0.000	1.000	0.000	0.00	1.000	0.00

Table 4.8. 15-unit vector representing desired output for example problem N10M265.

As can be seen from the output vector above, a minimum makespan can be obtained by using MWKR on machines 1 and 3; LPT on machine 4 and 5; and either SPT or MWKR with equal favor on machine 2. Table 4.9 shows a sample of several pairs of training patterns generated from the training problems where number of jobs (n) is 10. The optimal combinations of dispatching rule in each problem are used for training. In order to compute the neural input and output (optimal combinations of dispatching rule for each problem) vectors are computed by using a program in C++ (see Appendix A2). The back propagation neural network model proposed for selecting one of the three DRs for each of the machines in a 5-machine job shop appears as shown in Figure 4.2.

4.3.5 Validation of Training data set output

In order to validate the makespan results obtained from the C++ program of Appendix A2, a simulation model of a 5-machine job shop was developed with a student

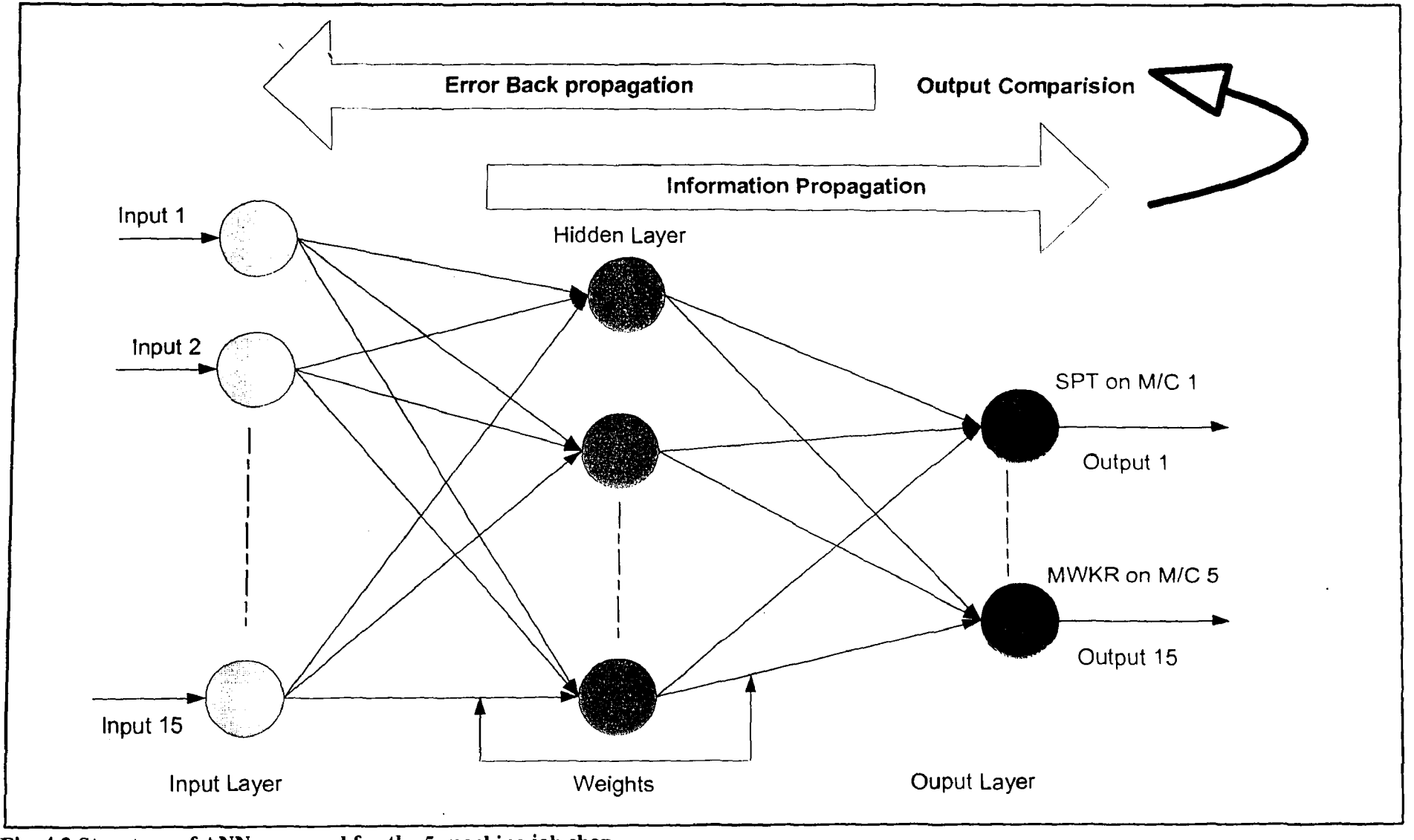


Fig. 4.2 Structure of ANN proposed for the 5-machine job shop.

Data Set	Total Processing time					Variance of Processing time					Mean Routing order				
Numbers	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10	Input 11	Input 12	Input 13	Input 14	Input 15
N10M301	0.50	0.71	0.80	0.93	1.00	0.75	0.98	0.51	1.00	0.46	0.46	0.60	0.76	0.60	0.58
N10M302	0.92	1.00	0.87	0.90	0.88	0.32	1.00	0.67	0.55	0.59	0.48	0.72	0.46	0.80	0.54
N10M303	0.75	0.35	1.00	0.67	0.72	0.64	1.00	0.58	0.54	0.42	0.68	0.36	0.50	0.58	0.88
N10M304	0.71	0.54	0.57	0.86	1.00	0.70	0.79	1.00	0.89	0.88	0.36	0.66	0.50	0.76	0.72
N10M305	1.00	0.88	0.63	0.94	0.82	0.74	0.97	1.00	0.60	0.77	0.32	0.60	0.60	0.76	0.72
N10M306	0.53	0.38	0.97	1.00	0.48	0.43	0.47	0.22	1.00	0.37	0.52	0.64	0.62	0.66	0.56
N10M307	0.48	1.00	0.84	0.84	0.53	0.76	0.77	0.39	0.44	1.00	0.48	0.88	0.52	0.32	0.80
N10M308	0.87	0.87	1.00	0.98	0.40	0.48	0.69	0.63	0.58	1.00	0.46	0.60	0.78	0.44	0.72
N10M309	0.84	0.74	1.00	0.55	0.63	0.88	0.77	0.84	0.84	1.00	0.72	0.68	0.50	0.54	0.56
N10M310	1.00	0.66	0.81	0.63	0.91	0.76	0.61	1.00	0.47	0.52	0.62	0.72	0.38	0.74	0.54
N10M311	0.52	0.78	0.95	1.00	0.63	0.20	0.68	0.70	1.00	0.71	0.68	0.60	0.64	0.44	0.64
N10M312	0.66	0.75	1.00	0.94	0.93	1.00	0.48	0.69	0.55	0.96	0.52	0.56	0.76	0.60	0.56
N10M313	0.80	0.95	0.79	0.57	1.00	0.88	0.15	0.51	1.00	0.64	0.66	0.54	0.60	0.54	0.66
N10M314	0.75	0.66	0.61	0.96	1.00	1.00	0.85	0.72	0.84	0.77	0.30	0.76	0.68	0.68	0.58

Target Output Patterns

Data Set	Machine1			Machine2			Machine3			Machine4			Machine5		
Number	o/p 1	o/p 2	o/p 3	o/p 4	o/p 5	o/p 6	o/p 7	o/p 8	o/p 9	o/p 10	o/p 11	o/p 12	o/p 13	o/p 14	o/p 15
N10M301	0.000	0.000	1.000	0.000	1.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000	0.000	0.000	1.000
N10M302	0.000	1.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000	0.500	0.500	0.000	1.000	0.000
N10M303	0.330	0.330	0.330	0.000	0.860	0.140	1.000	0.000	0.000	0.430	0.290	0.290	0.000	0.430	0.570
N10M304	1.000	0.000	0.000	0.330	0.330	0.330	0.400	0.200	0.400	0.000	0.000	1.000	0.000	0.600	0.400
N10M305	0.000	0.000	1.000	0.350	0.300	0.350	0.300	0.250	0.450	0.000	0.000	1.000	0.150	0.450	0.400
N10M306	0.500	0.080	0.420	0.330	0.330	0.330	0.420	0.000	0.580	0.000	0.420	0.580	0.330	0.330	0.330
N10M307	0.310	0.380	0.310	0.080	0.460	0.460	0.460	0.000	0.540	0.000	0.000	1.000	0.330	0.330	0.330
N10M308	1.000	0.000	0.000	0.500	0.000	0.500	0.000	0.500	0.500	1.000	0.000	0.000	0.330	0.330	0.330
N10M309	0.370	0.370	0.260	0.260	0.320	0.420	0.000	0.000	1.000	0.550	0.180	0.260	0.500	0.160	0.340
N10M310	0.330	0.000	0.670	0.330	0.330	0.330	0.000	0.330	0.670	0.330	0.330	0.330	0.000	0.000	1.000
N10M311	0.330	0.330	0.330	0.140	0.430	0.430	0.430	0.290	0.290	0.000	0.000	1.000	0.330	0.330	0.330
N10M312	0.480	0.260	0.260	0.320	0.550	0.130	0.420	0.100	0.480	0.520	0.130	0.350	0.190	0.000	0.810
N10M313	0.500	0.250	0.250	1.000	0.000	0.000	0.170	0.170	0.670	0.420	0.170	0.420	0.000	0.000	1.000
N10M314	1.000	0.000	0.000	0.330	0.330	0.330	0.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000	1.000

Table 4.9 Sample of input and output pattern pairs.

version of the simulation software package Arena (Kelton et al, 2002). This model is given in Appendix B.

Randomly selected problems from the training (data) set were tested using this Arena job shop model for all the combinations of the three dispatching rules on the 5 machines. It was seen that the simulation output from the Arena simulations matched the results from the C++ program.

Table 4.10 presents the different dispatching rules tested by the arena model for the performance objective of minimizing the makespan.

Arena 5-machine Job shop simulation Model		
Dispatching rules	Queue Ranking Rule	Expression/Attribute
SPT	Least Value First	Processing time (P_{ijk})
LPT	High Value First	Processing time (P_{ijk})
MWKR	High Value First	Remaining time $\sum_{i=1}^{n}\left(\sum_{j.k=1}^{j.k=5}(P_{ijk}) - (P_{i(j-1)k})\right)$

Table 4.10 Attributes used in job shop Arena model.

4.3.6 Data sorting

As seen in Table 4.9, several of the output patterns show that more than one dispatching rule is favored on one or more of the machines. This arises when there exists more than one combination of dispatching rules that produce the lowest makespan. For example, in data set N10M306, in Table 4.9, the three output units for machine 2 show equal values of 0.330. This indicates that all three dispatching rules appear with equal frequencies in the multiple combinations that yield minimum makespan for this problem. Using data from a problem such as this to train a network is not helpful because it introduces conflicting information and prevents the network from detecting the useful relationships between input data and optimal outputs. Therefore, if the network is trained without sorting and screening the input data, then chances are that the

network will not learn properly and may not select the best dispatching rules. The sorting is done conveniently, using Structure Query Language (SQL).

SQL has the capabilities to sort the data considering multiple criteria at a time. In this case there are 3 outputs for each machine. There are two different sorting criteria considered:

A sorting index of 5 identifies the problem whose output units for all the five machines is either (1,0,0), (0,1,0,) or (0,0,1). For example referring to the problem number N10M301 of Table 4.9 the output value of each machine satisfies the sorting criteria for a sorting index of 5.

A sorting index of 4 identifies the problem whose output units for any four out of the five machines is either (1,0,0), (0,1,0,) or (0,0,1). For the remaining machine the value of the output unit is treated as zero while sorting; resulting in the sum of sorting index as 4. For example, for the problem number N10M302 of Table 4.9 the sorting index is 4 because four out of the five machines satisfy the sorting criteria. Likewise the patterns can be sorted for sorting indices of 3, 2 , 1 and 0.

The data is sorted in the descending order having the maximum sorting index (5 in this case). The sorting procedure is illustrated in Table 4.11, after the same patterns of Table 4.9 are sorted. The first column in the target output patterns of Table 4.11 lists the sorting index for each problem. The number of outputs (equivalent to the sorting index) satisfied by the respective problems is highlighted in Table 4.11.

For all of the 7500 training problems (as explained in section 4.3.4) data are sorted simultaneously. For training the network it is not advisable to use only data whose sorting index is 5. This is because many problems exhibit multiple optimal combinations, and to help the network deal with such cases it is a good idea to incorporate some data having characteristics where more

Data Set	Total Processing time					Variance of Processing time					Mean Routing order				
Numbers	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7	Input 8	Input 9	Input 10	Input 11	Input 12	Input 13	Input 14	Input 15
N10M301	0.50	0.71	0.80	0.93	1.00	0.75	0.98	0.51	1.00	0.46	0.46	0.60	0.76	0.60	0.58
N10M302	0.92	1.00	0.87	0.90	0.88	0.32	1.00	0.67	0.55	0.59	0.48	0.72	0.46	0.80	0.54
N10M314	0.75	0.35	1.00	0.67	0.72	0.64	1.00	0.58	0.54	0.42	0.68	0.36	0.50	0.58	0.88
N10M304	0.71	0.54	0.57	0.86	1.00	0.70	0.79	1.00	0.89	0.88	0.36	0.66	0.50	0.76	0.72
N10M305	1.00	0.88	0.63	0.94	0.82	0.74	0.97	1.00	0.60	0.77	0.32	0.60	0.60	0.76	0.72
N10M308	0.53	0.38	0.97	1.00	0.48	0.43	0.47	0.22	1.00	0.37	0.52	0.64	0.62	0.66	0.56
N10M313	0.48	1.00	0.84	0.84	0.53	0.76	0.77	0.39	0.44	1.00	0.48	0.88	0.52	0.32	0.80
N10M303	0.87	0.87	1.00	0.98	0.40	0.48	0.69	0.63	0.58	1.00	0.46	0.60	0.78	0.44	0.72
N10M307	0.84	0.74	1.00	0.55	0.63	0.88	0.77	0.84	0.84	1.00	0.72	0.68	0.50	0.54	0.56
N10M309	1.00	0.66	0.81	0.63	0.91	0.76	0.61	1.00	0.47	0.52	0.62	0.72	0.38	0.74	0.54
N10M310	0.52	0.78	0.95	1.00	0.63	0.20	0.68	0.70	1.00	0.71	0.68	0.60	0.64	0.44	0.64
N10M311	0.66	0.75	1.00	0.94	0.93	1.00	0.48	0.69	0.55	0.96	0.52	0.56	0.76	0.60	0.56
N10M306	0.80	0.95	0.79	0.57	1.00	0.88	0.15	0.51	1.00	0.64	0.66	0.54	0.60	0.54	0.66
N10M312	0.75	0.66	0.61	0.96	1.00	1.00	0.85	0.72	0.84	0.77	0.30	0.76	0.68	0.68	0.58

Target Output Patterns

Sorting	Data Set	Machine1			Machine2			Machine3			Machine4			Machine5		
Index	Number	o/p 1	o/p 2	o/p 3	o/p 4	o/p 5	o/p 6	o/p 7	o/p 8	o/p 9	o/p 10	o/p 11	o/p 12	o/p 13	o/p 14	o/p 15
5	N10M301	0.000	0.000	1.000	0.000	1.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000	0.000	0.000	1.000
4	N10M302	0.000	1.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000	0.500	0.500	0.000	1.000	0.000
4	N10M314	1.000	0.000	0.000	0.330	0.330	0.330	0.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000	1.000
2	N10M304	1.000	0.000	0.000	0.330	0.330	0.330	0.400	0.200	0.400	0.000	0.000	1.000	0.000	0.600	0.400
2	N10M305	0.000	0.000	1.000	0.350	0.300	0.350	0.300	0.250	0.450	0.000	0.000	1.000	0.150	0.450	0.400
2	N10M308	1.000	0.000	0.000	0.500	0.000	0.500	0.000	0.500	0.500	1.000	0.000	0.000	0.330	0.330	0.330
2	N10M313	0.500	0.250	0.250	1.000	0.000	0.000	0.170	0.170	0.670	0.420	0.170	0.420	0.000	0.000	1.000
1	N10M303	0.330	0.330	0.330	0.000	0.860	0.140	1.000	0.000	0.000	0.430	0.290	0.290	0.000	0.430	0.570
1	N10M307	0.310	0.380	0.310	0.080	0.460	0.460	0.460	0.000	0.540	0.000	0.000	1.000	0.330	0.330	0.330
1	N10M309	0.370	0.370	0.260	0.260	0.320	0.420	0.000	0.000	1.000	0.550	0.180	0.260	0.500	0.160	0.340
1	N10M310	0.330	0.000	0.670	0.330	0.330	0.330	0.000	0.330	0.670	0.330	0.330	0.330	0.000	0.000	1.000
1	N10M311	0.330	0.330	0.330	0.140	0.430	0.430	0.430	0.290	0.290	0.000	0.000	1.000	0.330	0.330	0.330
0	N10M306	0.500	0.080	0.420	0.330	0.330	0.330	0.420	0.000	0.580	0.000	0.420	0.580	0.330	0.330	0.330
0	N10M312	0.480	0.260	0.260	0.320	0.550	0.130	0.420	0.100	0.480	0.520	0.130	0.350	0.190	0.000	0.810

Table 4.11 Input and output patterns of Table 4.9 after sorting.

than one dispatching rules is favored by one of the machines. This will help the network to learn the stronger relationships between input and output patterns. On the other hand, using patterns with lower sorting indices could hamper the learning. Therefore, it was decided to incorporate only those patterns with sorting indices 4 and above for the training. Thus, ignoring all the data having a sorting index of less than 4, the result is a 2494 training data set extracted from the 7500 original problems for training the neural network.

4.4 Design of Neural Network

Once the pre-processing (sorting) of the data is done, the next step is to train the neural network. However, the final number of units for the hidden layer still needs to be decided.

4.4.1 Hidden Layer

The hidden layer is mainly required to overcome the non-linear learning problem. It is also where the network learns interdependencies in the model. Figure 4.3 provides some detail into what goes on inside a hidden node. Neurons of the hidden layer receive inputs from neurons of the input layer, depending on the sum of the input weights that produce an output. The output is then further transferred to units of the output layer.

Garson, (1998) states that "if the problem has a linear solution then it may not be appropriate to use hidden layers. In theory, a neural network with at least one layer and adequate number of hidden units is capable of solving most problems. However, in practice one or more layers may be used depending on the complexity of a problem (such

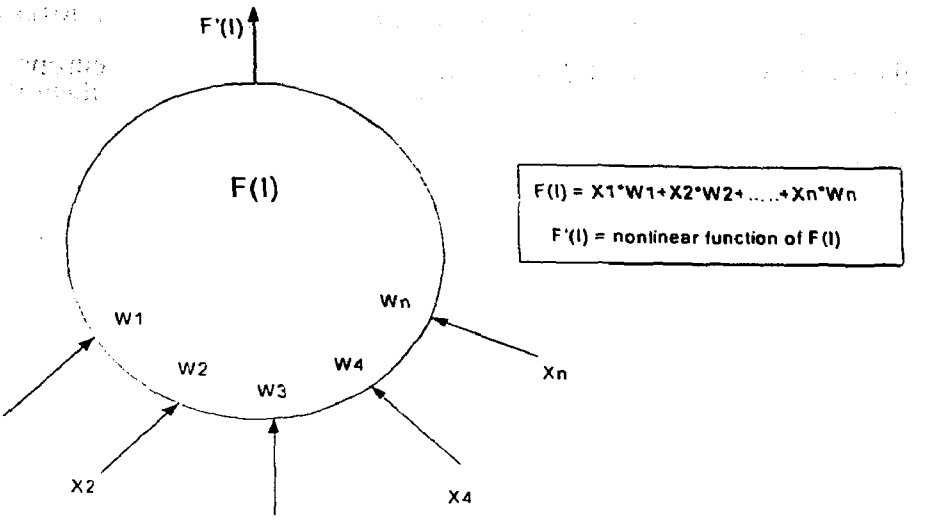


Fig. 4.3 Details of a hidden node

as complicated function fitting problems) and cost and resources. As the number of hidden layers increase, the meaningfulness of the back propagated error term decreases. Moreover, the training time increases by an order of the magnitude for each additional hidden layer." Thus, considering all the above-mentioned factors, only one hidden layer is considered for building the neural network.

4.4.2 Hidden Units (neurons)

In most situations, there is no magic formula that finds the best number of hidden units without training several networks and estimating the generalization error of each. If only a few hidden units are considered, then the network may not train well because the model lacks sufficient complexity to reflect input-output patterns in the training set. On the other hand, if too many hidden units are used, then the network may overtrain and generalization will suffer as the network simply memorizes the input-output patterns in the training set.

Some researchers have developed a heuristic approach to determine the best number of hidden neurons, (Smith, 1999). Two examples of these are the following:

- If the input patterns are of dimension N and the output of K neurons, then the number of hidden layers would be $J = \sqrt{NxK}$ neurons.

- Another approximation is $J = \frac{1}{2}(N + K) + \sqrt{P}$, where P is the number of patterns in the training set. This is the default formula used in the commercial neural network package Neuroshell 2.

- On the other hand (Baily and Thompson, (1990)) suggest that the number of hidden neurons in a three layer neural network, which include input, output and hidden layers should be 75% of the number of neurons in the input layer.

Most of the above heuristics are based on the assumptions that the training set is at least twice as large as the number of weights and preferably at least four or more times larger than the number of weights. If the same is not the case, then the number of hidden neurons will be affected by the number of training sets. Moreover, there are some other factors like the amount of noise in the test patterns, the complexity of the function or classification to be learned, the architecture, etc. that will also have impact on the number of hidden neurons. A trial and error approach is often taken starting with a modest number of hidden neurons and gradually increasing this number until the network fails to reduce its error Kaastra and Boyd, (1996) and Garson, (1999). This last approach is the one adopted in this study for determining the number of units.

4.4.3 Training Stopping Criteria

A neural network is trained in epochs, where each epoch represents a complete pass of the training set through the network. When each training pattern is presented to the

network, the error between the actual outputs in the training pattern and the network's predictions for each of the network's outputs is computed. The total error for each pattern is the sum of the squares of the differences. At the end of each epoch the average error over all training patterns is computed. As the epochs progress and the training continues, the network learning improves, i.e., the error on the training and test sets decreases. For the test set, however, there comes a point where the error in the test examples starts to increase with additional training. While training a neural network, one expects to obtain a network with optimal generalization performance. Thus, the question of when to end the training is a critical one. There are different choices for stopping criteria for both training and testing sets adopted from Neuroshell 2, which are based on:

- *The average error in the training set.* End training when the lowest value for the average error in the training set is reached.
- *The largest average error in the training set.* This is the network's latest computation for the difference between the network's predictions and the actual predictions for data in the training set.
- *The events since the minimum average error in the testing set.* An event is the presentation of a single training pattern to the neural network.

Additionally, Neuroshell 2 offers two options about automatic saving of the training based on:

- *Best training set* saves the network every time it reaches a new minimum average error for the training set.
- *Best testing set* saves the network every time it reaches a new minimum average error for the test set.

4.4.4 Training Procedure

The following steps are used in order to achieve a generalization capability in the performance of a trained neural network.

- *Step 1*: Train the network with the training patterns and monitor the minimum average error. Once a new minimum average error is observed, then interrupt the training. Save the neural weights and go to step 2.
- *Step 2*: Apply the partially trained network to the test set and compute the performance measure (using the C++ program of Appendix A3). If the performance measure (either total makespan or total mean flowtime) is improved upon from the previous trial, then return to step 1. Otherwise proceed to step 3.
- *Step 3*: Save the current weights as a final trained network.

4.4.5 To Find optimal number of Hidden neurons (units)

In order to find the best number of hidden neurons, a trial and error approach is carried out involving 6 to 18 hidden neurons using the above-mentioned training stopping criteria. The smallest number represents too few and the highest number too many hidden neurons. The training trials were done using the commercial software "Neuroshell 2", and with the parameters listed in Appendix C1.

A new set of 1200 randomly generated test problems (having random seeds different than the training data set) were used. The original 2494 training patterns (as explained in the section 4.3.6) are considered for training. The objective is to find the size of the hidden layer that minimizes the obtained makespan for the test set (refer step 2 of the training procedure).

Once the training is completed the test job shop problems are run (using the C++ program of Appendix A3) to see how accurately the neural network has assigned the priority rules. The neural network's final performance for each trial with a different number of hidden neurons is tabulated in Table 4.12, which expresses the BPNN results in terms of the total makespans of the test set problems. This is further illustrated graphically in Figure 4.4.

Experiment Number	Number of Hidden Neurons	Total Makespan	Minimum Average Error
1	6	1398179	1.447
2	8	1394552	1.456
3	9	1393378	1.437
4	10	1395058	1.439
5	15	1397199	1.457
6	18	1397533	1.482

Table 4.12 Total makespan using BPNN with various number of hidden neurons.

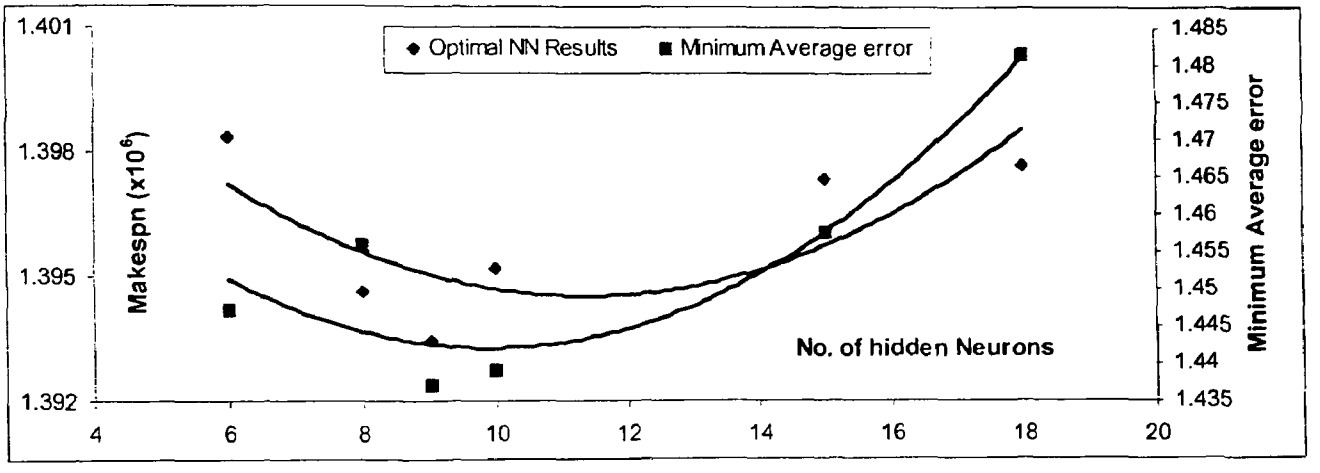


Fig.4.4 Effect of number of hidden neuron on the minimum average error for makespan.

When the number of hidden neurons is small, the corresponding minimum average error (see section 4.4.3) is also low. However, the BPNN results are not the best. Figure 4.3 shows that the BPNN performs best, and the minimum average error is lowest when the number of hidden neurons is 9. With a higher number of hidden neurons, the network appears to memorize rather than learn the patterns. So, the chosen network has a 15-9-15 structure. The next section explains in greater detail the training of the 15-9-15 network.

It is noteworthy that the same training process, training data and parameters described in the next section were also used in the trials to determine the number of hidden neurons, presented in Table 4.12.

4.5 Training the BPNN

A total of 2494 training patterns extracted from problem instances combining n=10, 15, 20 for the five machines is considered for training the network. Similarly, a test data set of 1200 problems with the same combinations of n as the training set is generated separately. As explained in the preceding chapter, the default values of Neuroshell 2 for training are a learning rate = 0.1, momentum = 0.1 and initial weights = 0.3. The training was performed on AMD 900MHz Presario personal computer. The training is carried out and the total of the makespans for the test set is recorded for every epoch that produces further reduction in the minimum average error (see section 4.4.3). The training results are tabulated as shown in Table 4.13. The training procedure is extended from generalization test number 9 to 16 (refer to the first column of Table 4.13) in order to observe the behavior of the neural network on the test set, if the training is continued beyond a certain point. In other words, to observe what happens when the neural network is over trained.

Figure 4.5 presents the behavior of the minimum average error on the training set and the corresponding neural network results on the test set. It can be seen that as the minimum average error is reduced, the BPNN results improve correspondingly up to a certain point (generalization test number 8 in this case), beyond which the results start

deteriorating. It is at this point that the network generalizes best, and no further improvement is possible with continued training.

Generalization Test	Minimum Average Error	Epoch	Learning Event	Time (Sec)	BPNN Suggested DR
1	1.5517	2	4988	3	1449074
2	1.5295	5	12470	8	1439644
3	1.4938	12	29928	17	1414140
4	1.4906	14	34916	19	1410225
5	1.4760	18	44892	25	1409133
6	1.4711	21	52374	30	1400621
7	1.4558	23	57362	33	1403802
8	1.4372	38	92278	52	1393378
9	1.4358	46	114724	65	1398482
10	1.4336	50	124700	70	1399688
11	1.4238	75	187050	105	1398570
12	1.4196	80	199520	113	1400944
13	1.4051	148	369112	213	1401905
14	1.3890	214	533716	315	1405274
15	1.3857	407	1015058	583	1400672
16	1.3776	642	1601148	1002	1402333

Table 4.13 Training and generalization results for minimizing makespan.

4.6 Implementation

Once the training of the proposed network is completed, the next step is to see how well the neural network performs in selecting dispatching rules. In order to explain how the neural network assigns a dispatching rule to each machine, an example of 20 jobs on 5 machines is given in Table 4.14. The above example data are converted to a neural network input representation by following the procedure explained in section 4.3.2. The resulting 15-unit input vector is given in Table 4.15.

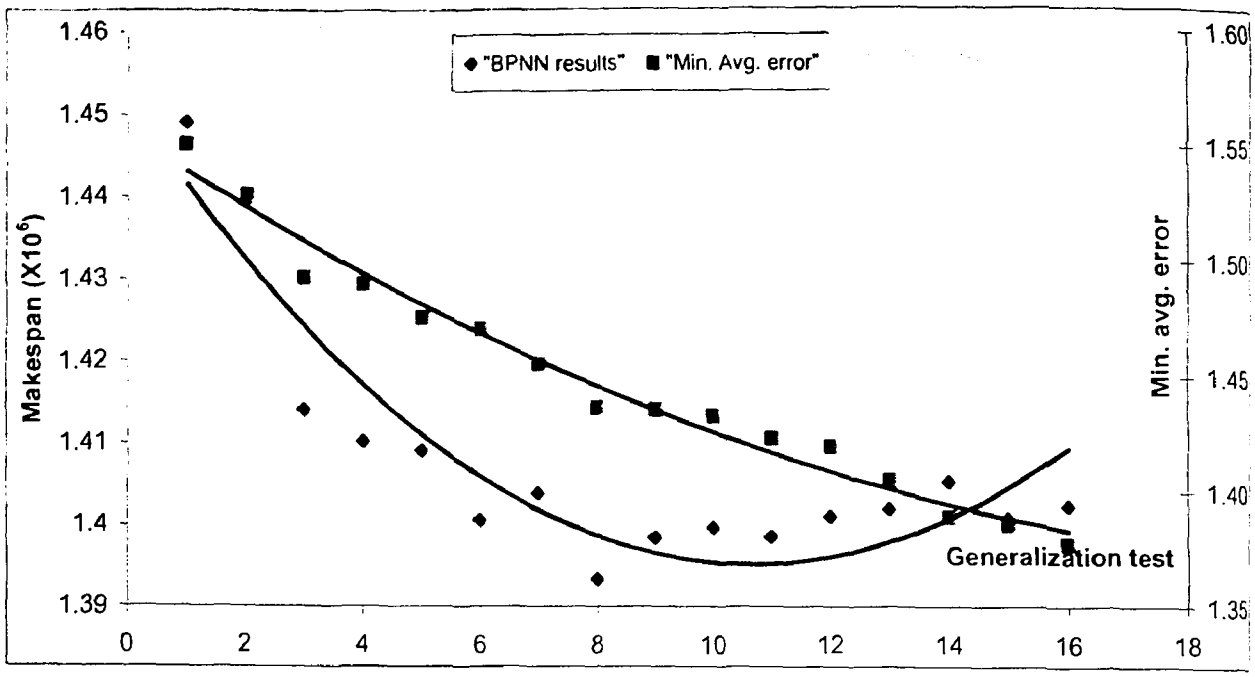


Fig. 4.5 BPNN generalization V/S minimum average error for makespan.

Job no	Routes					Processing time				
1	2	1	4	3	5	52	60	94	33	23
2	2	1	5	3	4	58	41	20	34	57
3	5	4	3	1	2	51	77	33	74	37
4	5	2	4	3	1	46	34	92	62	53
5	5	4	3	1	2	46	92	58	63	35
6	5	2	4	3	1	37	66	61	29	55
7	5	2	4	3	1	53	68	90	34	63
8	5	4	3	1	2	28	84	45	70	46
9	1	4	3	5	2	68	87	42	49	32
10	5	2	4	3	1	22	53	92	46	50
11	2	1	4	3	5	61	67	64	24	53
12	5	2	4	3	1	26	57	63	32	76
13	3	1	4	2	5	41	68	65	67	38
14	2	1	4	3	5	36	37	95	47	45
15	5	2	4	3	1	40	66	67	30	71
16	2	1	5	3	4	61	40	15	58	69
17	3	1	4	2	5	55	61	91	59	14
18	1	4	3	5	2	65	84	52	38	67
19	1	4	3	5	2	42	82	26	16	32
20	2	1	4	3	5	51	45	80	45	51

Table 4.14 A 20-job example problem.

Set No.	Total Processing time					Variance of Processing time					Mean routing order				
Input	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
N20E1	0.74	0.65	0.52	1.0	0.45	0.792	0.91	0.69	0.86	1	0.61	0.56	0.68	0.58	0.57

Table 4.15 Input vector for example problem of Table 4.14.

A trained network generates output in response to the data vector presented to it at the input layer. When the input vector of Table 4.15 is introduced to the BPNN, which has been just been trained for the minimization of makespan, a feed forward set of computations produces the output vector given in Table 4.16.

	Machine1			Machine2			Machine3			Machine4			Machine5		
Output	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	SPT	LPT	MWKR	SPT	LPT	MWKR	SPT	LPT	MWKR	SPT	LPT	MWKR	SPT	LPT	MWKR
BPNN	0.332	0.206	**0.449**	**0.462**	0.117	0.411	0.296	**0.394**	0.266	0.259	0.062	**0.694**	**0.507**	0.228	0.229

Table 4.16 BPNN output for an example problem of Table 4.14.

The maximum value in each set of 3 units associated with each machine is identified. The dispatching rule corresponding to those units (**highlighted** in the Table 4.16) is then assigned to the machines. In the above example, the neural network's choice for the first machine is MWKR; the second machine is SPT; the third machine is LPT; the fourth machine is MWKR; and the last machine is SPT. Applying this allocation of dispatching rules, the makespan for the problem is found to be 1652. This compares to a minimum makespan of 1628 for the optimal rule combination of SPT-SPT-LPT-MWKR-SPT applied on machines 1 to 5 respectively. The neural network's suggested combination of dispatching rules deviates from the optimal combination only for the first machine, where instead of SPT, the BPNN applies MWKR. Moreover, if all the machines use only a single rule for all machines (either SPT, LPT or MWKR), then the resulting makespans are 1720, 1699, and 1651 respectively.

4.7 BPNN model for minimizing mean flowtime criteria

For the performance criterion of minimizing the mean flowtime, a BPNN model is constructed by the same methodology as used for minimizing makespan. The only difference however, lies in the 3 dispatching rules, where SPT, PT+WINQ and LWKR are considered for the flowtime objective. This is because past research shows that all these rules are effective in minimizing mean flowtime (Waikar et al, 1995 and Rajender and Holthaus, 1999).

In order to find an optimal number of hidden neurons, some modification was done in the step used for the makespan BPNN. A total of 2636 training patterns were extracted from problems of n=10, 15 and 20. The same 1200 test data set used for makespan was also used in the flowtime case. The NN parameters used for finding optimal number of hidden neurons is given in Appendix C2.

Table 4.17 represents neural network results for seven different numbers of hidden neurons and Figure 4.6 depicts the effect of the number of different hidden neurons on the neural network and the minimum average error. Hence, the network used for minimizing the mean flowtime has a 15-20-15 structure.

Experiment Number	Number of Hidden Neurons	Total Mean Flowtime	Minimum Average Error
1	10	873181.62	1.700
2	15	872529.56	1.671
3	17	872254.31	1.668
4	18	871955	1.641
5	19	871687.81	1.659
6	20	871669.31	1.594
7	25	871967.44	1.655

Table 4.17 Total mean flowtimes for BPNN with various numbers of hidden neurons.

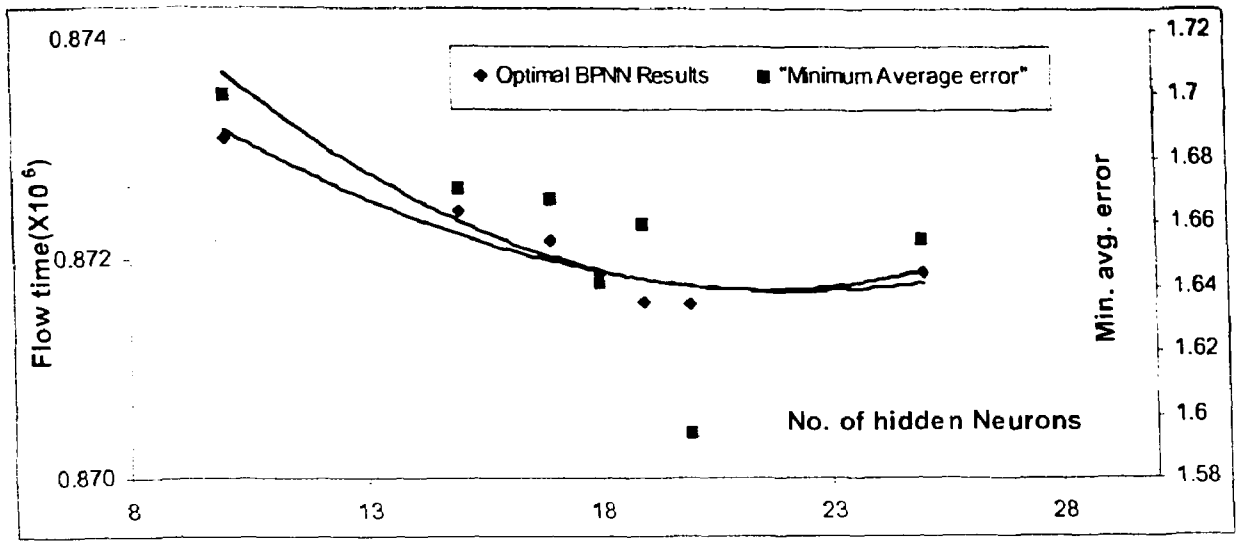


Fig. 4.6 Effect of number of hidden neurons on the minimum average error for mean flowtime.

In order illustrate how the trained network selects dispatching rules to minimize flowtime, the same example problem of Table 4.14 is considered. The input vector is the same; however, the output vector represents different rules. Output resulting from the feed forward processing induced by the application of input vector of table 3.16 to the BPNN (15-20-15) trained for flowtime minimization is given in Table 4.18.

Output	Machine1			Machine2			Machine3			Machine4			Machine5		
DR	SPT	WINQ +PT	LWKR	SPT	WINQ +PT	LWKR	SPT	WINQ +PT	LWKR	SPT	WINQ +PT	LWKR	SPT	WINQ +PT	LWKR
Output	0.167	0.070	**0.737**	0.014	0.150	**0.776**	0.216	0.239	**0.556**	**0.956**	0.0	0.041	0.07	0.228	**0.708**

Table 4.18 Mean flowtime BPNN optimal example problem of Table 4.14.

The above neural network's recommendations (**highlighted** in the Table 4.18) are LWKR for the first, second and third machines, SPT for the fourth machine and LWKR for the last machine. This combination results in a mean flowtime of 936.65, which compares to a possible minimum of 921.45. The neural network's suggested combination of dispatching rule deviates from the optimal combination only for the last machine, where instead of WINQ+PT, the BPNN suggests LWKR. On the other hand, if all the machines applied only one of SPT, WINQ+PT or LWKR dispatching rules on all the machines, then the mean flowtimes are 989.95, 1031.55, and 952.85 respectively.

In this chapter, a detailed description of the design of BPNN for two different performance criteria is described. For the makespan, a BPNN model of 15-9-15 was designed and for the mean flowtime a BPNN model of 15-20-15. For both the performance criteria, the trained network suggested a combination of dispatching rules that were reasonably close to the optimal combinations. The next chapter will test the generalization capabilities of both trained networks for problems with the number of jobs ranging between a minimum of 10 to a maximum of 100 jobs.

CHAPTER 5

Analysis and Discussion

In this chapter, the generalization capability of the BPNN and its effectiveness in providing results to new problems (unseen during the neural network's training) will be examined for the performance objectives of minimizing makespan and mean flowtime. This will be followed by a discussion on the results and an analysis of variance.

5.1 Analysis of Trained Neural Network

Once the training of the neural network as explained in the previous chapter is completed, it is necessary to test how well the neural network's suggested combination of dispatching rules perform in new problems.

5.1.1 Neural Network Weights

The trained neural network holds its knowledge in the weights between the nodes. The final weights from the training process using Neuroshell 2 (as explained in the section 4.4.4) are provided in Appendix D1 and D2 for the makespan and mean flowtime networks respectively.

5.2 Test Problems

A total of fourteen different sets of new problems with the job numbers (n) ranging from 10 to 100 jobs is created using the same parameters used for the testing sets in the

previous chapter. Each set contains fifty test problems generated randomly using different random seeds for each different category of n. Altogether, 700 problems are generated using the C program of Appendix A4.

5.3 Test Results for Minimizing Makespan

Once the generation of the various problems is completed as mentioned above, the next step is to obtain and compare the total makespan in each set. Each set of fifty problems is tested 4 times. In the first trial, the SPT rule is applied on all five machines and the total makespan is recorded. In the second and third trials, the LPT and MWKR rules are tested respectively, in a similar fashion. Finally, the BPNN is used in the fourth trial to select dispatching rule combinations for the test problems. The BPNN generates the preferred dispatching rule for each machine by feed forward processing identical to that described for the example problem in section 4.7.

Table 5.1 presents the total of the makespans in each of the trials for all 14 sets of test problems. In order to evaluate how well the neural network's (BPNN) suggested combination of dispatching rules performs, a comparison with the makespans produced by optimal combinations of the dispatching rules is also presented in Table 5.1.

As can be seen from the results of Table 5.1, the trained neural network generates better total makespan results as compared to using the same dispatching rule on all the machines. Furthermore, the percentage deviation of the neural network (BPNN) results from the optimal is close.

Deviation is calculated by using equation 5.1. The deviation of BPNN from optimal results ranged from a minimum of 0.3% to a maximum of 3.0%.

Test Set no.	n	Dispatching rule (time units)				Optimal Results	Deviation (%) (Eq. No. 5.1)
		SPT	LPT	MWKR	BPNN		
1	10	41317	42962	38565	38460	37337	3.008
2	15	60826	62951	58414	57405	55987	2.533
3	20	76971	80075	74785	73939	72498	1.988
4	25	94894	98384	92800	91082	90547	0.591
5	30	112414	116351	110828	109676	107831	1.711
6	35	131193	133983	128464	127539	126307	0.975
7	40	143277	146885	141227	139827	138829	0.719
8	45	165143	169518	163726	162822	160345	1.545
9	50	180936	186490	179384	177479	176721	0.429
10	55	204696	209076	203679	201029	199326	0.854
11	60	220154	223899	217278	215592	214898	0.323
12	75	274736	277632	275343	271626	268818	1.045
13	85	307177	313202	307296	304342	303392	0.313
14	100	356962	361673	355889	353841	352714	0.320

Table 5.1 Summary of total makespan in sets of 50 test problem for various n.

$$\text{Deviation (\%)} = \frac{(BPNN - OPTIMAL)}{OPTIMAL} * 100 \qquad \ldots. \quad (5.1)$$

As a sample, results for all the 50 problems of test set no. 9 (n=50) are presented in Table 5.2. These results show that there is not much difference between the neural network's performance and the optimal combination of dispatching rules, as further illustrated in Figure 5.1. The neural network achieved optimal results in 80% of the test problems and deviated by an average of 2.1 % from the optimal in the rest. For comparison SPT, LPT and MWKR matched the optimal result in 14%, 35% and 4% respectively of the test problems. Similar comparisons between optimal and BPNN results for individual problems in the other thirteen sets are attached in the Appendix E1.

5.4 Test Results for Minimizing Mean Flowtime

The same sets of test problems used for testing the makespan are also used for testing the generalization capability and the effectiveness of the neural network that has been trained to minimize mean flowtime. This neural network is tested in the same manner as the previous network used for testing makespan. In the first three trials each

Data set	Dispatching rule (time units)					Data set	Dispatching rule (time units)				
No.	SPT	LPT	MWKR	BPNN	Optimal	No.	SPT	LPT	MWKR	BPNN	Optimal
50N1	2087	2181	2087	2087	2087	50N26	3017	3224	3190	2993	2993
50N2	3358	3379	3355	3239	3234	50N27	3971	3999	3881	3881	3881
50N3	2517	2669	2449	2449	2449	50N28	3406	3322	3189	3189	3189
50N4	3520	3571	3508	3508	3508	50N29	3654	3614	3540	3540	3540
50N5	3719	3774	3677	3677	3677	50N30	4119	4154	3986	3986	3986
50N6	3805	3754	3754	3754	3754	50N31	3548	3648	3503	3503	3503
50N7	3427	3569	3406	3406	3406	50N32	3911	4014	3874	3874	3874
50N8	3748	3769	3620	3620	3620	50N33	3632	3629	3482	3482	3482
50N9	4349	4743	4134	4134	4134	50N34	3821	4106	4035	3920	3689
50N10	3954	3957	3856	3856	3856	50N35	3887	3994	3885	3885	3856
50N11	3759	3887	3739	3584	3568	50N36	3324	3351	3221	3183	3183
50N12	3144	3360	3143	3143	3055	50N37	3946	3982	3774	3774	3774
50N13	3138	3321	3104	3104	3104	50N38	3317	3551	3325	3287	3287
50N14	3993	4043	4001	4001	3993	50N39	4049	4159	3946	3946	3946
50N15	3814	4025	4115	3865	3645	50N40	3849	3994	3845	3845	3845
50N16	3952	4152	4002	3952	3952	50N41	3433	3562	3515	3462	3433
50N17	3546	3919	3635	3546	3546	50N42	3688	3799	3683	3683	3683
50N18	4029	4214	3835	3835	3835	50N43	3740	3828	3590	3590	3590
50N19	3612	3826	3572	3572	3572	50N44	3720	3814	3960	3699	3601
50N20	3248	3396	3163	3163	3163	50N45	3940	3917	3798	3798	3798
50N21	3713	3705	3566	3566	3566	50N46	3516	3408	3408	3408	3408
50N22	3534	3626	3409	3409	3409	50N47	3331	3379	3263	3263	3263
50N23	3451	3558	3451	3451	3451	50N48	3862	3876	3719	3719	3719
50N24	3751	4017	3743	3743	3743	50N49	3313	3419	3156	3156	3156
50N25	4037	4080	3978	3978	3978	50N50	3737	4252	4314	3771	3737
						Total	180936	186490	179384	177479	176721

Table 5.2 Results for individual problems for n=50 (set no. 9 in Table 5.1).

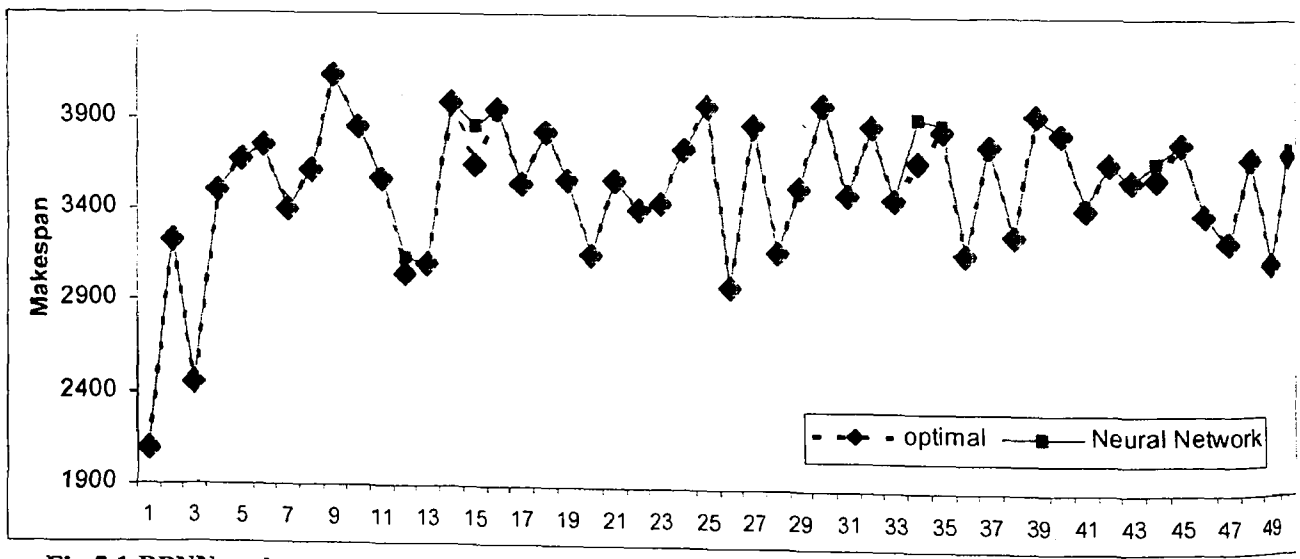


Fig 5.1 BPNN makespan compared to makespans from optimal rule combinations for n=50.

set is tested using SPT, WINQ+PT and LWKR respectively on all machines. BPNN is tested in the fourth trial. The results (expressed as total of mean flowtime) are given in Table 5.3.

Test		Dispatching rule (time units)					Deviation (%)
Set no.	n	SPT	WINQ+PT	LWKR	BPNN	Optimal	(Eq. No. 5.1)
1	10	27446.3	27596	26831.5	26889.2	26075.1	3.122
2	15	37677.47	37709.07	36280.73	36191.93	34558.2	4.727
3	20	45734.46	45676.21	43935.94	43842.51	43053.48	1.833
4	25	55211.6	55025.8	52772.55	52316.51	51516.56	1.553
5	30	64106.99	63345.39	60897.44	60255.63	59386.82	1.463
6	35	74322.56	72803.34	70214.73	69623.42	68469.23	1.686
7	40	80264.66	78441.74	75373.9	74975.38	73573.33	1.906
8	45	93754.33	90887.78	86977.52	86073.49	84897.65	1.385
9	50	101784.1	98259.69	94922.15	94021.89	92624.78	1.508
10	55	113800.9	109890.98	106158.4	104845.4	103785.08	1.022
11	60	120913.9	117916.74	113117.8	111846.1	110364.75	1.342
12	75	149977.6	146410.08	139655.6	137759.3	136286.26	1.081
13	85	170545.7	162467.98	156726.2	154907.9	152938.76	1.288
14	100	193658.7	188070.2	179319.3	176437.4	174695.08	0.997

Table 5.3 Summary of total mean flowtime results of test sets consisting of 50 problems each.

Once again, the neural network results are better as compared to using either SPT, WINQ+PT or LWKR common on all five machines. The network's results deviate from the optimal by an average of 1.74% (having a minimum of 1.0 to a maximum of 4.7%). SPT, PT+WINQ and LWKR results deviate on average by 9.9%, 6.25% and 2.52 % respectively from optimal results.

Similarly, results for all the 50 problems in test set no. 9 (n=50) are presented in Table 5.4. Also, the comparison between BPNN and optimal mean flowtime is shown graphically in Figure 5.2. It can be observed that BPNN achieves optimal results in 20% of the problems, but deviates by an average of 2.3% from optimum in the remainder. Similar comparisons between optimal flowtimes and BPNN results for individual problems in the other thirteen sets are attached in the Appendix E2.

Data set No.	Dispatching rule (time units)				Optimal	Data set No.	Dispatching rule (time units)				Optimal
	SPT	WINQ+PT	LWKR	BPNN			SPT	WINQ+PT	LWKR	BPNN	
50F1	1258.62	1126.34	1111.6	1111.6	1092.82	50F26	1774.12	1679.06	1540.98	1540.98	1540.98
50F2	1822.46	1783.34	1756.04	1700.62	1680.32	50F27	2363.84	2250.56	2129.72	2129.72	2105.04
50F3	1442.54	1378.4	1347.78	1347.78	1272.62	50F28	1851.14	1728.36	1666.48	1715.28	1662.92
50F4	2051.6	1902.76	1826.3	1826.3	1810.28	50F29	1947.46	1977.44	1876.34	1814.5	1804.70
50F5	2032.9	2030.58	1962.68	1985.78	1930.58	50F30	2052.44	2139.38	2073.84	2006.34	2004.38
50F6	2205	2092.04	2011.22	1928.54	1920.02	50F31	1917.3	1962.5	1810.08	1775.84	1775.84
50F7	1779.68	1827.26	1749.02	1673.34	1673.34	50F32	2280.92	2073.72	2086.36	2265.38	2070.38
50F8	2192.38	1949.8	1941.28	1941.28	1914.10	50F33	1852.84	1900.9	1897.68	1821.36	1810.70
50F9	2491.36	2301.16	2388.74	2388.74	2276.36	50F34	2115.08	2027.34	1994.06	1994.06	1942.26
50F10	2022.3	1987.08	1960.96	1878.02	1877.92	50F35	2281.58	2254.2	2054.12	2054.12	2052.00
50F11	2111.84	2004.24	1970.34	1975.08	1952.14	50F36	1789.68	1854.4	1727.32	1727.32	1663.20
50F12	1701.34	1779.46	1637.22	1637.22	1579.18	50F37	2082.58	2091.44	2040.54	1958.48	1958.48
50F13	1801.34	1656.96	1638.82	1638.82	1630.08	50F38	1961.8	1853.54	1772.26	1772.26	1720.42
50F14	2254.12	2201.16	2160.08	2071.16	2071.16	50F39	2150.1	2139.1	2153.04	2038.56	2033.08
50F15	2042.28	2055.64	1940.16	1940.16	1886.16	50F40	2329.06	2134.48	2089.64	2089.64	2080.86
50F16	2327.5	2147.8	2092.14	2134.22	2062.82	50F41	2040.58	1939.46	1863.86	1863.86	1856.58
50F17	2073.26	1922.46	1878.52	1935.8	1871.16	50F42	2162.28	2011.5	1990.16	1966.62	1966.62
50F18	2326.98	2236.56	2066.92	2067.68	2066.92	50F43	2208.6	2020.3	1956.88	1981.08	1946.96
50F19	2121.58	1973.62	1962.84	1962.84	1926.58	50F44	2138.88	2041.98	1982.9	1982.9	1982.90
50F20	1847.28	1929.04	1695.08	1651.9	1651.90	50F45	2351.62	2151.16	2042	2045.08	2016.58
50F21	2062.26	1979.3	1884.82	1857.3	1857.30	50F46	1921.72	1867.52	1812.36	1830.46	1798.40
50F22	1796.18	1902.14	1837.14	1730.46	1729.62	50F47	1858.82	1884.4	1745.54	1688.36	1681.46
50F23	1916.54	1930.78	1873.16	1727.46	1727.46	50F48	2229.18	2051.36	1984.04	2011	1972.10
50F24	2302	2102.06	2088.08	2088.08	2060.26	50F49	1799.56	1677.2	1708.88	1669.98	1642.24
50F25	2214.38	2212.34	2125.26	2055.04	2055.04	50F50	2125.22	2136.08	2016.88	2023.5	1959.56
						Total	**101784.1**	**98259.69**	**94922.15**	**94021.89**	**92624.78**

Table 5.4 Mean flowtime for individual test problems for n=50 (set no. 9 in Table 5.3).

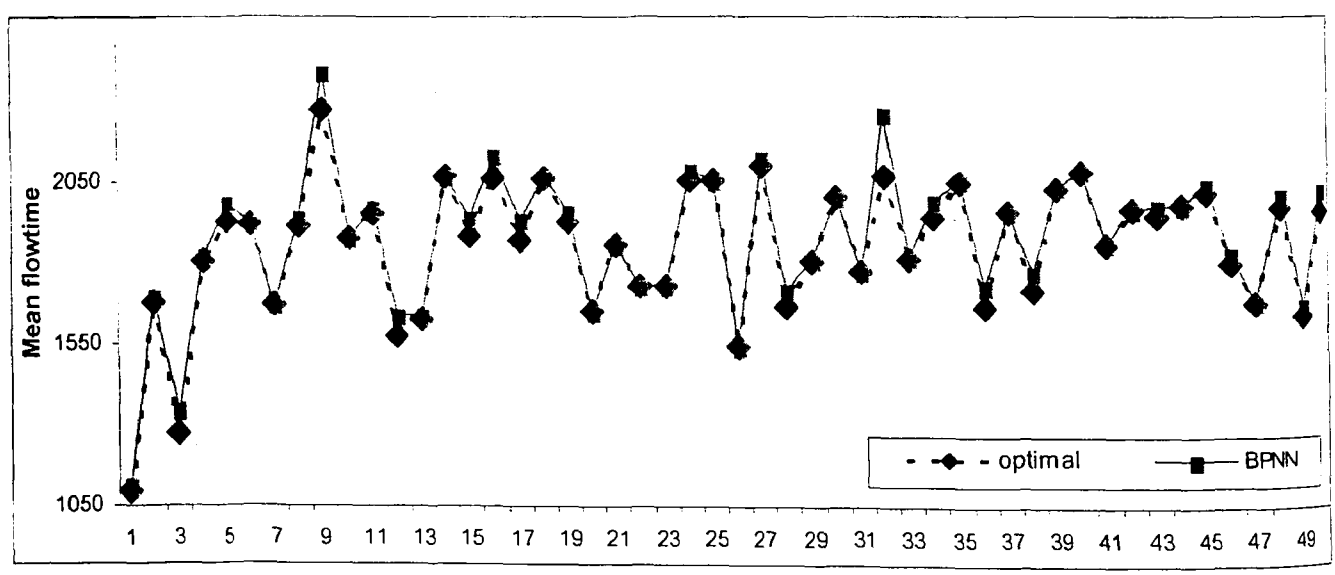


Fig. 5.2 BPNN mean flowtime compared to mean flowtime from optimal rule combinations for n=50.

5.5 Further Discussion on Results

Computational results from Table 5.1 and Appendix E1 indicate that the trained neural network has the capability of selecting an appropriate combination of dispatching rules in more than 75% of all the 700 problem instances for the performance criterion of minimizing makespan. Moreover, the neural network provides the required result more quickly as compared to enumerating all possible combinations. Further, the efficacy of the proposed neural network approach to find the best combination of dispatching rules does not significantly decrease as the number of jobs increase. Figure 5.3 compares BPNN results with the total makespan of the optimal combinations carried out for the various number of jobs (n) (ranging from a minimum of 10 to a maximum 100) for a 5-machine job shop. It can be seen that BPNN performed consistently better than the three other dispatching rules for all n.

Figure. 5.3 portrays that, as the number of jobs increases, the difference between BPNN and other dispatching rules decreases (particularly in the case with SPT). This explains SPT's improved performance under higher shop congestion levels; however, this does not imply that the trained network is weakening. In case of SPT, the job with the smallest processing time is given preference over other jobs so that the amount of time and the number of jobs waiting in the queue are reduced, and thus the desired objective of minimizing makespan is achieved well.

Further observations from Figure 5.3 show that, as the number of jobs increases, SPT also improves in comparison to MWKR. This is because jobs with a larger total remaining processing time may be waiting in queue for a longer time as compared to the jobs with the smallest processing time when SPT is used. Also, the neural network can be

seen as gradually losing its advantage for the larger number of jobs due to the improved performance of SPT for larger n.

For the performance objective of minimizing the mean flowtime, a study is carried out using the results from Table 5.3. The difference between optimal and BPNN mean flowtime for the number of jobs (n) ranging from a minimum of 10 to a maximum of 100 is presented in Figure 5.4 (on the next page).

It is evident from Figure 5.4 that the neural network's effectiveness is still maintained for large size problems. It is also observed that LWKR works well for both small and large size problems. However, the trained neural network proves that a combination of three competing dispatching rules in the job shop generate better mean flowtime for both small and large size problems. It is also evident from this study that the neural network is fairly consistent while others tend to improve only when n gets bigger.

From Figures 5.3 and 5.4 there is some indication that MWKR (for makespan minimization) and LWKR (for flowtime minimization) become more competitive with respect to the BPNN as n grows large. Therefore, it is of interest to determine whether the difference between these dispatching rules and BPNN is significantly different in job shops processing a large number of jobs. This is done by means of ANOVA for the results from the largest problems tested, the 100-job problems.

5.6 Analysis Of Variance (ANOVA)

The trained neural network is tested with different sets of jobs. It is desirable to test the significance in the difference between the observed results. The technique used on the post-trained neural network is a one factor ANOVA where a comparison of all means is

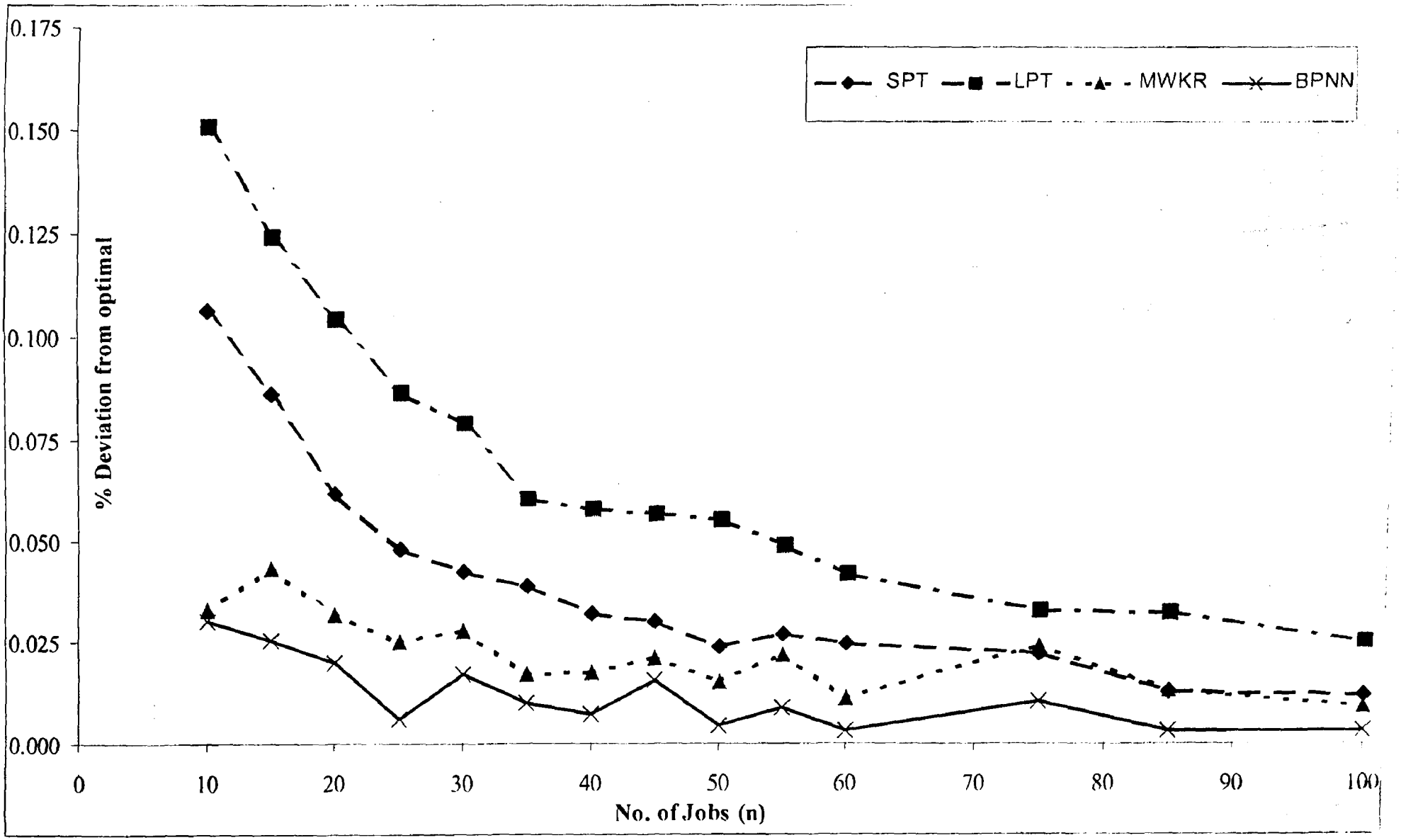


Fig. 5.3 Comparison of BPNN, SPT, LPT and MWKR with respect to optimal total makespan (ref. Table 5.1).

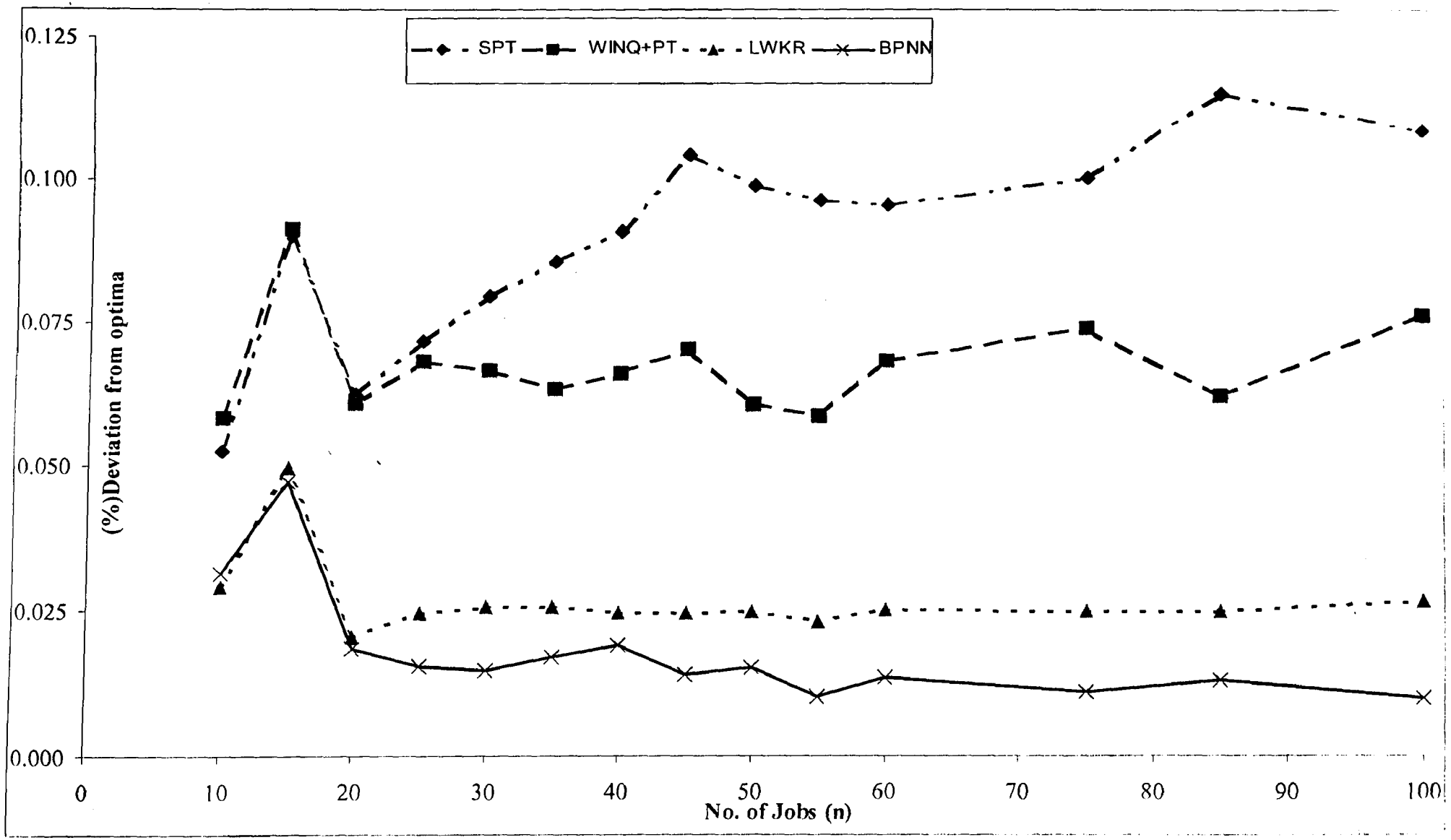


Fig. 5.4 Comparison of BPNN, SPT, WINQ+PT and LWKR with respect to optimal total mean flowtime (ref. Table 5.3).

done separately for the two objectives of minimizing makespan and mean flowtime.This means to formulate a hypothesis test and to decide whether or not there is sufficient evidence to reject a null hypothesis that there is no significant difference between the means.

While designing the experiment, the objective was to compare the performances of dispatching rules, which includes either single or a combination of dispatching rules. The " between dispatching rules" source of variation, which may be assumed to be due to changes in dispatching rules or a combination of dispatching rules, is of a different kind from that due to sampling and analytical errors. Following are the assumptions considered while carrying out the one factor ANOVA.

5.6.1 Assumptions

- The population from which the samples obtained are normally distributed.
- The populations have the same standard deviation (σ).

5.6.2 Data Representation

In this case, there are four different treatments (K=4), the first three representing different dispatching rules and the fourth one representing a neural combination of dispatching rules. Consider 10 different random samples. Each sample is composed of 50 problems for n=100 (as explained previously in section 5.6). Each entry is categorized as a value of x_{ij}, which is the average makespan or flowtime on the j^{th} individual problem taken from i^{th} sample.

5.6.3 General Logic - Analysis Of Variance

In analysis of variance, between-group variance (difference between treatments) is tested for dissimilarity from within group variance. If the null hypothesis is true, both the above-mentioned variances are about the same, and all the variation can be attributed to random variation. On the other hand, if the between-group variance in comparison to the within-group is larger, then chances are that the samples do not come from populations with equal means. Together, the within-group and between-group variations are the two sources that contribute to the total variation. F is a statistic that represents ratio of two variances:

$$F_{ratio} = \frac{\text{Between - group Variance}}{\text{Within - group Variance}} \quad \ldots. \quad (5.2)$$

If the group means are equal, then F will equal 1.0 and the null hypothesis can be accepted. However, when the sample means are different, the difference can be attributed to the effect of the independent variable plus the sampling error. Thus,

$$F_{ratio} = \frac{\text{Effect of independent variable} + \text{Sampling error}}{\text{Sampling error}} \quad \ldots. \quad (5.3)$$

5.6.4 ANOVA Calculations

A one-way ANOVA studies the effect of a single independent variable on a dependent variable. The computational formula for the $F_{statistic}$ involves the sum of squares, which is the sum of squared deviations around the mean. There are two sum of squares that are considered: (1) the total of the sum of squares within the groups (SS_{total}); this is based on the deviation between each observation and $\overline{X_{..}}$, and (2) the sum of squares for the group mean relative to the grand mean (SS_{group}). Thus,

(a) $SS_{total} = \sum X^2 - \frac{(\sum X)^2}{N}$ .... (5.4)

(b) $(SS_{group}) = \sum n_j (\overline{X_j} - \overline{X})^2$ .... (5.5)

Where, $\overline{X}$ is the grand mean.

$\overline{X_j}$ is the group mean.

(c) The total variance is composed of SS_{group} & SS_{error}, so that

$SS_{error} = SS_{total} - SS_{group}$ .... (5.6)

Next is the computation of mean square obtained by dividing the sum of the squared deviations by the degrees of freedom. Thus,

(d) $MS_{group} = \frac{SS_{group}}{df_{group}}$ .... (5.7)

(e) $MS_{error} = \frac{SS_{error}}{df_{error}}$ .... (5.8)

The degrees of freedom are:

(f) df_{group} = K – 1 degrees of freedom between groups.

(g) df_{total} = N – 1 degrees of freedom within groups.

(h) df_{error} = N - K

The F. statistic is the ratio of the group mean square to the mean square error. Thus,

$F. = \frac{MS_{group}}{MS_{error}}$ .... (5.9)

5.6.5 Hypothesis

For the one factor ANOVA the null hypothesis is that the population means (μ) are equal, which is stated as follows:

$\mathbf{H_0}$: $\mu_1 = \mu_2 = \ldots = \mu_k$

And the alternative hypothesis is:

$\mathbf{H_1}$: At least two means are significantly different.

There are several alternative ways in which the null hypothesis may be false. For example, only $\mu_1 \neq \mu_2$ or $\mu_3 \neq \mu_4$ or all four population means are unequal, and so on. The alternative hypothesis does not distinguish among these various possibilities, but rather asserts that a relationship exists between the independent and dependent variables such that the population means are not equal.

5.7 ANOVA Results For Makespan

The experiment is carried out with 10 samples, each sample consisting of fifty 100-job problems. Thus, 500 problems are considered for this experiment. Table 5.5 presents the average makespan in each of the samples, where three independent dispatching rules are applied exclusively, along with the combination of dispatching rules suggested by the neural network (BPNN).

The ANOVA is performed using the single factor ANOVA function in Microsoft Excel with a level of significance $\alpha = 0.05$. Table 5.6 displays the ANOVA results.

From the above results, the F_o statistic, $F > F_{0.05,3,36}$, and therefore the null hypotheses H_O can be rejected. It may be concluded that a significant difference exists among the dispatching rules for the performance objective of minimizing makespan.

Sample No.	SPT	LPT	MWKR	BPNN
1	7133.1	7247.74	7142.52	7082.22
2	7236.66	7336.36	7288.96	7218.88
3	7190.48	7310.94	7230.04	7144.34
4	7247.64	7377.3	7246.56	7202.06
5	7100.54	7246.58	7238.24	7116.42
6	7090.24	7234.74	7202.48	7082.54
7	7257.36	7367.54	7231.96	7185.02
8	7157.24	7237.14	7197.3	7116.6
9	7228.22	7369.08	7232.66	7165.82
10	7188.88	7312.96	7231.38	7172.62

Table 5.5 Experimental data for ANOVA (average makespan).

SUMMARY

Groups	Count	Sum	Average	Variance
SPT	10	71830.36	7183.036	3697.794649
LPT	10	73040.38	7304.038	3394.519951
MWKR	10	72242.1	7224.21	1441.709533
BPNN	10	71486.52	7148.652	2316.117351

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	134363.4292	3	44787.80973	16.51141962	6.5068E-07	2.866265447
Within Groups	97651.27336	36	2712.535371			
Total	232014.7026	39				

Table 5.6 ANOVA results for the performance objective of minimizing makespan.

However, it is still unknown which rules are significantly different from the others. Therefore, a multiple comparison procedure using Fisher's Least Significant Difference (LSD) method (also referred to as the protected t-test) is carried out to compare two group means simultaneously. This tests the significant difference between the means. The formula for the LSD procedure is:

$$t = \frac{\left(\overline{x_1} - \overline{x_2}\right)}{\sqrt{MS_w}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)} \quad \ldots \quad (5.10)$$

Where $\overline{x_1}, \overline{x_2}$ = mean for group1 and group 2 respectively.

MS_{error} = mean square within groups (from ANOVA result).

n_1. n_2 = Sample sizes for group1 and 2.

In this case BPNN results are compared with the results of all three different dispatching rules and the computation of Fisher's LSD is presented in Table 5.7

	BPNN	SPT	LPT	MWKR
Mean	$\overline{y_1}$ = 7148.652	$\overline{y_2}$ =7183.036	$\overline{y_3}$ =7304.038	$\overline{y_4}$ =7224.21
Sample size	n_1 = 10	n_2 = 10	n_3 = 10	n_4 = 10
MS_{error}	2712.535371			
Pairs	Calculation of protected t			
BPNN and SPT	$t = \frac{(7148.652 - 7183.036)}{\sqrt{\frac{2*2712.535}{10}}} = -1.476$			
BPNN and LPT	$t = \frac{(7148.652 - 7304.38)}{\sqrt{\frac{2*2712.535}{10}}} = -6.671$			
BPNN and MWKR	$t = \frac{(7148.652 - 7224.21)}{\sqrt{\frac{2*2712.535}{10}}} = -3.244$			

Table 5.7 LSD calculations for the minimization of makespan experiment.

The critical value for t at a significance level $\alpha = .05$ and df = 36 is t = 2.02 (from the standard t distribution table) (Montgomery, 1997).

The protected t-tests show significant differences between LPT and MWKR with respect to BPNN. On the other hand, there is no evidence of a significant difference between the group mean of SPT and BPNN. This means that SPT might be improving as the problem size increases. The results could also indicate that the neural network may be getting weaker as n grows. Similarly, it has been observed that there is a significant difference between other groups of means such as MWKR and LPT, except between SPT and MWKR. Thus, it is further evident that SPT improves as the number of jobs increase, as compared to both BPNN and MWKR.

5.8 ANOVA Results For Mean Flowtime

A similar ANOVA analysis is also carried out for the mean flowtime result. The details of the experiment and the ANOVA results are presented in Table 5.8 and Table 5.9 respectively.

Sample No.	SPT	WINQ+PT	LWKR	BPNN
1	3827.6104	3721.1716	3563.1432	3512.494
2	3902.2484	3762.8834	3597.3018	3545.122
3	3900.6868	3780.1184	3646.3354	3583.527
4	3944.2476	3773.6718	3619.7972	3578.526
5	3931.0576	3838.7032	3655.1484	3605.348
6	3963.7254	3827.3976	3677.554	3617.025
7	3861.4982	3734.0116	3567.4578	3525.153
8	3879.2966	3773.5424	3594.816	3535.001
9	3924.7204	3783.5388	3634.73	3577.858
10	3913.9456	3767.9396	3624.6554	3565.213

Table 5.8 ANOVA experimental data (average mean flowtime).

SUMMARY

Groups	Count	Sum	Average	Variance
SPT	10	39049.037	3904.9037	1633.918554
WINQ+PT	10	37762.9784	3776.29784	1295.996328
LWKR	10	36180.9392	3618.09392	1398.511358
BPNN	10	35645.2654	3564.52654	1184.850026

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	718502.3905	3	239500.7968	173.7629571	1.8166E-21	2.866265447
Within Groups	49619.4864	36	1378.319067			
Total	768121.8769	39				

Table 5.9 ANOVA results for the performance objective of minimizing mean flowtime.

From the ANOVA results, since $F > F_{0.05,3,36}$, the null hypothesis can be rejected and it can be concluded that there is a significant difference among the mean of different dispatching rules and BPNN in minimizing the mean flowtime. Fisher's LSD procedure is also carried out and Table 5.10 depicts these calculations.

The protected t-tests show that there is a significant difference between BPNN on

	BPNN	SPT	WINQ +PT	LWKR
Mean	$\bar{y}_1$ = 3564.53	$\bar{y}_2$ =3904.90	$\bar{y}_3$ =3776.29	$\bar{y}_4$ =3618.09
Sample Size	$n_1 = 10$	$n_2 = 10$	$n_3 = 10$	$n_4 = 10$
MS_{error}	1378.319			
Pairs	Calculation of protected t			
BPNN and SPT	$t = \frac{(3564.53 - 3904.90)}{\sqrt{\frac{2*1378.319}{10}}} = -20.501$			
BPNN and WINQ+PT	$t = \frac{(3564.53 - 3776.29)}{\sqrt{\frac{2*1378.319}{10}}} = -12.755$			
BPNN and LWKR	$t = \frac{(3564.53 - 3618.09)}{\sqrt{\frac{2*1378.319}{10}}} = -3.226$			

Table 5.10 LSD calculations for the mean flowtime experiment.

one hand, and all three dispatching rules, SPT, WINQ+PT and LWKR on the other.

In this chapter, the performances of different dispatching rules including BPNN are compared with the optimal results for various problem sizes. The results show that BPNN performs better than individual dispatching rules. The BPNN results were close to optimal for both minimizing makespan and mean flowtime. Also, for larger size problems, ANOVA tests were performed on both the performance objectives separately. This was for examining significant differences between different groups of means. The results show that for the makespan criterion, in the larger size problem, there is a significant difference between means of BPNN and LPT, and BPNN and MWKR, but not between SPT and BPNN. On the other hand, for the mean flowtime criterion, ANOVA shows a significant difference among all the dispatching rules and BPNN, for the large size problem. In the next chapter, a summary of the results, followed by the conclusion and further research, is discussed.

CHAPTER 6

Conclusion and Further research

6.1 Summary

This research has considered the problem of selecting an appropriate combination of dispatching/priority rules for scheduling job shops. The dispatching rule selection problem is shown to be a prediction problem that can be successfully solved using a proposed neural network. For building the neural network, simulations were carried out to constitute a training set. Once the network was trained, it was shown to be able to select well-suited dispatching rules for new problems. Computational results showed that the proposed neural network correctly predicted makespan in more than 75% of the problem instances. Further, the effectiveness and generalization capability of the proposed neural network was retained with increase in the number of jobs. Results indicated that, for the problem situation considered, the concept of using a combination of dispatching rules for different situations (number of jobs) in a neural network yields better results than using a single dispatching rule. The proposed neural network for minimizing the makespan criterion produced an average improvement of about 1.5% over MWKR (the best competing individual rule) and 6% over LPT (the poorest performer among the three rules considered), as seen in Table 5.1. On the other hand, for minimizing the mean flowtime objective, the proposed neural network performed better by an average of 1% over the best competing individual rule (LWKR), and by 6.5% over SPT (the poorest performing rule considered), according to the results in Table 5.3.

In this study, the test problems were randomly generated, and a small fraction of the 120 possible job routes for five machines were considered in each problem instance. This simulates a realistic scenario where jobs with the same routes belong to either a part family or a group, and have similar sequences of operations but different processing times.

6.2 Conclusion

The proposed neural network is an input-output model. In this research the input to the proposed network consists of total processing time, variance of processing time and the mean routing order on each machine. The trained neural network identifies such characteristics and quickly assigns one of three dispatching rules to use on each machine. It has been observed during the study that if only a single dispatching rule like smallest processing time (SPT) is applied on all five machines, than the optimality of the desired performance objective (in this case minimizing makespan) is not always achieved in the shop. On the other hand, if neural selections of dispatching rules are applied to each of the machines, then better results are obtained as compared to using a single dispatching rule.

Although the neural network was trained with small job numbers, the efficacy of the trained neural network for larger size problems appears to be maintained. For example, the BPNN results deviate only 0.429 % from the total of the makespans achieved by the optimal mix of the three dispatching rules for job shops with n=50.

Hence, the BPNN, from a practical application point of view, can be used as a tool to aid scheduling decisions in a plant, or can be embedded in a production computer integrated manufacturing system for automated and dynamic selection of appropriate

priority rules. The developed model offers significant advantages regarding time consumption and simplicity for scheduling new job shop problems.

6.3 Further Research

The suggested methodology is not restricted to a static job shop model, but there are a number of fruitful directions that can be identified for further research from this study, some of which are:

1) Dynamic job shop problems.
2) Different performance objectives.
3) Relaxation of assumptions.
4) Number of dispatching rules.
5) Job shops with more machines.

The details of each future direction are as follows:

The present study can be extended to consider a dynamic job shop problem, where the arrival times of jobs are not known (in advance) before scheduling. The jobs may arrive at any time; in such a real-world situation, the job characteristics are fed to the neural network. The trained network can decide in real-time when to change rules on specific machines in response to the changing characteristics of the work-in process so as to better meet the performance objectives. It is worth investigating how this network trained for a static job shop, will perform when subjected to a dynamic environment.

A similar neural network approach can be used for other performance objectives where the jobs will have an additional characteristic such as due date, like minimizing mean tardiness, minimizing number of tardy jobs or minimizing the cost. The current study made some assumptions, like no machine breakdowns, material-handling systems

are available at all times, etc. The neural network can be trained by relaxing such assumptions in order to capture more realistic real world situations. In a flexible manufacturing system that resembles the job shop model, if one of the material handling systems breaks down, then in such a situation an intelligent neural network can optimize the job sequence and thus the desired performance objective.

In the present study, a set of three dispatching rules was considered for two different performance objectives, minimizing makespan and mean flowtime. The study can be extended to test the possibilities of getting better solutions if combinations of more than three dispatching rules are used for the same or different performance objectives.

In the present study, a five-machine job shop problem was considered; a similar approach can be extended to a higher number of machines, for example, ten machines for the same or different performance objectives.

Summing up, the objective/purpose of this research was to develop and train an artificial neural network to select the best combination among three dispatching rules for a five-machine job shop problem and two different performance criteria. Experimental results showed that the trained neural network was able to successfully predict good rules to use on each machine, leading to better satisfaction of the performance criteria when compared with the alternative of using one identical rule on all machines.

References

Baker K. (2002) *Elements of sequencing and scheduling.* Hanover N.H., ISBN 0963974610.

Bialey D. and Thompson D. (1990) How to develop neural-network. Al expert, vol.5 (3), pp.38-47.

Blackstone J., Phillips D. and Hogg G. (1982) A state of-the-art survey of dispatching rules for manufacturing job shop operations. International journal of production research, vol.20 (1), pp.27-45.

Blazewicz J., Domschke W. and Pesch E. (1996) The job shop scheduling problem: conventional and new solution techniques. European journal of operations research, vol. 93, pp.1-33.

Conway R., Maxwell W.and W Miller L. (1967) *Theory of Scheduling.* Addison-Wesley.

French S. (1982) *Sequencing and scheduling: An introduction to the mathematics of the job shop.* John Wiley & sons Inc., New York.

Foo S.Y. and Takefuji Y. (1988a) Stochastic neural networks for solving job shop scheduling: Part 1, Problem definition. Proceedings of the 1988 IEEE international conference on neural networks, San Diego, vol.2, pp.275-282.

Foo S.Y. and Takefuji Y. (1988b) Stochastic neural networks for solving job shop scheduling: Part 2, Architecture and simulations. Proceedings of the 1988 IEEE international conference on neural networks, San Diego, vol.2, pp.283-290.

Garson D. (1999) *Neural networks an introductory guide for social scientists.* SAGE publications, ISBN 0-7619-5731-6.

Klimasauskas, C.C. (1992) Applying neural networks in finance and Investing. R.R Trippi and E. Turban (eds), Chicago: Probus, pp.47-72.

Jain A. and Meeran S. (1998) Job-shop scheduling using neural networks. International journal of production research, vol.36 (5), pp.1249-1272.

Jones A., Rabelo L. and Yih Y. (1996) Job shop scheduling. Encyclopedia of operations research, (**http://www.mel.nist.gov/msidlibrary/doc/jobshop1.pdf**).

Kiran, A. S. and Smith M. L. (1984) Simulation Studies in Job Shop Scheduling-I. Computer. & industrial engineering, vol.8, pp.87- 93.

Kaastra I. and Boyd M. (1996) Designing a neural network for forecasting financial and economic time series. Neurocomputing, vol.10 (3), pp.215-236.

Kelton D., Sadowski R and Sadowski D. (2002) *Simulation with Arena.* McGraw-Hill series, ISBN 0-07-239270-3.

Kumar H and Srinivasan G. (1996) A genetic algorithm for job shop scheduling – A case study. Computers in industry, vol.31, pp.155-160.

Montgomery D. (1997) *Introduction to statistical quality control - Third edition.* John Wiley & Sons, Inc New York.

Neuroshell 2 software manual. Ward systems group, Inc. **http://www.wardsystems.com**.

Panwalkar S. and Iskander W. (1977) A survey of scheduling rules. Operations research, vol.25 (1), pp. 45-61.

Pierreval H. (1992) Training a neural network by simulation for dispatching problems. IEEE Proceeding of the third rensselear international conference on computer integrated engineering, pp.332-336.

Pierreval H. (1993) Neural network to select dynamic scheduling heuristics. Journal of decision science, vol.2 (2), pp.173-190.

Pinadeo M. (1995) *Scheduling theory, algorithms and systems.* Prentic Call Englewood Cliffs, NJ.

Rabelo L. (1990) A hybrid artificial neural networks and knowledge-based expert systems approach to flexible manufacturing system scheduling. PhD dissertation.

Raghu. T.S. and Rajendran. C. (1993). An Efficient Dynamic Dispatching Rule for Scheduling in a job shop. International Journal of Production Research, vol.28, pp.2277-2292.

Rajendran C. and Holthaus O. (1999) A comparative study of dispatching rules in dynamic flowshops and job shops. European journal of operational research, vol.116, pp. 156-170.

Reinhardt M. (1990) *Neural networks An Introduction*. Springer-verlag, Berlin Heidelberg.

Sabuncuoglo I. and Gurgun B. (1996) A neural network model for scheduling problems. European journal of operational research, vol.93, pp.288-299.

Sabuncuoglo I. (1998) Scheduling with neural networks: a review of the literature and new research directions. Production planning & control, vol.9 (1), pp. 2-12.

Satake T., Morikawa K. and Nakamura N. (1994) Neural network approach for minimizing the makespan of the general job-shop. International journal of production economics, vol.33, pp. 67-74.

Sim S.K., Yeo T.Y and Lee W.H. (1994) An expert neural network system for dynamic job shop scheduling. International journal of production research, vol.32 (8), pp.1759-1773.

Smith K. (1999) *Introduction to Neural networks and data mining for business applications*. Eruditions publishing. ISBN 1 864910046.

Subramaniam V., Ramesh T., Lee K.G., Wong S.Y. and Hong S G. (2000) Job shop scheduling with dynamic fuzzy selection of dispatching rules. Advanced manufacturing technology, vol.16, pp.759-764.

Swingler K. (1999) *Applying Neural networks A project guide*. Morgan kaufman publishers, Inc, California.

Waikar A., Saker B. and Lal A. (1995) A comparative study of some priority dispatching rules under different shop loads. Production planning and control, vol.6 (3), pp.301-310.

Zhang H.C and Huang S.H. (1995) Applications of neural networks in manufacturing: a state of art survey. International journal of production research, vol.33 (3), pp.705-728.

Appendix A1: Problem generation

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

main(void)
{
FILE *scheda,*combo;
float TF,RDD,UL;
int Y[101][21],P[21],route_choice[26],load_dist[16],routes,Counter;
int i,k,n,m,r,reps,Z,X,Wf,Wt,PP,g,q,Liner,lcounter,a1,a2,a3,a4,a5;
char FFname[12],dest[30],Fdat[12],gstr[4],str1[10],str2[10];

if((combo = fopen("c:\\tc\\bin  jobshops\\perm5!.dat","r")) == NULL)
  { fprintf(stderr,"Cannot open output file . \n");
   return 1;
  }

for(i=1;i<=100;++i)
 for(k=1;k<=21;++k)
    Y[i][k] = 0;

printf("What is the number of JOBS ? \n");
scanf("%d",&n);

printf("How many test problems ? \n");
scanf("%d",&reps);
m=5;
srand(121);

          for(g=1;g<=reps;++g)
             {
          strcpy(str1,"t");
          itoa(g,gstr,10);
          strcat(str1,gstr);
          strcpy(Fdat,str1);
          strcat (Fdat,".dat");
strcpy(dest,"c:\\tc\\bin\\jobshops\\data\\");
          strcat(dest,Fdat);
if((scheda = fopen(dest,"w")) == NULL)
            { fprintf(stderr,"Cannot open output file . \n");
             return 1;
            }
          routes = random(6)+5;
                   for(k=1; k<=21; ++k)
                     P[k] = 0;
           for(r=0;r<=25;++r)
                    route_choice[r] = 0;
                   for(r=1;r<=routes;++r)
                    route_choice[r] = random(121);
                   for(r=0;r<=15;++r)
                    load_dist[r] = 0;
                   for(r=1;r<=m;++r)
                     load_dist[r] = random(49)+10;
              fprintf(scheda,"%d %d \n", n,m);
                   for(i=1;i<=n;++i)
```

```
                for(k=1;k<=m;++k)
                  {
                   Y[i][k] = random(41) - load_dist[k];
                   P[k] = P[k] - Y[i][k];
                  }

            for(i=1; i<=n; ++i)
             {
                   Z=1000;
                   lcounter=0;
                   Liner = route_choice[random(routes)+1];
                   while( Liner != lcounter)
                         { fscanf(combo,"%d %d %d %d %d \n",&a1,&a2,&a3,&a4,&a5);
                          ++lcounter; }
                   fprintf(scheda,"%d %d %d %d %d     ",a1,a2,a3,a4,a5);
                   rewind(combo);
                   fprintf(scheda,"%d ",Y[i][a1]);
                   fprintf(scheda,"%d ",Y[i][a2]);
                   fprintf(scheda,"%d ",Y[i][a3]);
                   fprintf(scheda,"%d ",Y[i][a4]);
                   fprintf(scheda,"%d ",Y[i][a5]);
                  fprintf(scheda,"\t   %d ",Z);
                   fprintf(scheda,"\n");
             }

            fprintf(scheda,"\n\n");
        fclose(scheda);
            ++Counter;
    }
printf("counter iss : %d\n",Counter);
fclose(combo);
}
```

Appendix A2: Training set generation

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>
struct table
  {
    int jobno;
    int protime;
    int duedate;
    int slack;
    int slackp;
    int flowt;
    int tarwt;
    float neuro;
  };
struct measures
   {
    float flwtime;
    float meantard;
    int numtar;
    float maxtar;
    float makespan;
   };
struct picks
  {
   int partno;
   float costco;
  };
struct probdata
  {
   int jobno;
   int operation[11];
   int machine_no[11];
   int Due;
   int elapsed;
   int route_track;
   int flowt;
   int tarwt;
   int totalwk;
  };
struct timetrack
  {
   int endtime;
   int partnum;
   int machine;
  };
struct edata
  {
   int jobindex;
   int completion;
   int duedate;
   int elapsed;
   int flowt;
   int tarwt;
  };
struct bufs
   {
    int queue[102];
    int maxcap;
   };
```

```
float LOAD[11],PLOAD[11],maxload;
struct measures MM[10];
struct bufs buffer[11];
struct timetrack event[211];
struct edata sink[111];
struct probdata PRT[102];int dispatching_rule[16];
int objsel,consel;
int TIMENOW,dd,bst;
float Tardiness,TTard,Bcost,CCost;
int Count=0,machines,h,v;
char ncat[3];
int droutput[31];
main(void)
{
FILE *combo,*sinput,*soutput,*neuinput,*neuout;
struct table spt_sort(),edd_sort(),mslk_sort(),lpt_sort(),mslp_sort(),wspt_sort();
struct table clear_job(),update_jobs(),random_sort();
struct measures calculate_measures(),calculate_cost(),schedule_costs();
struct edata find_sched();
struct bufs
spt_buf(),lpt_buf(),twspt_buf(),CD(),tspt_buf(),edd_buf(),slack_buf(),covert_buf(),mdd_buf(),lwkr_buf(),twk_buf();
struct bufs snq_buf(),winq_buf(),append_to_buffer(),unload_machine();
int combox[244][6],k1,k2,k3,k4,k5,jj,combination;
float mc_square[11],maxstdev;
float best_result,performance;
int remove_job(),max_prot(),max_date,member_of(),Avertot;
float Mflow,Bflow;
int i,k,ki,u,n,p,q,r,t,y,j,job,sequence,next_mac,curmac,best_tardy;
int place,found;
int best_tardiness,best_flow,best_combo,mac_candidate[14],mincostjob,totpro,mincostdue,mondue;
int BEST[300][11],njobs[3];
int greater_of(),counter,tie_counter;
char FFname[12],WWR[44],dest2[30],dest1[30],dest[30],Fdat[12];
char *str1;
int combocounter=0;
    if((soutput = fopen("c:\\tc\\bin\\jobshops\\data\\result.dat","w")) == NULL)
        { fprintf(stderr,"Cannot open output file . \n");
            return 1;
        }
    if((neuinput = fopen("c:\\tc\\bin\\jobshops\\data\\neujob.inp","a")) == NULL)
        { fprintf(stderr,"Cannot open output file . \n");
            return 1;
        }
fprintf(soutput,"\n  M/C 1    M/C 2    M/C 3    M/C 4    M/C 5  \n");
fprintf(soutput,"  ---------------------------------------------------- \n");

/*
printf("Select an Objective Function : \n\n");
printf("1. Minimizing Mean Flowtime \n");
printf("2. Minimizing Makespan \n");
scanf("%d",&objsel); */
objsel = 2;
if(objsel==1)
   {
    if((neuout = fopen("c:\\tc\\bin\\jobshops\\data\\neujobf.out","a")) == NULL)
        { fprintf(stderr,"Cannot open output file . \n");
            return 1;
        }
   }
else
   {
    if((neuout = fopen("c:\\tc\\bin\\jobshops\\data\\neujobm.out","a")) == NULL)
        { fprintf(stderr,"Cannot open output file . \n");
            return 1;
```

```
        }
    }
for(k=1;k<=30;++k)
  droutput[k]=0;
best_result = 999999999.9;
jj=1;
for(k1=1;k1<=3;++k1)
 for(k2=1;k2<=3;++k2)
  for(k3=1;k3<=3;++k3)
   for(k4=1;k4<=3;++k4)
        for(k5=1;k5<=3;++k5)
          { combox[jj][1]=k1;
            combox[jj][2]=k2;
            combox[jj][3]=k3;
            combox[jj][4]=k4;
            combox[jj][5]=k5;
             ++jj; }
for(combination=1;combination<=243;++combination)
 {
 ++combocounter;
        str1 = "T30.dat";
        stpcpy(FFname,str1);
strcpy(dest,"c:\\tc\\bin\\jobshops\\data\\");
        strcat(dest,str1);
    if((sinput = fopen(dest,"r")) == NULL)
        { fprintf(stderr,"Cannot open input file . \n");
              return 1;
        }
   fscanf(sinput,"%d  %d",&Count,&machines);
   for(k=0;k<=10;++k)
        buffer[k].maxcap = 0;
  for(k=0;k<=15;++k)
         dispatching_rule[k] = 0;

  for(k=1;k<=machines;++k)
        dispatching_rule[k] = combox[combination][k];
*   for(k=1;k<=5;++k)
        { printf("%d*".dispatching_rule[k]); if(dispatching_rule[k]==0)  getchar();}
*
   for(i=1;i<=210;++i)
    event[i].endtime = event[i].partnum = event[i].machine = 0;


   for(i=1;i<=10;++i)
        for(j=0; j<=101; ++j)
          buffer[i].queue[j] = 0;
   for(i=1;i<=110;++i)
    sink[i].jobindex = sink[i].completion = sink[i].duedate = sink[i].elapsed = 0;
   for(i=1;i<=Count;++i)
    for(k=0;k<=10;++k)
          { PRT[i].operation[k] = 0;  PRT[i].machine_no[k] = 0;  mc_square[k]=0; }
         for(i=1;i<=Count;++i)
          {
           totpro = 0;
           for(k=1;k<=machines;++k)
                {
                 fscanf(sinput,"%d ",&PRT[i].machine_no[k]);
                }
           for(k=1;k<=machines;++k)
                {
                 fscanf(sinput,"%d ",&PRT[i].operation[PRT[i].machine_no[k]]);
                 totpro = totpro + PRT[i].operation[PRT[i].machine_no[k]];
                 }
           PRT[i].operation[0] = totpro;
```

```
            PRT[i].totalwk = totpro;
            PRT[i].jobno = i;
            PRT[i].elapsed = 0;
            PRT[i].route_track =0;
            PRT[i].flowt = 1;
            PRT[i].tarwt = 1;
            fscanf(sinput," %d",&PRT[i].Due);
            LOAD[machines] = 0;
            for(k=1;k<=machines;++k)
                    {
                    LOAD[k] = 0;
                    }
          }
  for(k=1;k<=machines;++k)
        for(i=1;i<=Count;++i)
                    LOAD[k] = LOAD[k] + PRT[i].operation[k];
           maxload = 0;
           for(k=1;k<=machines;++k)
         if(LOAD[k] > maxload) maxload = LOAD[k];
  for(k=1;k<=machines;++k)
fprintf(neuinput," %5.3f ",LOAD[k]/maxload);
    maxstdev=0.0;
    for(k=1;k<=machines;++k)
          {
          for(i=1;i<=Count;++i)
                  mc_square[k] = mc_square[k] + PRT[i].operation[k]*PRT[i].operation[k];
           mc_square[k] =( (mc_square[k] - LOAD[k]*LOAD[k]/Count)/(Count-1) );
           if(mc_square[k] > maxstdev)  maxstdev = mc_square[k];
     }
          for(k=1;k<=machines;++k)
                  fprintf(neuinput," %5.3f ", mc_square[k]/maxstdev);
          for(k=1;k<=machines;++k)
            {
          place=0;
          for(j=1;j<=Count;++j)
           {
                  for(p=1;p<=machines;++p)
                          if(PRT[j].machine_no[p] == k)
                              {++place; break;}
                            else ++place;
           }
                  fprintf(neuinput," %5.3f ",(float)place/(Count*machines));
                  }
    fprintf(neuinput," \n");
    fclose(sinput);
    fclose(neuinput);
/***  Assign all jobs to their initial buffers and initialize event tracking   ********/
for(i=1; i<=Count; ++i)
 {
   found = 0;
   for (k=1; k<=machines; ++k)
          if(PRT[i].operation[k] != 0 && found == 0)
            {
             append_to_buffer(PRT[i].machine_no[k],i);
             ++PRT[i].route_track;
             found = 1;
            }
 }
   for(k=1;k<=machines;++k)
    if(buffer[k].queue[1] != 0)
          add_event(0.0,k);
/*************************************************************************/
  TIMENOW =0;
  while(event[1].machine != 0)
```

```
    {
        sequence=PRT[event[1].partnum].route_track;
        curmac = event[1].machine;
        job=event[1].partnum;

/*      printf("TIME : %d - EVENT on machine %d  for PART %d
\n",event[1].endtime,event[1].machine,event[1].partnum);
        printf("TIMENOW = %d \n",event[1].endtime);  */
        TIMENOW = event[1].endtime;
            j=1;
            while(event[j].machine != 0)
                {
                event[j].endtime = event[j].endtime - TIMENOW;
                ++j;
                }
          for(j=1;j<=Count;++j)
                {
                PRT[j].elapsed = PRT[j].elapsed + TIMENOW;
                PRT[j].Due = PRT[j].Due - TIMENOW;
                }
          TIMENOW = 0;
        update_event_list();
        if(job != 0)
        {
          /*  move job to the next machine's buffer      */
          found = 0;
                if(PRT[job].operation[sequence+1] != 0 && found==0 )
                {
                  next_mac = PRT[job].machine_no[sequence+1];
                  unload_machine(curmac);
                  append_to_buffer(next_mac,job);
                  ++PRT[job].route_track;
                  if(buffer[next_mac].queue[0] == 0)
                        {
                         load_machine(next_mac,dispatching_rule[next_mac]);
                         add_event(TIMENOW +
PRT[buffer[next_mac].queue[0]].operation[next_mac],buffer[next_mac].queue[0],next_mac);
                        }
                  found=1;
                }
            if(found == 0)
                {
                  add_to_sink(job,TIMENOW,0);
                  unload_machine(curmac);
                }
            load_machine(curmac,dispatching_rule[curmac]);
            if(buffer[curmac].queue[0] != 0)
                add_event(TIMENOW +
PRT[buffer[curmac].queue[0]].operation[curmac],buffer[curmac].queue[0],curmac);
        }
    else
        {
          load_machine(curmac,dispatching_rule[curmac]);
          add_event(TIMENOW +
PRT[buffer[curmac].queue[0]].operation[curmac],buffer[curmac].queue[0],curmac);
        }
  }   /* end of combination for loop                */
   schedule_costs();
 /*  printf("mean flowtime = %7.2f \n",MM[1].flwtime);
   printf("makespan = %7.2f \n",MM[1].makespan);
   printf("number of jobs tardy = %d \n",MM[1].numtar);
   printf("maximum job tardiness = %7.2f \n",MM[1].maxtar);
   printf("\n");
   for(i=1; i<=machines; ++i)
```

```
    printf("The maximum queue length in buffer %d = %d unints \n",i, buffer[i].maxcap);
  printf(" n");            */

  if(objsel == 1) performance = MM[1].flwtime;
            else performance = MM[1].makespan;
  if(performance < best_result - 0.00001)
            { best_result = performance;
             tie_counter = 1;
             for(u=1;u<=machines;++u)
               BEST[tie_counter][u] = dispatching_rule[u]; }
    else if (performance > best_result-0.00001 && performance < best_result+0.00001)
             {
               ++tie_counter;
               for(u=1;u<=machines;++u)
                     BEST[tie_counter][u] = dispatching_rule[u]; }
}  /****  end of the main while   ****/
  for(p=1;p<=tie_counter;++p)
    {
             for(u=1;u<=machines;++u)
            {
                      if(BEST[p][u] == 1) { fprintf(soutput," SPT     "); ++ droutput[3*u-2]; }
                      if(BEST[p][u] == 2) { fprintf(soutput," LPT     "); ++ droutput[3*u-1]; }
                      if(BEST[p][u] == 3) { fprintf(soutput," MWKR    "); ++ droutput[3*u]; }
                     }
     fprintf(soutput,"\n ");
     }
fprintf(soutput,"\n ---------------------------------------------------- \n");
if(objsel == 1)
    fprintf(soutput," mean flowtime = %7.2f  ",best_result);
 else
    fprintf(soutput," makespan = %7.2f  ",best_result);

    for(i=1;i<=3*machines;++i)
             fprintf(neuout," %5.2f ",(float)droutput[i]/tie_counter);
      fprintf(neuout,"\n ");
 fclose(neuout);
 fclose(combo);
 fclose(soutput);
 return(0);
 }
 struct bufs append_to_buffer(bnum,jobnum)
    int bnum, jobnum;
    {
     int i=1;
     while(buffer[bnum].queue[i] != 0)
             {
             ++i;
             };
             if (i > buffer[bnum].maxcap)
                      buffer[bnum].maxcap = i;
 buffer[bnum].queue[i] = jobnum;
         return;
    }
 /*  Function :  unload_machi( )  This function removes a job from machine (t)          */
    struct bufs unload_machine(finmac)
    int finmac;
    {
     int i,finjob;
     finjob = buffer[finmac].queue[0];
     LOAD[finmac] = LOAD[finmac] - PRT[finjob].operation[finmac];
     PRT[finjob].operation[0] =  PRT[finjob].operation[0] - PRT[finjob].operation[finmac];
     buffer[finmac].queue[0] = 0;
     return;
    }
```

```
add_event(time,job,mac)
int time,job,mac;
{
 int i=1;
 int p,spot;
 while (event[i].machine !=0 && event[i].endtime < time)
  {
        ++i;
  };
 spot = i;
 while (event[i].machine !=0)
        ++i;
 for(p=i;p>=spot;--p)
       event[p]= event[p-1];

 event[spot].endtime = time;
 event[spot].partnum = job;
 event[spot].machine = mac;
 return;
}
load_machine(mac,selected)
int mac,selected;
{
 int i=1;
 if(buffer[mac].queue[2] != 0)
            {
                  if(selected == 1)  spt_buf(mac);
                  if(selected == 2)  lpt_buf(mac);
                  if(selected == 3)  mwkr_buf(mac);
                  if(selected == 4)  winq_buf(mac);
                  if(selected == 5)  lwkr_buf(mac);
            }
 do
        {
          buffer[mac].queue[i-1] = buffer[mac].queue[i];
          ++i;
        }
  while (buffer[mac].queue[i-1] != 0);
 return;
}
update_event_list()
{
 int i=1;
 do
        {
          event[i-1] = event[i];
          ++i;
        }
  while (event[i-1].machine != 0);
 return;
}
add_to_sink(jobno,comp,due)
int jobno,comp,due;
{
 int u=1;
 while(sink[u].jobindex != 0)
        {
        ++u;
        };
        sink[u].jobindex = jobno;
        sink[u].completion = comp;
        sink[u].duedate = due;
        sink[u].flowt = PRT[jobno].flowt;
        sink[u].tarwt = PRT[jobno].tarwt;
```

```
        sink[u].elapsed = PRT[sink[u].jobindex].elapsed + comp;
 return;
}
 dismiss_job(j)
 int j;
 {
        int i;

        for (i=1;i<=machines;++i)
         PRT[j].operation[i] = 0;
        PRT[j].Due = 0;
        PRT[j].flowt = PRT[j].tarwt = 0;
        PRT[j].jobno = 0;
 return(0);
 }
 struct bufs spt_buf(bufno)
 int bufno;
 {
        #define FALSE 0
        #define TRUE 1

        int j,k, sorted = FALSE;
        float trig1,trig2;
        int temp;

        int n = 1;
        while(buffer[bufno].queue[n] !=0)
            ++n;
            --n;
        while (!sorted)
         {
          sorted = TRUE;
          for (j=1; j<n; ++j)
             if (PRT[buffer[bufno].queue[j]].operation[bufno] > PRT[buffer[bufno].queue[j+1]].operation[bufno])
                    {  temp = buffer[bufno].queue[j];
                               buffer[bufno].queue[j]=buffer[bufno].queue[j+1];
                               buffer[bufno].queue[j+1]=temp;
                               sorted = FALSE;
                    }
         }
return;
}
 struct bufs lpt_buf(bufno)
 int bufno;
  {
        #define FALSE 0
        #define TRUE 1
        int j,k, sorted = FALSE;
        float trig1,trig2;
        int temp;
        int n = 1;
        while(buffer[bufno].queue[n] !=0)
            ++n;
            --n;
        while (!sorted)
         {
          sorted = TRUE;
          for (j=1; j<n; ++j)
   if (PRT[buffer[bufno].queue[j]].operation[bufno] < PRT[buffer[bufno].queue[j+1]].operation[bufno])
                    {
                               temp = buffer[bufno].queue[j];
                               buffer[bufno].queue[j]=buffer[bufno].queue[j+1];
                               buffer[bufno].queue[j+1]=temp;
                               sorted = FALSE;
```

```
                ;
        }
return;
}
  struct bufs mwkr_buf(bufno)
  int bufno;
  {
        #define FALSE 0
        #define TRUE 1

        int j, sorted = FALSE;
        int temp;

        int n = 1;
        while(buffer[bufno].queue[n] !=0)
            ++n;
            --n;

        while (!sorted)
         {
          sorted = TRUE;
          for (j=1; j<n; ++j)
            if (PRT[buffer[bufno].queue[j]].operation[0] < PRT[buffer[bufno].queue[j+1]].operation[0])
                    {
                            temp = buffer[bufno].queue[j];
                            buffer[bufno].queue[j]=buffer[bufno].queue[j+1];
                            buffer[bufno].queue[j+1]=temp;
                            sorted = FALSE;
                    }
         }
return;
}

struct bufs winq_buf(bufno)
  int bufno;
   {
        #define FALSE 0
        #define TRUE 1

        int j,k, sorted = FALSE;
        int temp;

        int n = 1;
        while(buffer[bufno].queue[n] !=0)
            ++n;
            --n;

        while (!sorted)
         {
          sorted = TRUE;
          for (j=1; j<n; ++j)

            if (work_in_next_queue( buffer[bufno].queue[j] , bufno) > work_in_next_queue(
buffer[bufno].queue[j+1] , bufno))
                    {
                            temp = buffer[bufno].queue[j];
                            buffer[bufno].queue[j]=buffer[bufno].queue[j+1];
                            buffer[bufno].queue[j+1]=temp;
                            sorted = FALSE;
                    }
         }
return;
}
```

```
work_in_next_queue(jobnumber,present_mac)
 int jobnumber, present_mac;
 {
 int q,k1,numb,waiting_job,next_mac,workinqueue=0;

 for(q=1;q<=Count;++q)
  if(PRT[q].jobno == jobnumber) break;

 for(k1=1;k1<machines;++k1)
  if(PRT[q].machine_no[k1] == present_mac)
        {
         numb = 1;
     next_mac = PRT[q].machine_no[k1+1];
         while(buffer[next_mac].queue[numb] !=0)
          {
       waiting_job = buffer[next_mac].queue[numb];
          workinqueue = workinqueue + PRT[waiting_job].operation[next_mac];
              ++numb; }
              workinqueue = workinqueue + PRT[q].operation[PRT[q].machine_no[k1]];
     return(workinqueue);
    }
  else
       return(0);
 }
  struct bufs lwkr_buf(bufno)
  int bufno;
  {
       #define FALSE 0
       #define TRUE 1

       int j, sorted = FALSE;
       int temp;

       int n = 1;
       while(buffer[bufno].queue[n] !=0)
           ++n;
           --n;

       while (!sorted)
        {
         sorted = TRUE;
         for (j=1; j<n; ++j)
           if (PRT[buffer[bufno].queue[j]].operation[0] > PRT[buffer[bufno].queue[j+1]].operation[0])
                {
                        temp = buffer[bufno].queue[j];
                        buffer[bufno].queue[j]=buffer[bufno].queue[j+1];
                        buffer[bufno].queue[j+1]=temp;
                        sorted = FALSE;
                }
        }
 return;
 }
```

Appendix A3: Comparison of NN results with other dispatching rule results

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include <neuro.h>
struct table
  {
    int jobno;
    int protime;
    int duedate;
    int slack;
    int slackp;
    int flowt;
    int tarwt;
    float neuro;
  };
struct measures
   {
    float flwtime;
    float meantard;
    int numtar;
    float maxtar;
    float makespan;
   };
struct picks
  {
   int partno;
   float costco;
  };
struct probdata
  {
   int jobno;
   int operation[11];
   int machine_no[11];
   int Due;
   int elapsed;
   int route_track;
   int flowt;
   int tarwt;
   int totalwk;
  };
struct timetrack
  {
   int endtime;
   int partnum;
   int machine;
  };
struct edata
  {
   int jobindex;
   int completion;
   int duedate;
   int elapsed;
   int flowt;
   int tarwt;
  };
struct bufs
  {
   int queue[102];
   int maxcap;
```

```
};
float LOAD[11],PLOAD[11],maxload;
struct measures MM[10];
struct bufs buffer[21];
struct timetrack event[211];
struct edata sink[111];
struct probdata PRT[101];int dispatching_rule[16];
int rulesel,consel;
int TIMENOW,dd,bst;
float Tardiness,TTard,Bcost,CCost;
int Count=0,machines,h,v;
char ncat[3];
int droutput[31];
main(void)
{
FILE *sinput,*soutput;
struct table spt_sort(),edd_sort(),mslk_sort(),lpt_sort(),mslp_sort(),wspt_sort();
struct table clear_job(),update_jobs(),random_sort();
struct measures calculate_measures(),calculate_cost(),schedule_costs();
struct edata find_sched();
struct bufs
spt_buf(),lpt_buf(),twspt_buf(),CD(),tspt_buf(),edd_buf(),slack_buf(),covert_buf(),mdd_buf(),lwkr_buf(),twk_buf();
struct bufs snq_buf(),awinq_buf(),append_to_buffer(),unload_machine();
int combox[244][6],k1,k2,k3,k4,k5,jj,combination;
float mc_square[11],maxvar,makespans,flowtimes;
float best_result,performance;
int remove_job(),max_prot(),max_date,member_of(),Avertot;
float Mflow,Bflow;
double n_vector[16],n_output[16],maxout;
int i,k,ki,u,n,p,q,r,t,y,j,job,sequence,next_mac,curmac,best_tardy;
int place,found;
int best_tardiness,best_flow,best_combo,mac_candidate[14],mincostjob,totpro,mincostdue,mondue;
int BEST[300][11],njobs[3];
int greater_of(),counter,tie_counter;
char FFname[12],WWR[44],dest2[30],dest1[30],dest[30],Fdat[12];
char *str1;
int combocounter=0;
    if((sinput = fopen("c:\\tc\\bin\\jobshops\\data\\multest.dat","r")) == NULL)
        { fprintf(stderr,"Cannot open output file . \n");
            return 1;
        }
    if((soutput = fopen("c:\\tc\\bin\\jobshops\\data\\multest.out","w")) == NULL)
        { fprintf(stderr,"Cannot open output file . \n");
            return 1;
        }
printf("Select Dispatching Rule Policy : \n\n");
printf("1. SPT for all machines \n");
printf("3. LWKR for all machines \n");
printf("2. LPT for all machines \n");
printf("4. Neural Selection \n");
scanf("%d",&rulesel);
if(rulesel != 1 && rulesel !=2 && rulesel !=3 && rulesel !=4)
    { printf("Leaving the program . \n");
        return 1;
    }
if(rulesel == 1)
  for(k=1;k<=machines;++k)  dispatching_rule[k]=1;
if(rulesel == 2)
  for(k=1;k<=machines;++k)  dispatching_rule[k]=2;
if(rulesel == 3)
  for(k=1;k<=machines;++k)  dispatching_rule[k]=3;
makespans = flowtimes = 0;
while(!feof(sinput))
 {
```

```
fscanf(sinput,"%d %d",&Count,&machines);
if(feof(sinput))
        break;
for(k=0;k<=10;++k)
      buffer[k].maxcap = 0;
for(k=0;k<=15;++k)
       { dispatching_rule[k] = 0;  n_vector[k]=0.0; }
for(k=1;k<=machines;++k)
      dispatching_rule[k] = rulesel;
for(i=1;i<=210;++i)
 event[i].endtime = event[i].partnum = event[i].machine = 0;
for(i=1;i<=10;++i)
      for(j=0; j<=101; ++j)
        buffer[i].queue[j] = 0;
for(i=1;i<=110;++i)
 sink[i].jobindex = sink[i].completion = sink[i].duedate = sink[i].elapsed = 0;
for(i=0;i<=Count;++i)
 for(k=0;k<=10;++k)
          { PRT[i].operation[k] = 0;  PRT[i].machine_no[k] = 0;  mc_square[k]=0; }
       for(i=1;i<=Count;++i)
        {
         totpro = 0;
         for(k=1;k<=machines;++k)
                {
                 fscanf(sinput,"%d ",&PRT[i].machine_no[k]);
                }
         for(k=1;k<=machines;++k)
                {
                 fscanf(sinput,"%d ",&PRT[i].operation[PRT[i].machine_no[k]]);
                 totpro = totpro + PRT[i].operation[PRT[i].machine_no[k]];
                }
          PRT[i].operation[0] = totpro;
          PRT[i].totalwk = totpro;
          PRT[i].jobno = i;
          PRT[i].elapsed = 0;
          PRT[i].route_track =0;
          PRT[i].flowt = 1;
          PRT[i].tarwt = 1;
          fscanf(sinput," %d",&PRT[i].Due);
        }
for(k=1;k<=machines;++k)
                  LOAD[k] = 0;
          for(k=1;k<=machines;++k)
      for(i=1;i<=Count;++i)
                  LOAD[k] = LOAD[k] + PRT[i].operation[k];
          maxload = 0;
          for(k=1;k<=machines;++k)
       if(LOAD[k] > maxload) maxload = LOAD[k];
          for(k=1;k<=machines;++k)
                n_vector[k] =  LOAD[k]/maxload;
  maxvar=0.0;
  for(k=1;k<=machines;++k)
        {
         for(i=1;i<=Count;++i)
                mc_square[k] = mc_square[k] + PRT[i].operation[k]*PRT[i].operation[k];
    mc_square[k] =(mc_square[k] - LOAD[k]*LOAD[k]/Count)/(Count-1);
          if(mc_square[k] > maxvar)  maxvar = mc_square[k];
        }

         for(k=machines+1;k<=2*machines;++k)
                n_vector[k] = mc_square[k-machines]/maxvar;
          for(k=1;k<=machines;++k)
            {
         place=0;
```

```
        for(j=1;j<=Count;++j)
          {
                    for(p=1;p<=machines;++p)
                          if(PRT[j].machine_no[p] == k)
                             {++place; break;}
                            else ++place;
          }
                   n_vector[2*machines+k] = (float)place/(Count*machines);
                  }
  for(p=1;p<=15;++p)
        n_vector[p-1] = n_vector[p];
  if(rulesel == 4)
        {
          neuro_dispatch(n_vector,n_output);
   for(k=1;k<=machines;++k)
     {
          maxout=0.0;
          for(p=0;p<3;++p)
            if(n_output[3*(k-1)+p] > maxout)
                    { maxout = n_output[3*(k-1)+p]; dispatching_rule[k] = p+1;}
     }
        }
        for(i=1;i<=machines;++i)
        fprintf(soutput,"%d  ",dispatching_rule[i]);

/***  Assign all jobs to their initial buffers and initialize event tracking    ********/
```

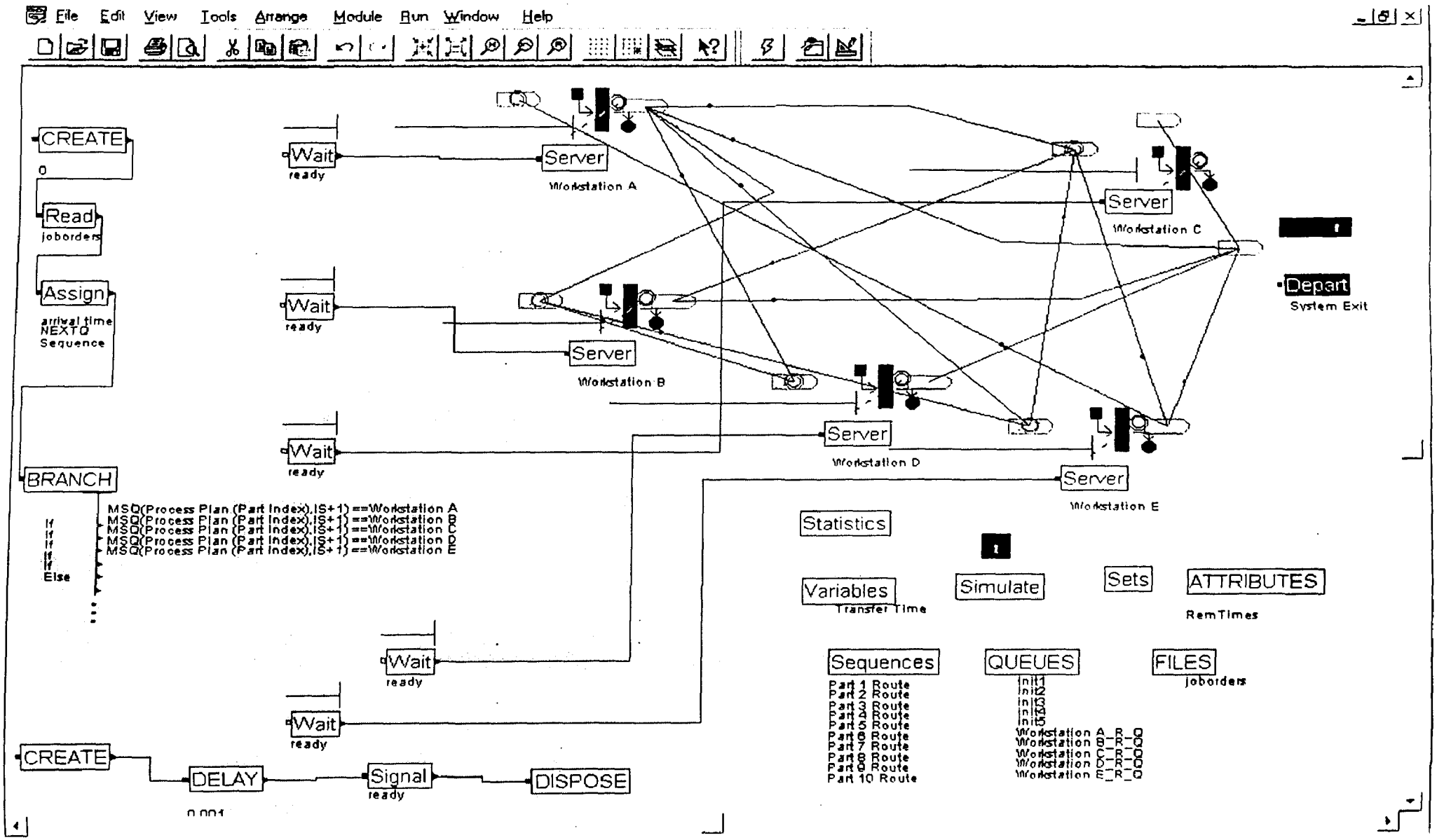
Refer Appendix A2

Appendix A4: Test Problem generation

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
/*  generates test problems for use by appendix D      */
main(void)
{
FILE *scheda,*combo;
float TF,RDD,UL;
int Y[101][21],P[21],route_choice[26],load_dist[16],routes,Counter;
int i,k,n,m,r,reps,Z,X,Wf,Wt,PP,g,q,Liner,lcounter,a1,a2,a3,a4,a5;
if((scheda = fopen("c:\\tc\\bin\\jobshops\\data\\multest.dat","w")) == NULL)
  { fprintf(stderr,"Cannot open output file . \n");
   return 1;
  }
if((combo = fopen("c:\\tc\\bin\\jobshops\\perm5!.dat","r")) == NULL)
  { fprintf(stderr,"Cannot open output file . \n");
   return 1;
  }
for(i=1;i<=100;++i)
 for(k=1;k<=21;++k)
    Y[i][k] = 0;
printf("What is the number of JOBS ? \n");
scanf("%d",&n);
printf("How many test problems ? \n");
scanf("%d",&reps);
m=5;
srand(2177);
for(g=1;g<=reps;++g)
            {
                routes = random(6)+5;
                for(k=1; k<=21; ++k)
                 P[k] = 0;
          for(r=0;r<=25;++r)
                 route_choice[r] = 0;
                for(r=1;r<=routes;++r)
                 route_choice[r] = random(121);
                for(r=0;r<=15;++r)
                 load_dist[r] = 0;
                for(r=1;r<=m;++r)
                 load_dist[r] = random(49)+10;
             fprintf(scheda,"%d %d \n", n,m);
                for(i=1;i<=n;++i)
                  for(k=1;k<=m;++k)
                    {
                     Y[i][k] = random(41) + load_dist[k];
                     P[k] = P[k] + Y[i][k];
                    }
             for(i=1; i<=n; ++i)
              {
                     Z=1000;
             lcounter=0;
          Liner = route_choice[random(routes)+1];
```

```
                while( Liner != lcounter)
                        { fscanf(combo,"%d %d %d %d %d \n",&a1,&a2,&a3,&a4,&a5);
                          ++lcounter; }
                fprintf(scheda,"%d %d %d %d %d     ",a1,a2,a3,a4,a5);
                rewind(combo);


                fprintf(scheda,"%d ",Y[i][a1]);
                fprintf(scheda,"%d ",Y[i][a2]);
                fprintf(scheda,"%d ",Y[i][a3]);
                fprintf(scheda,"%d ",Y[i][a4]);
                fprintf(scheda,"%d ",Y[i][a5]);

                fprintf(scheda,"\t   %d ",Z);
                fprintf(scheda,"\n");

        }

        fprintf(scheda,"\n\n");
        ++Counter;

    }

printf("counter iss : %d\n",Counter);
fclose(combo);
fclose(scheda);
}
```

Appendix B: Arena simulation model



Appendix C1: Determining number of hidden neurons for performance objective of Minimizing Makespan

Number of hidden layer	1
Learning rate	0.1
Momentum	0.1
Initial Weights	0.3
Pattern Selection	Random
Weights updates	Vanilla
Number of training patter	2494
Number of test	1200, combination of n=10,15,20
Number of hidden neurons	6 to 18

Appendix C2: Determining number of hidden neurons for performance objective of Minimizing Mean Flowtime

Number of hidden layer	1
Learning rate	0.05
Momentum	0.05
Initial Weights	0.3
Pattern Selection	Rotational
Weights updates	Turboprop
Number of training patter	2636
Number of test	1200, combination of n=10,15,20
Number of hidden neurons	10 to 25

Appendix D1: Neural Weights for Makespan network

```
/* Insert this code into your C program to fire the C:\NSHELL2\SPT&LPT&MWKR network */
/* This code is designed to be simple and fast for porting to any machine */
/* Therefore all code and weights are inline without looping or data storage */
/*   which might be harder to port between compilers. */

#include <math.h>
 void neuro_dispatch(double *inarray, double *outarray)
 {
 double netsum;
 double feature2[9];
 /* inarray[0] is total processing time on machine 1 */
 /* inarray[1] is total processing time on machine 2 */
 /* inarray[2] is total processing time on machine 3 */
 /* inarray[3] is total processing time on machine 4 */
 /* inarray[4] is total processing time on machine 5 */
 /* inarray[5] is variance processing time on machine 1 */
 /* inarray[6] is variance processing time on machine 2 */
 /* inarray[7] is variance processing time on machine 3 */
 /* inarray[8] is variance processing time on machine 4 */
 /* inarray[9] is variance processing time on machine 5 */
 /* inarray[10] is mean routing order of machine 1 */
 /* inarray[11] is mean routing order of machine 2 */
 /* inarray[12] is mean routing order of machine 3 */
 /* inarray[13] is mean routing order of machine 4 */
 /* inarray[14] is mean routing order of machine 5 */
 /* outarray[0] is SPT on machine 1 */
 /* outarray[1] is LPT on machine 1 */
 /* outarray[2] is MWKR on machine 1 */
 /* outarray[3] is SPT on machine 2 */
 /* outarray[4] is LPT on machine 2 */
 /* outarray[5] is MWKR on machine 2 */
 /* outarray[6] is SPT on machine 3 */
 /* outarray[7] is LPT on machine 3 */
 /* outarray[8] is MWKR on machine 3 */
 /* outarray[9] is SPT on machine 4 */
 /* outarray[10] is LPT on machine 4 */
 /* outarray[11] is MWKR on machine 4 */
 /* outarray[12] is SPT on machine 5 */
 /* outarray[13] is LPT on machine 5 */
 /* outarray[14] is MWKR on machine 5 */

  inarray[0] = 1.0 / (1.0 + exp( -( inarray[0] -  .8072053) / .1779804 ));

  inarray[1] = 1.0 / (1.0 + exp( -( inarray[1] -  .7982979) / .1812125 ));

  inarray[2] = 1.0 / (1.0 + exp( -( inarray[2] -  .8054764) / .1795151 ));

  inarray[3] = 1.0 / (1.0 + exp( -( inarray[3] -  .8004816) / .1790274 ));

  inarray[4] = 1.0 / (1.0 + exp( -( inarray[4] -  .8064976) / .1814937 ));

  inarray[5] = 1.0 / (1.0 + exp( -( inarray[5] -  .771103) / .1943457 ));

  inarray[6] = 1.0 / (1.0 + exp( -( inarray[6] -  .7586824) / .1922863 ));

  inarray[7] = 1.0 / (1.0 + exp( -( inarray[7] -  .7655738) / .1942723 ));

  inarray[8] = 1.0 / (1.0 + exp( -( inarray[8] -  .769994) / .1929265 ));
```

```
inarray[9] = 1.0 / (1.0 + exp( -( inarray[9] -  .7713115) / .1924995 ));

inarray[10] = 1.0 / (1.0 + exp( -( inarray[10] -  .5963829) / .1359357 ));

inarray[11] = 1.0 / (1.0 + exp( -( inarray[11] -  .598911) / .1376096 ));

inarray[12] = 1.0 / (1.0 + exp( -( inarray[12] -  .6040698) / .137564 ));

inarray[13] = 1.0 / (1.0 + exp( -( inarray[13] -  .596164) / .137554 ));

inarray[14] = 1.0 / (1.0 + exp( -( inarray[14] -  .6044775) / .1383298 ));

netsum =  .2456497;
netsum += inarray[0] *  2.046942;
netsum += inarray[1] * -1.739467E-02;
netsum += inarray[2] *  1.950621;
netsum += inarray[3] *  .7895958;
netsum += inarray[4] * -4.393935;
netsum += inarray[5] *  .6341079;
netsum += inarray[6] *  .7639786;
netsum += inarray[7] * -.3696065;
netsum += inarray[8] *  .1025962;
netsum += inarray[9] *  .2775704;
netsum += inarray[10] *  2.588418E-03;
netsum += inarray[11] * -.1057494;
netsum += inarray[12] *  1.179916;
netsum += inarray[13] *  .2022499;
netsum += inarray[14] * -.2809177;
feature2[0] = 1 / (1 + exp(-netsum));

netsum =  .7736111;
netsum += inarray[0] *  .2019996;
netsum += inarray[1] *  .1082652;
netsum += inarray[2] *  9.714916E-02;
netsum += inarray[3] *  .8964988;
netsum += inarray[4] *  .9009568;
netsum += inarray[5] *  1.312576E-02;
netsum += inarray[6] * -1.136535E-03;
netsum += inarray[7] *  .5345301;
netsum += inarray[8] *  .429602;
netsum += inarray[9] *  .6620308;
netsum += inarray[10] *  .3335176;
netsum += inarray[11] *  .4408983;
netsum += inarray[12] * -.8332653;
netsum += inarray[13] *  1.068497;
netsum += inarray[14] *  .8551623;
feature2[1] = 1 / (1 + exp(-netsum));

netsum =  .2736917;
netsum += inarray[0] *  1.667238;
netsum += inarray[1] *  1.011702;
netsum += inarray[2] * -.6062995;
netsum += inarray[3] * -.4746662;
netsum += inarray[4] * -6.320269E-02;
netsum += inarray[5] * -.3131128;
netsum += inarray[6] *  .4646354;
netsum += inarray[7] * -.4525496;
netsum += inarray[8] *  .4268539;
netsum += inarray[9] *  .153007;
netsum += inarray[10] *  2.659758;
netsum += inarray[11] * -1.245716;
netsum += inarray[12] *  6.915511E-02;
```

```
netsum += inarray[13] * -1.128644;
netsum += inarray[14] *  1.168092;
feature2[2] = 1 / (1 + exp(-netsum));

netsum =  .4272505;
netsum += inarray[0] *  .7500188;
netsum += inarray[1] *  1.000085;
netsum += inarray[2] * -1.751807;
netsum += inarray[3] *  .2685844;
netsum += inarray[4] *  1.027902;
netsum += inarray[5] *  .6474725;
netsum += inarray[6] *  .1521454;
netsum += inarray[7] *  .2068681;
netsum += inarray[8] * -.5180956;
netsum += inarray[9] *  .1931846;
netsum += inarray[10] * -1.069892;
netsum += inarray[11] *  .6269137;
netsum += inarray[12] *  1.828143;
netsum += inarray[13] * -1.174372;
netsum += inarray[14] *  .3147894;
feature2[3] = 1 / (1 + exp(-netsum));

netsum =  1.253435;
netsum += inarray[0] *  .4281614;
netsum += inarray[1] *  .263829;
netsum += inarray[2] *  .7757708;
netsum += inarray[3] *  .8198949;
netsum += inarray[4] *  .4531571;
netsum += inarray[5] *  .7415228;
netsum += inarray[6] *  .3905536;
netsum += inarray[7] *  .5716818;
netsum += inarray[8] *  .5664634;
netsum += inarray[9] *  .4108108;
netsum += inarray[10] *  1.020716;
netsum += inarray[11] *  .6482075;
netsum += inarray[12] *  .3916479;
netsum += inarray[13] *  .82701;
netsum += inarray[14] *  .5516309;
feature2[4] = 1 / (1 + exp(-netsum));

netsum =  1.020303;
netsum += inarray[0] *  .6176658;
netsum += inarray[1] * -.1641615;
netsum += inarray[2] *  .6528724;
netsum += inarray[3] *  .5118068;
netsum += inarray[4] *  1.187099;
netsum += inarray[5] *  .357643;
netsum += inarray[6] *  .5779907;
netsum += inarray[7] *  .6766816;
netsum += inarray[8] *  .337665;
netsum += inarray[9] *  .5777485;
netsum += inarray[10] *  .7961912;
netsum += inarray[11] *  .6893328;
netsum += inarray[12] *  .5017431;
netsum += inarray[13] * -.2587531;
netsum += inarray[14] *  .545307;
feature2[5] = 1 / (1 + exp(-netsum));

netsum = -.857754;
netsum += inarray[0] * -.7458848;
netsum += inarray[1] * -1.533975;
netsum += inarray[2] * -1.908274;
```

```
netsum += inarray[3] *  6.414888;
netsum += inarray[4] * -1.083793;
netsum += inarray[5] * -.1197545;
netsum += inarray[6] *  .1958044;
netsum += inarray[7] *  .4577059;
netsum += inarray[8] * -.8148838;
netsum += inarray[9] * -.3346533;
netsum += inarray[10] * -.4140265;
netsum += inarray[11] * -.9390895;
netsum += inarray[12] *  .1293133;
netsum += inarray[13] * -.2089598;
netsum += inarray[14] * -.150009;
feature2[6] = 1 / (1 + exp(-netsum));

netsum =  .9072358;
netsum += inarray[0] * -4.788719;
netsum += inarray[1] *  1.544868;
netsum += inarray[2] *  2.137923;
netsum += inarray[3] *  .6501429;
netsum += inarray[4] *  1.509744;
netsum += inarray[5] * -.552145;
netsum += inarray[6] *  .7359388;
netsum += inarray[7] * -.3814477;
netsum += inarray[8] *  .207289;
netsum += inarray[9] * -.3526275;
netsum += inarray[10] *  .7480308;
netsum += inarray[11] * -.7977181;
netsum += inarray[12] *  1.289536;
netsum += inarray[13] *  .1694474;
netsum += inarray[14] * -.1155627;
feature2[7] = 1 / (1 + exp(-netsum));

netsum =  .2858913;
netsum += inarray[0] * -1.584813;
netsum += inarray[1] *  3.49939;
netsum += inarray[2] * -1.439081;
netsum += inarray[3] *  .1945082;
netsum += inarray[4] * -2.07647;
netsum += inarray[5] * -6.070224E-03;
netsum += inarray[6] * -.3148423;
netsum += inarray[7] *  .3856115;
netsum += inarray[8] * -.1825979;
netsum += inarray[9] *  .2919024;
netsum += inarray[10] * -.3072858;
netsum += inarray[11] *  1.34602;
netsum += inarray[12] * -.3143554;
netsum += inarray[13] * -.6019555;
netsum += inarray[14] * -.4247503;
feature2[8] = 1 / (1 + exp(-netsum));

netsum = -.3141774;
netsum += feature2[0] *  .2389331;
netsum += feature2[1] * -.54418;
netsum += feature2[2] * -1.70218;
netsum += feature2[3] *  1.199972;
netsum += feature2[4] * -.4256195;
netsum += feature2[5] * -.2628123;
netsum += feature2[6] *  .1917116;
netsum += feature2[7] *  1.139177;
netsum += feature2[8] *  .2864685;
outarray[0] = 1 / (1 + exp(-netsum));
```

```
netsum = -.5591048;
netsum += feature2[0] * -.6884778;
netsum += feature2[1] * -.1268542;
netsum += feature2[2] * .4580247;
netsum += feature2[3] * -1.403934;
netsum += feature2[4] * -5.491569E-02;
netsum += feature2[5] * -.1763379;
netsum += feature2[6] * .2373756;
netsum += feature2[7] * 1.29709;
netsum += feature2[8] * .2638615;
outarray[1] = 1 / (1 + exp(-netsum));

netsum = .1457091;
netsum += feature2[0] * .5011446;
netsum += feature2[1] * .217678;
netsum += feature2[2] * 1.359661;
netsum += feature2[3] * .0896292;
netsum += feature2[4] * -.1945114;
netsum += feature2[5] * .3449156;
netsum += feature2[6] * -.4215003;
netsum += feature2[7] * -2.235487;
netsum += feature2[8] * -.5853001;
outarray[2] = 1 / (1 + exp(-netsum));

netsum = -.5933613;
netsum += feature2[0] * .1887896;
netsum += feature2[1] * -.496328l;
netsum += feature2[2] * 1.171266;
netsum += feature2[3] * -.2313004;
netsum += feature2[4] * -.127795;
netsum += feature2[5] * -8.012733E-02;
netsum += feature2[6] * .5547236;
netsum += feature2[7] * .5958884;
netsum += feature2[8] * -1.437345;
outarray[3] = 1 / (1 + exp(-netsum));

netsum = -2.756067E-03;
netsum += feature2[0] * -.2146351;
netsum += feature2[1] * .4711423;
netsum += feature2[2] * -1.461678;
netsum += feature2[3] * -.1164255;
netsum += feature2[4] * 5.208082E-02;
netsum += feature2[5] * .4290704;
netsum += feature2[6] * 5.005599E-02;
netsum += feature2[7] * -.7152472;
netsum += feature2[8] * -.4304705;
outarray[4] = 1 / (1 + exp(-netsum));

netsum = -.1631336;
netsum += feature2[0] * -9.112045E-03;
netsum += feature2[1] * .1418273;
netsum += feature2[2] * .3139661;
netsum += feature2[3] * .4594962;
netsum += feature2[4] * -.5077903;
netsum += feature2[5] * -.7215487;
netsum += feature2[6] * -.6620205;
netsum += feature2[7] * .1405424;
netsum += feature2[8] * 1.835448;
outarray[5] = 1 / (1 + exp(-netsum));

netsum = .1286572;
netsum += feature2[0] * -1.496963;
```

```
netsum += feature2[1] *  .6675627;
netsum += feature2[2] *  3.467289E-02;
netsum += feature2[3] * -.211388;
netsum += feature2[4] *  .4022121;
netsum += feature2[5] *  9.279378E-02;
netsum += feature2[6] *  .221469;
netsum += feature2[7] * -1.08096;
netsum += feature2[8] *  .5696868;
outarray[6] = 1 / (1 + exp(-netsum));

netsum = -.307033;
netsum += feature2[0] * -.2450359;
netsum += feature2[1] * -1.057701;
netsum += feature2[2] *  .643006;
netsum += feature2[3] *  1.318899;
netsum += feature2[4] * -.593664;
netsum += feature2[5] * -.3161325;
netsum += feature2[6] *  .6486963;
netsum += feature2[7] * -.2953574;
netsum += feature2[8] *  .2141867;
outarray[7] = 1 / (1 + exp(-netsum));

netsum = -2.572132E-03;
netsum += feature2[0] *  1.719462;
netsum += feature2[1] * -.3831635;
netsum += feature2[2] * -.6865217;
netsum += feature2[3] * -1.433495;
netsum += feature2[4] * -.1088401;
netsum += feature2[5] *  1.501535E-02;
netsum += feature2[6] * -.8874764;
netsum += feature2[7] *  1.299559;
netsum += feature2[8] * -.7590157;
outarray[8] = 1 / (1 + exp(-netsum));

netsum = -.3838195;
netsum += feature2[0] * -.1236574;
netsum += feature2[1] * -.9973538;
netsum += feature2[2] *  .7501443;
netsum += feature2[3] *  1.06568;
netsum += feature2[4] * -.3194561;
netsum += feature2[5] *  .4040247;
netsum += feature2[6] * -1.106399;
netsum += feature2[7] * -.2665701;
netsum += feature2[8] *  .2000748;
outarray[9] = 1 / (1 + exp(-netsum));

netsum =  .253037;
netsum += feature2[0] *  .0630246;
netsum += feature2[1] *  .3430368;
netsum += feature2[2] * -.9319632;
netsum += feature2[3] * -.5230323;
netsum += feature2[4] *  .1765063;
netsum += feature2[5] * -.2045487;
netsum += feature2[6] * -1.250098;
netsum += feature2[7] *  4.259654E-02;
netsum += feature2[8] * -.1516763;
outarray[10] = 1 / (1 + exp(-netsum));

netsum = -.5481886;
netsum += feature2[0] *  .1021582;
netsum += feature2[1] *  .2943085;
netsum += feature2[2] *  .166375;
```

```
netsum += feature2[3] * -.4570191;
netsum += feature2[4] * -.25356;
netsum += feature2[5] * -.6339683;
netsum += feature2[6] * 2.116526;
netsum += feature2[7] * .342081;
netsum += feature2[8] * -.1411358;
outarray[11] = 1 / (1 + exp(-netsum));

netsum = -.1036133;
netsum += feature2[0] * 1.514996;
netsum += feature2[1] * -.5860161;
netsum += feature2[2] * -.473146;
netsum += feature2[3] * -.5940139;
netsum += feature2[4] * .2028816;
netsum += feature2[5] * -.1521696;
netsum += feature2[6] * .1656087;
netsum += feature2[7] * -.2721972;
netsum += feature2[8] * .3705154;
outarray[12] = 1 / (1 + exp(-netsum));

netsum = -.2633717;
netsum += feature2[0] * .9698912;
netsum += feature2[1] * -.680374;
netsum += feature2[2] * .3501442;
netsum += feature2[3] * -.4401273;
netsum += feature2[4] * -.3729998;
netsum += feature2[5] * -.5060975;
netsum += feature2[6] * .1538693;
netsum += feature2[7] * -.320682;
netsum += feature2[8] * .2357144;
outarray[13] = 1 / (1 + exp(-netsum));

netsum = -.1887729;
netsum += feature2[0] * -2.338608;
netsum += feature2[1] * .5265903;
netsum += feature2[2] * .0815384;
netsum += feature2[3] * .7473056;
netsum += feature2[4] * -.1151378;
netsum += feature2[5] * .6119967;
netsum += feature2[6] * -.3616328;
netsum += feature2[7] * .5329512;
netsum += feature2[8] * -.6226013;
outarray[14] = 1 / (1 + exp(-netsum));


outarray[0] = (outarray[0] - .1) / .8 ;
if (outarray[0]< 0) outarray[0] = 0;
if (outarray[0]> 1) outarray[0] = 1;

outarray[1] = (outarray[1] - .1) / .8 ;
if (outarray[1]< 0) outarray[1] = 0;
if (outarray[1]> 1) outarray[1] = 1;

outarray[2] = (outarray[2] - .1) / .8 ;
if (outarray[2]< 0) outarray[2] = 0;
if (outarray[2]> 1) outarray[2] = 1;

outarray[3] = (outarray[3] - .1) / .8 ;
if (outarray[3]< 0) outarray[3] = 0;
if (outarray[3]> 1) outarray[3] = 1;

outarray[4] = (outarray[4] - .1) / .8 ;
```

```
if (outarray[4]< 0) outarray[4] = 0;
if (outarray[4]> 1) outarray[4] = 1;

outarray[5] = (outarray[5] - .1) / .8 ;
if (outarray[5]< 0) outarray[5] = 0;
if (outarray[5]> 1) outarray[5] = 1;

outarray[6] = (outarray[6] - .1) / .8 ;
if (outarray[6]< 0) outarray[6] = 0;
if (outarray[6]> 1) outarray[6] = 1;

outarray[7] = (outarray[7] - .1) / .8 ;
if (outarray[7]< 0) outarray[7] = 0;
if (outarray[7]> 1) outarray[7] = 1;

outarray[8] = (outarray[8] - .1) / .8 ;
if (outarray[8]< 0) outarray[8] = 0;
if (outarray[8]> 1) outarray[8] = 1;

outarray[9] = (outarray[9] - .1) / .8 ;
if (outarray[9]< 0) outarray[9] = 0;
if (outarray[9]> 1) outarray[9] = 1;

outarray[10] = (outarray[10] - .1) / .8 ;
if (outarray[10]< 0) outarray[10] = 0;
if (outarray[10]> 1) outarray[10] = 1;

outarray[11] = (outarray[11] - .1) / .8 ;
if (outarray[11]< 0) outarray[11] = 0;
if (outarray[11]> 1) outarray[11] = 1;

outarray[12] = (outarray[12] - .1) / .8 ;
if (outarray[12]< 0) outarray[12] = 0;
if (outarray[12]> 1) outarray[12] = 1;

outarray[13] = (outarray[13] - .1) / .8 ;
if (outarray[13]< 0) outarray[13] = 0;
if (outarray[13]> 1) outarray[13] = 1;

outarray[14] = (outarray[14] - .1) / .8 ;
if (outarray[14]< 0) outarray[14] = 0;
if (outarray[14]> 1) outarray[14] = 1;


}
```

Appendix D2: Neural Weights for Mean Flowtime network

```
/* Insert this code into your C program to fire the C:\NSHELL2\SPT&PTWINQ&LWKR network */
/* This code is designed to be simple and fast for porting to any machine */
/* Therefore all code and weights are inline without looping or data storage */
/*   which might be harder to port between compilers. */

#include <math.h>

void neuro_dispatch(double *inarray, double *outarray)
{
 double netsum;
 double feature2[20];

/* inarray[0] is B */
/* inarray[1] is C */
/* inarray[2] is D */
/* inarray[3] is E */
/* inarray[4] is F */
/* inarray[5] is G */
/* inarray[6] is H */
/* inarray[7] is I */
/* inarray[8] is J */
/* inarray[9] is K */
/* inarray[10] is L */
/* inarray[11] is M */
/* inarray[12] is N */
/* inarray[13] is O */
/* inarray[14] is P */
/* outarray[0] is Q */
/* outarray[1] is R */
/* outarray[2] is S */
/* outarray[3] is T */
/* outarray[4] is U */
/* outarray[5] is V */
/* outarray[6] is W */
/* outarray[7] is X */
/* outarray[8] is Y */
/* outarray[9] is Z */
/* outarray[10] is AA */
/* outarray[11] is AB */
/* outarray[12] is AC */
/* outarray[13] is AD */
/* outarray[14] is AE */

 if (inarray[0]< .33) inarray[0] =  .33;
 if (inarray[0]> 1) inarray[0] =  1;
 inarray[0] = (inarray[0] -  .33) / .67;

 if (inarray[1]< .31) inarray[1] =  .31;
 if (inarray[1]> 1) inarray[1] =  1;
 inarray[1] = (inarray[1] -  .31) / .69;

 if (inarray[2]< .32) inarray[2] =  .32;
 if (inarray[2]> 1) inarray[2] =  1;
 inarray[2] = (inarray[2] -  .32) / .68;

 if (inarray[3]< .31) inarray[3] =  .31;
 if (inarray[3]> 1) inarray[3] =  1;
 inarray[3] = (inarray[3] -  .31) / .69;

 if (inarray[4]< .34) inarray[4] =  .34;
```

```
if (inarray[4]> 1) inarray[4] =  1;
inarray[4] = (inarray[4] -  .34) / .66;

if (inarray[5]< .49) inarray[5] =  .49;
if (inarray[5]> 1) inarray[5] =  1;
inarray[5] = (inarray[5] -  .49) / .51;

if (inarray[6]< .51) inarray[6] =  .51;
if (inarray[6]> 1) inarray[6] =  1;
inarray[6] = (inarray[6] -  .51) / .49;

if (inarray[7]< .53) inarray[7] =  .53;
if (inarray[7]> 1) inarray[7] =  1;
inarray[7] = (inarray[7] -  .53) / .47;

if (inarray[8]< .55) inarray[8] =  .55;
if (inarray[8]> 1) inarray[8] =  1;
inarray[8] = (inarray[8] -  .55) / .45;

if (inarray[9]< .57) inarray[9] =  .57;
if (inarray[9]> 1) inarray[9] =  1;
inarray[9] = (inarray[9] -  .57) / .43;

if (inarray[10]< .22) inarray[10] =  .22;
if (inarray[10]> .93) inarray[10] =  .93;
inarray[10] = (inarray[10] -  .22) / .71;

if (inarray[11]< .25) inarray[11] =  .25;
if (inarray[11]> .92) inarray[11] =  .92;
inarray[11] = (inarray[11] -  .25) / .67;

if (inarray[12]< .22) inarray[12] =  .22;
if (inarray[12]> .96) inarray[12] =  .96;
inarray[12] = (inarray[12] -  .22) / .74;

if (inarray[13]< .22) inarray[13] =  .22;
if (inarray[13]> .95) inarray[13] =  .95;
inarray[13] = (inarray[13] -  .22) / .73;

if (inarray[14]< .23) inarray[14] =  .23;
if (inarray[14]> .96) inarray[14] =  .96;
inarray[14] = (inarray[14] -  .23) / .73;

netsum = -2.127025;
netsum += inarray[0] * -1.534661;
netsum += inarray[1] *  1.385606;
netsum += inarray[2] * -1.67123;
netsum += inarray[3] * -.2299798;
netsum += inarray[4] *  5.66486;
netsum += inarray[5] * -.4372981;
netsum += inarray[6] * -.5978695;
netsum += inarray[7] * -.5532722;
netsum += inarray[8] * -.4884661;
netsum += inarray[9] * -.6174343;
netsum += inarray[10] * -.3179735;
netsum += inarray[11] * -.7406918;
netsum += inarray[12] * -.3327855;
netsum += inarray[13] * -.5212398;
netsum += inarray[14] * -1.144423;
feature2[0] = 1 / (1 + exp(-netsum));

netsum =  .1270918;
```

```
netsum += inarray[0] *  5.635597E-02;
netsum += inarray[1] *  2.101648;
netsum += inarray[2] * -.3158018;
netsum += inarray[3] * -.2156307;
netsum += inarray[4] *  .2873424;
netsum += inarray[5] * -7.687316E-02;
netsum += inarray[6] * -8.572415E-02;
netsum += inarray[7] * -.2063143;
netsum += inarray[8] * -.2226138;
netsum += inarray[9] * -.2139049;
netsum += inarray[10] * -.2817429;
netsum += inarray[11] *  .1300127;
netsum += inarray[12] *  4.634594E-02;
netsum += inarray[13] * -8.277769E-02;
netsum += inarray[14] *  .2510398;
feature2[1] = 1 / (1 + exp(-netsum));

netsum = -.3106934;
netsum += inarray[0] * -.3552261;
netsum += inarray[1] *  .1258201;
netsum += inarray[2] *  .2594771;
netsum += inarray[3] *  .1737312;
netsum += inarray[4] * -7.878974E-02;
netsum += inarray[5] * -.1456324;
netsum += inarray[6] *  .1960943;
netsum += inarray[7] *  .2037766;
netsum += inarray[8] * -.2186606;
netsum += inarray[9] *  5.458169E-02;
netsum += inarray[10] *  .2276493;
netsum += inarray[11] *  .1183151;
netsum += inarray[12] *  .1211713;
netsum += inarray[13] * -5.565529E-02;
netsum += inarray[14] * -.2561574;
feature2[2] = 1 / (1 + exp(-netsum));

netsum = -1.454346;
netsum += inarray[0] * -1.93679;
netsum += inarray[1] * -3.000002;
netsum += inarray[2] * -1.563219;
netsum += inarray[3] * -1.450586;
netsum += inarray[4] *  6.923056;
netsum += inarray[5] * -7.124248E-02;
netsum += inarray[6] * -.2007552;
netsum += inarray[7] *  7.105684E-02;
netsum += inarray[8] * -.3891807;
netsum += inarray[9] * -9.084614E-02;
netsum += inarray[10] * -.2354679;
netsum += inarray[11] *  .1472126;
netsum += inarray[12] * -.3897824;
netsum += inarray[13] * -.4571439;
netsum += inarray[14] * -.9498448;
feature2[3] = 1 / (1 + exp(-netsum));

netsum =  .4064647;
netsum += inarray[0] *  6.080595E-02;
netsum += inarray[1] *  .5041848;
netsum += inarray[2] * -8.277338E-02;
netsum += inarray[3] * -4.359403;
netsum += inarray[4] *  .2567576;
netsum += inarray[5] * -.1333682;
netsum += inarray[6] * -.1788362;
netsum += inarray[7] * -6.317507E-03;
```

```
netsum += inarray[8] *  3.181771E-02;
netsum += inarray[9] * -2.644823E-03;
netsum += inarray[10] * -.4072654;
netsum += inarray[11] *  .2850039;
netsum += inarray[12] * -1.406715E-02;
netsum += inarray[13] *  1.495292;
netsum += inarray[14] * -6.633072E-03;
feature2[4] = 1 / (1 + exp(-netsum));

netsum =  3.044856;
netsum += inarray[0] *  2.98366;
netsum += inarray[1] *  9.195838E-02;
netsum += inarray[2] * -.8917376;
netsum += inarray[3] * -.5956516;
netsum += inarray[4] *  .4019775;
netsum += inarray[5] * -.2431124;
netsum += inarray[6] * -.4644013;
netsum += inarray[7] * -.4797978;
netsum += inarray[8] *  5.842389E-02;
netsum += inarray[9] * -.5646109;
netsum += inarray[10] * -.4507366;
netsum += inarray[11] * -.3701822;
netsum += inarray[12] * -4.062822E-02;
netsum += inarray[13] * -.3810509;
netsum += inarray[14] * -.4784395;
feature2[5] = 1 / (1 + exp(-netsum));

netsum =  4.163668;
netsum += inarray[0] *  4.272491;
netsum += inarray[1] * -2.513001;
netsum += inarray[2] * -2.495854;
netsum += inarray[3] * -1.830051;
netsum += inarray[4] * -1.775271;
netsum += inarray[5] * -.3403684;
netsum += inarray[6] * -.1819456;
netsum += inarray[7] * -.2830323;
netsum += inarray[8] * -.4010035;
netsum += inarray[9] *  .247472;
netsum += inarray[10] * -3.256461;
netsum += inarray[11] * -1.079243;
netsum += inarray[12] * -.4956464;
netsum += inarray[13] * -.6977581;
netsum += inarray[14] * -.744369;
feature2[6] = 1 / (1 + exp(-netsum));

netsum = -1.064907;
netsum += inarray[0] * -1.259773;
netsum += inarray[1] * -1.361285;
netsum += inarray[2] * -1.633782;
netsum += inarray[3] *  4.440379;
netsum += inarray[4] * -1.530057;
netsum += inarray[5] *  .023477;
netsum += inarray[6] * -.5001901;
netsum += inarray[7] * -.4571561;
netsum += inarray[8] * -6.483155E-02;
netsum += inarray[9] *  .3028651;
netsum += inarray[10] * -.4103746;
netsum += inarray[11] *  .1935083;
netsum += inarray[12] *  .2063555;
netsum += inarray[13] * -.3256872;
netsum += inarray[14] * -.2339511;
feature2[7] = 1 / (1 + exp(-netsum));
```

```
netsum =  .1447327;
netsum += inarray[0] *  1.208711E-02;
netsum += inarray[1] * -1.8431;
netsum += inarray[2] *  9.931198;
netsum += inarray[3] *  .7859221;
netsum += inarray[4] * -.7343228;
netsum += inarray[5] * -.5437335;
netsum += inarray[6] * -.2232141;
netsum += inarray[7] * -.4139314;
netsum += inarray[8] * -3.65996;
netsum += inarray[9] *  1.675354;
netsum += inarray[10] * -.2901072;
netsum += inarray[11] * -.326509;
netsum += inarray[12] * -.463695;
netsum += inarray[13] * -9.982181E-02;
netsum += inarray[14] * -.2578963;
feature2[8] = 1 / (1 + exp(-netsum));

netsum =  .6833475;
netsum += inarray[0] *  .5996841;
netsum += inarray[1] * -8.933963E-02;
netsum += inarray[2] * -.1969835;
netsum += inarray[3] *  .2354695;
netsum += inarray[4] *  2.379485;
netsum += inarray[5] * -.6763925;
netsum += inarray[6] * -.4754021;
netsum += inarray[7] *  6.257617E-02;
netsum += inarray[8] *  .5929466;
netsum += inarray[9] *  7.118178E-02;
netsum += inarray[10] * -.4555093;
netsum += inarray[11] * -.3480999;
netsum += inarray[12] * -.2399379;
netsum += inarray[13] * -4.061316E-02;
netsum += inarray[14] * -5.547979E-02;
feature2[9] = 1 / (1 + exp(-netsum));

netsum =  2.05816;
netsum += inarray[0] *  2.053436;
netsum += inarray[1] * -.544432;
netsum += inarray[2] *  .4541343;
netsum += inarray[3] * -.1915346;
netsum += inarray[4] * -.3568715;
netsum += inarray[5] * -.2190787;
netsum += inarray[6] * -.5233611;
netsum += inarray[7] * -.274958;
netsum += inarray[8] * -.3953035;
netsum += inarray[9] * -.3348909;
netsum += inarray[10] * -.4108803;
netsum += inarray[11] * -.1989731;
netsum += inarray[12] * -.377194;
netsum += inarray[13] *  3.258267E-02;
netsum += inarray[14] *  8.858351E-02;
feature2[10] = 1 / (1 + exp(-netsum));

netsum = -.4608839;
netsum += inarray[0] * -.5784227;
netsum += inarray[1] * -.4134966;
netsum += inarray[2] * -.2498437;
netsum += inarray[3] *  7.369254E-02;
netsum += inarray[4] *  .4111851;
netsum += inarray[5] * -.645071;
```

```
netsum += inarray[6] * -.2638268;
netsum += inarray[7] * -8.220226E-02;
netsum += inarray[8] * .2338365;
netsum += inarray[9] * -.4312649;
netsum += inarray[10] * 1.318989;
netsum += inarray[11] * -.6365233;
netsum += inarray[12] * .390272;
netsum += inarray[13] * 1.793994;
netsum += inarray[14] * -.3162842;
feature2[11] = 1 / (1 + exp(-netsum));

netsum = 2.085266;
netsum += inarray[0] * 1.929439;
netsum += inarray[1] * .4223873;
netsum += inarray[2] * 9.113403E-02;
netsum += inarray[3] * 4.301021E-02;
netsum += inarray[4] * -5.955182;
netsum += inarray[5] * -1.898142E-02;
netsum += inarray[6] * .2580843;
netsum += inarray[7] * -.1405491;
netsum += inarray[8] * .198081;
netsum += inarray[9] * -6.756395E-02;
netsum += inarray[10] * -2.512729;
netsum += inarray[11] * -1.975562;
netsum += inarray[12] * .2905951;
netsum += inarray[13] * .3234188;
netsum += inarray[14] * 1.691879;
feature2[12] = 1 / (1 + exp(-netsum));

netsum = -.1084178;
netsum += inarray[0] * -.2052653;
netsum += inarray[1] * .1929678;
netsum += inarray[2] * .2697195;
netsum += inarray[3] * .7687581;
netsum += inarray[4] * -8.798962E-02;
netsum += inarray[5] * -.038018;
netsum += inarray[6] * -.1868947;
netsum += inarray[7] * -.2239236;
netsum += inarray[8] * -8.798546E-03;
netsum += inarray[9] * .1866033;
netsum += inarray[10] * -4.679623E-02;
netsum += inarray[11] * -5.749673E-02;
netsum += inarray[12] * 1.084791E-02;
netsum += inarray[13] * 2.286516E-02;
netsum += inarray[14] * -.3621812;
feature2[13] = 1 / (1 + exp(-netsum));

netsum = -1.368212;
netsum += inarray[0] * -1.516386;
netsum += inarray[1] * -1.312811;
netsum += inarray[2] * 3.834776;
netsum += inarray[3] * -1.949248;
netsum += inarray[4] * -1.077547;
netsum += inarray[5] * -9.715545E-02;
netsum += inarray[6] * -2.381796E-02;
netsum += inarray[7] * -.239951;
netsum += inarray[8] * -9.391313E-03;
netsum += inarray[9] * -5.044011E-02;
netsum += inarray[10] * .190827;
netsum += inarray[11] * 4.736579E-02;
netsum += inarray[12] * -.2320667;
netsum += inarray[13] * -.1441375;
```

```
netsum += inarray[14] * -2.852006E-02;
feature2[14] = 1 / (1 + exp(-netsum));

netsum = -.6073152;
netsum += inarray[0] * -.2516953;
netsum += inarray[1] * -.4811573;
netsum += inarray[2] *  .242735;
netsum += inarray[3] *  1.989833;
netsum += inarray[4] * -.609577;
netsum += inarray[5] * -.4903045;
netsum += inarray[6] *  .3586577;
netsum += inarray[7] * -.0742056;
netsum += inarray[8] * -.1698011;
netsum += inarray[9] *  5.020628E-02;
netsum += inarray[10] *  .2822423;
netsum += inarray[11] * -9.924112E-02;
netsum += inarray[12] * -.3541629;
netsum += inarray[13] * -.1074304;
netsum += inarray[14] * -.2848581;
feature2[15] = 1 / (1 + exp(-netsum));

netsum = -.19364;
netsum += inarray[0] *  .2215094;
netsum += inarray[1] *  2.662252E-02;
netsum += inarray[2] *  4.830183;
netsum += inarray[3] * -.2915323;
netsum += inarray[4] * -1.967856;
netsum += inarray[5] *  .2577111;
netsum += inarray[6] * -.1678625;
netsum += inarray[7] *  .1838287;
netsum += inarray[8] *  .04581;
netsum += inarray[9] * -.1416859;
netsum += inarray[10] *  .1741706;
netsum += inarray[11] *  .3041297;
netsum += inarray[12] * -2.209;
netsum += inarray[13] * -.3314497;
netsum += inarray[14] *  .1475125;
feature2[16] = 1 / (1 + exp(-netsum));

netsum = -2.626755;
netsum += inarray[0] * -2.363863;
netsum += inarray[1] *  1.01529;
netsum += inarray[2] * -.213787;
netsum += inarray[3] * -.2568656;
netsum += inarray[4] * -.1444943;
netsum += inarray[5] *  .1128776;
netsum += inarray[6] *  .1793339;
netsum += inarray[7] * -1.735728E-02;
netsum += inarray[8] * -.3037648;
netsum += inarray[9] * -5.242693E-02;
netsum += inarray[10] *  .3987421;
netsum += inarray[11] *  .4650992;
netsum += inarray[12] * -.34305;
netsum += inarray[13] * -.3603955;
netsum += inarray[14] *  6.268135E-02;
feature2[17] = 1 / (1 + exp(-netsum));

netsum =  .5671079;
netsum += inarray[0] *  .7906501;
netsum += inarray[1] * -2.482796;
netsum += inarray[2] *  .2383201;
netsum += inarray[3] *  1.422667;
```

```
netsum += inarray[4] *  1.250559;
netsum += inarray[5] * -.2164256;
netsum += inarray[6] * -.2014819;
netsum += inarray[7] *  .3662656;
netsum += inarray[8] *  .1328816;
netsum += inarray[9] *  3.565036E-02;
netsum += inarray[10] * -.2044796;
netsum += inarray[11] * -.1553417;
netsum += inarray[12] *  .6052962;
netsum += inarray[13] *  .1387127;
netsum += inarray[14] * -.1252561;
feature2[18] = 1 / (1 + exp(-netsum));

netsum = -.3439108;
netsum += inarray[0] * -.2622799;
netsum += inarray[1] *  .6619859;
netsum += inarray[2] * -.2700452;
netsum += inarray[3] *  5.999915E-02;
netsum += inarray[4] *  4.309079;
netsum += inarray[5] * -3.464223E-02;
netsum += inarray[6] * -.177265;
netsum += inarray[7] *  .2377198;
netsum += inarray[8] *  1.981383;
netsum += inarray[9] *  5.538443E-02;
netsum += inarray[10] *  2.151327E-02;
netsum += inarray[11] *  .1784591;
netsum += inarray[12] *  .2804021;
netsum += inarray[13] *  .1073797;
netsum += inarray[14] *  1.576134E-02;
feature2[19] = 1 / (1 + exp(-netsum));

netsum = -.4018244;
netsum += feature2[0] * -.6004633;
netsum += feature2[1] *  .1079664;
netsum += feature2[2] * -5.970221E-02;
netsum += feature2[3] * -.2646531;
netsum += feature2[4] * -.8192776;
netsum += feature2[5] * -.2910616;
netsum += feature2[6] *  3.688971;
netsum += feature2[7] * -2.120626;
netsum += feature2[8] *  .5011812;
netsum += feature2[9] * -.2401061;
netsum += feature2[10] * -.1371416;
netsum += feature2[11] *  .3884444;
netsum += feature2[12] * -3.652697E-02;
netsum += feature2[13] * -.4143871;
netsum += feature2[14] * -1.290449;
netsum += feature2[15] *  .1108165;
netsum += feature2[16] * -.3153532;
netsum += feature2[17] *  9.063054;
netsum += feature2[18] *  .2038849;
netsum += feature2[19] * -.3027286;
outarray[0] = 1 / (1 + exp(-netsum));

netsum =  .3021354;
netsum += feature2[0] *  .5562754;
netsum += feature2[1] * -3.721453E-02;
netsum += feature2[2] *  .1274372;
netsum += feature2[3] * -.2567498;
netsum += feature2[4] * -.7406126;
netsum += feature2[5] * -.5004629;
netsum += feature2[6] * -1.118358;
```

```
netsum += feature2[7] * -.420098;
netsum += feature2[8] *  .3554352;
netsum += feature2[9] * -.2985365;
netsum += feature2[10] * -.5600451;
netsum += feature2[11] *  .1336511;
netsum += feature2[12] * -.3710997;
netsum += feature2[13] * -7.074618E-02;
netsum += feature2[14] *  .834084;
netsum += feature2[15] *  .5622373;
netsum += feature2[16] * -.2662526;
netsum += feature2[17] *  5.237437;
netsum += feature2[18] * -.3949004;
netsum += feature2[19] * -.3496082;
outarray[1] = 1 / (1 + exp(-netsum));

netsum = -2.538008E-02;
netsum += feature2[0] * -1.063961E-02;
netsum += feature2[1] *  .1183521;
netsum += feature2[2] *  .1714737;
netsum += feature2[3] *  .3209237;
netsum += feature2[4] *  1.463832;
netsum += feature2[5] *  .2554799;
netsum += feature2[6] * -3.548389;
netsum += feature2[7] *  1.80822;
netsum += feature2[8] * -.590492;
netsum += feature2[9] *  .289912;
netsum += feature2[10] *  .9830801;
netsum += feature2[11] * -.5915321;
netsum += feature2[12] *  .3082871;
netsum += feature2[13] *  4.501532E-02;
netsum += feature2[14] *  .7100854;
netsum += feature2[15] *  3.106604E-02;
netsum += feature2[16] * -.1417924;
netsum += feature2[17] * -16.6156;
netsum += feature2[18] * -7.406253E-02;
netsum += feature2[19] * -.1195654;
outarray[2] = 1 / (1 + exp(-netsum));

netsum = -4.347908;
netsum += feature2[0] * -4.47907;
netsum += feature2[1] *  1.432947;
netsum += feature2[2] *  .571112;
netsum += feature2[3] *  .6055597;
netsum += feature2[4] *  .3752851;
netsum += feature2[5] * -.1683265;
netsum += feature2[6] * -.7025571;
netsum += feature2[7] * -1.964335;
netsum += feature2[8] * -.3396918;
netsum += feature2[9] *  6.758477E-03;
netsum += feature2[10] * -.3069955;
netsum += feature2[11] *  1.882676;
netsum += feature2[12] *  .1543772;
netsum += feature2[13] * -.3090751;
netsum += feature2[14] * -2.520872;
netsum += feature2[15] * -1.261727E-02;
netsum += feature2[16] *  3.661972E-03;
netsum += feature2[17] *  11.65454;
netsum += feature2[18] * -1.169391;
netsum += feature2[19] *  3.663805;
outarray[3] = 1 / (1 + exp(-netsum));

netsum =  .770401;
```

```
netsum += feature2[0] *  .991416;
netsum += feature2[1] * -1.889331;
netsum += feature2[2] * -.139139;
netsum += feature2[3] *  .4393654;
netsum += feature2[4] * -.5469683;
netsum += feature2[5] * -.3776391;
netsum += feature2[6] * -.1153614;
netsum += feature2[7] *  .45415;
netsum += feature2[8] *  5.244106E-02;
netsum += feature2[9] * -.2061416;
netsum += feature2[10] *  .1450569;
netsum += feature2[11] * -1.786922;
netsum += feature2[12] * -.4290357;
netsum += feature2[13] * -.3400771;
netsum += feature2[14] *  .3615972;
netsum += feature2[15] *  3.215213E-02;
netsum += feature2[16] *  .2886165;
netsum += feature2[17] * -8.394811;
netsum += feature2[18] *  .9913402;
netsum += feature2[19] * -.3295543;
outarray[4] = 1 / (1 + exp(-netsum));

netsum =  3.686605;
netsum += feature2[0] *  3.340854;
netsum += feature2[1] * -.3266241;
netsum += feature2[2] * -.383816;
netsum += feature2[3] * -.9007248;
netsum += feature2[4] *  2.365861E-02;
netsum += feature2[5] * -1.179796;
netsum += feature2[6] *  3.917684E-02;
netsum += feature2[7] *  1.091974;
netsum += feature2[8] *  .1532727;
netsum += feature2[9] *  .2134319;
netsum += feature2[10] *  .1552497;
netsum += feature2[11] * -.130137;
netsum += feature2[12] *  .1949993;
netsum += feature2[13] * -.2376275;
netsum += feature2[14] *  1.101267;
netsum += feature2[15] * -.0493302;
netsum += feature2[16] * -.2057998;
netsum += feature2[17] * -12.09187;
netsum += feature2[18] * -9.530596E-02;
netsum += feature2[19] * -2.167364;
outarray[5] = 1 / (1 + exp(-netsum));

netsum =  .2261838;
netsum += feature2[0] *  .6117795;
netsum += feature2[1] *  .2556338;
netsum += feature2[2] *  .1104424;
netsum += feature2[3] * -1.76116;
netsum += feature2[4] * -.8814707;
netsum += feature2[5] * -.1399774;
netsum += feature2[6] * -.1515178;
netsum += feature2[7] * -.2513995;
netsum += feature2[8] * -.2032537;
netsum += feature2[9] * -.2282047;
netsum += feature2[10] *  .1178242;
netsum += feature2[11] *  9.989554E-02;
netsum += feature2[12] * -.1187363;
netsum += feature2[13] * -.1588543;
netsum += feature2[14] *  8.299998;
netsum += feature2[15] * -.4917489;
```

```
netsum += feature2[16] * -1.272974;
netsum += feature2[17] * -4.948971;
netsum += feature2[18] *  .2373703;
netsum += feature2[19] * -.1382861;
outarray[6] = 1 / (1 + exp(-netsum));

netsum =  .7143841;
netsum += feature2[0] *  .9115762;
netsum += feature2[1] * -7.350495E-02;
netsum += feature2[2] * -.4156601;
netsum += feature2[3] * -.7382054;
netsum += feature2[4] * -.5534458;
netsum += feature2[5] *  .1255922;
netsum += feature2[6] *  .4917387;
netsum += feature2[7] *  .1844895;
netsum += feature2[8] * -.5002103;
netsum += feature2[9] * -4.216522E-02;
netsum += feature2[10] * -.4226763;
netsum += feature2[11] *  .1202922;
netsum += feature2[12] * -.1485279;
netsum += feature2[13] * -.1823302;
netsum += feature2[14] * -1.62964;
netsum += feature2[15] * -.2875343;
netsum += feature2[16] * -.3986559;
netsum += feature2[17] * -3.04541;
netsum += feature2[18] * -1.760078E-02;
netsum += feature2[19] * -.4441942;
outarray[7] = 1 / (1 + exp(-netsum));

netsum = -1.106611;
netsum += feature2[0] * -1.257366;
netsum += feature2[1] * -5.262233E-02;
netsum += feature2[2] *  .1283339;
netsum += feature2[3] *  1.897849;
netsum += feature2[4] *  1.263269;
netsum += feature2[5] * -.0826937;
netsum += feature2[6] * -4.663133E-02;
netsum += feature2[7] *  .1077188;
netsum += feature2[8] *  .9141859;
netsum += feature2[9] * -1.942981E-02;
netsum += feature2[10] * -.0170227;
netsum += feature2[11] * -.5775023;
netsum += feature2[12] *  6.083321E-02;
netsum += feature2[13] *  .44481;
netsum += feature2[14] * -7.26222;
netsum += feature2[15] *  .454815;
netsum += feature2[16] *  1.172065;
netsum += feature2[17] *  6.179174;
netsum += feature2[18] * -.2194533;
netsum += feature2[19] *  2.928195E-03;
outarray[8] = 1 / (1 + exp(-netsum));

netsum = -.557344;
netsum += feature2[0] * -.8355444;
netsum += feature2[1] * -.7298361;
netsum += feature2[2] * -9.908194E-02;
netsum += feature2[3] * -1.338743;
netsum += feature2[4] *  2.726421;
netsum += feature2[5] * -.3369063;
netsum += feature2[6] * -1.602735;
netsum += feature2[7] *  5.736315;
netsum += feature2[8] *  .4491995;
```

```
netsum += feature2[9] * -.26771;
netsum += feature2[10] * -.3346169;
netsum += feature2[11] * .1424053;
netsum += feature2[12] * -.2673492;
netsum += feature2[13] * 9.440368E-02;
netsum += feature2[14] * -1.682431;
netsum += feature2[15] * -6.719103E-02;
netsum += feature2[16] * -.2724702;
netsum += feature2[17] * -8.343706;
netsum += feature2[18] * .4701097;
netsum += feature2[19] * -4.028329E-02;
outarray[9] = 1 / (1 + exp(-netsum));

netsum = .7294881;
netsum += feature2[0] * .478992;
netsum += feature2[1] * -.1699279;
netsum += feature2[2] * -.2918522;
netsum += feature2[3] * -.1400667;
netsum += feature2[4] * .3066254;
netsum += feature2[5] * .2507746;
netsum += feature2[6] * -.719865;
netsum += feature2[7] * -1.083979;
netsum += feature2[8] * -.3014532;
netsum += feature2[9] * -.300752;
netsum += feature2[10] * 2.493552E-02;
netsum += feature2[11] * -5.258249E-02;
netsum += feature2[12] * 7.411258E-02;
netsum += feature2[13] * -.1965154;
netsum += feature2[14] * .4011763;
netsum += feature2[15] * -1.669259;
netsum += feature2[16] * -3.424472E-02;
netsum += feature2[17] * -7.968295;
netsum += feature2[18] * -.602005;
netsum += feature2[19] * -6.404935E-02;
outarray[10] = 1 / (1 + exp(-netsum));

netsum = .362788;
netsum += feature2[0] * .0981926;
netsum += feature2[1] * 2.843576E-02;
netsum += feature2[2] * -.1703628;
netsum += feature2[3] * .8380809;
netsum += feature2[4] * -2.949265;
netsum += feature2[5] * .2401737;
netsum += feature2[6] * 1.392454;
netsum += feature2[7] * -5.736626;
netsum += feature2[8] * -.3045191;
netsum += feature2[9] * 7.248948E-02;
netsum += feature2[10] * -.2216617;
netsum += feature2[11] * 5.055316E-03;
netsum += feature2[12] * .1508938;
netsum += feature2[13] * .1570011;
netsum += feature2[14] * .8849996;
netsum += feature2[15] * 1.165793;
netsum += feature2[16] * .1073198;
netsum += feature2[17] * 10.06857;
netsum += feature2[18] * 2.068069E-02;
netsum += feature2[19] * .2028647;
outarray[11] = 1 / (1 + exp(-netsum));

netsum = .8200681;
netsum += feature2[0] * .9755294;
netsum += feature2[1] * -.2646338;
```

```
netsum += feature2[2] * -.3214362;
netsum += feature2[3] *  3.809073;
netsum += feature2[4] * -.6226831;
netsum += feature2[5] * -8.820433E-02;
netsum += feature2[6] * -1.496528;
netsum += feature2[7] * -1.773552;
netsum += feature2[8] * -9.658501E-03;
netsum += feature2[9] * -.6434142;
netsum += feature2[10] * -.3066177;
netsum += feature2[11] * -1.086938E-02;
netsum += feature2[12] *  .9212445;
netsum += feature2[13] * -.5968158;
netsum += feature2[14] * -1.807871;
netsum += feature2[15] *  3.398166E-02;
netsum += feature2[16] *  .2398897;
netsum += feature2[17] * -5.194323;
netsum += feature2[18] * -.1639087;
netsum += feature2[19] * -.1660768;
outarray[12] = 1 / (1 + exp(-netsum));

netsum = -3.714114E-04;
netsum += feature2[0] * -.3926134;
netsum += feature2[1] * -.4836594;
netsum += feature2[2] *  2.011262E-02;
netsum += feature2[3] * -.6448963;
netsum += feature2[4] * -.2702846;
netsum += feature2[5] * -3.631411E-02;
netsum += feature2[6] * -.6800619;
netsum += feature2[7] * -.5383763;
netsum += feature2[8] * -9.367502E-03;
netsum += feature2[9] * -.4512094;
netsum += feature2[10] *  .7111861;
netsum += feature2[11] * -1.238454;
netsum += feature2[12] *  .7630368;
netsum += feature2[13] *  5.750604E-02;
netsum += feature2[14] * -.9760057;
netsum += feature2[15] *  .3790655;
netsum += feature2[16] * -5.644364E-02;
netsum += feature2[17] * -.5015465;
netsum += feature2[18] * -.4474064;
netsum += feature2[19] * -.3726536;
outarray[13] = 1 / (1 + exp(-netsum));

netsum = -1.247646;
netsum += feature2[0] * -1.071091;
netsum += feature2[1] * -2.893961E-02;
netsum += feature2[2] * -2.861611E-02;
netsum += feature2[3] * -3.618598;
netsum += feature2[4] *  .6013362;
netsum += feature2[5] *  .1153496;
netsum += feature2[6] *  1.41837;
netsum += feature2[7] *  1.492391;
netsum += feature2[8] * -.2125802;
netsum += feature2[9] *  1.506482;
netsum += feature2[10] *  .3234671;
netsum += feature2[11] *  .3145125;
netsum += feature2[12] * -1.559603;
netsum += feature2[13] *  7.990149E-02;
netsum += feature2[14] *  2.460018;
netsum += feature2[15] *  .2015774;
netsum += feature2[16] * -.3996591;
netsum += feature2[17] *  7.581014;
```

```
netsum += feature2[18] * .2085034;
netsum += feature2[19] * .2459512;
outarray[14] = 1 / (1 + exp(-netsum));


outarray[0] = (outarray[0] - .1) / .8 ;
if (outarray[0]< 0) outarray[0] = 0;
if (outarray[0]> 1) outarray[0] = 1;

outarray[1] = (outarray[1] - .1) / .8 ;
if (outarray[1]< 0) outarray[1] = 0;
if (outarray[1]> 1) outarray[1] = 1;

outarray[2] = (outarray[2] - .1) / .8 ;
if (outarray[2]< 0) outarray[2] = 0;
if (outarray[2]> 1) outarray[2] = 1;

outarray[3] = (outarray[3] - .1) / .8 ;
if (outarray[3]< 0) outarray[3] = 0;
if (outarray[3]> 1) outarray[3] = 1;

outarray[4] = (outarray[4] - .1) / .8 ;
if (outarray[4]< 0) outarray[4] = 0;
if (outarray[4]> 1) outarray[4] = 1;

outarray[5] = (outarray[5] - .1) / .8 ;
if (outarray[5]< 0) outarray[5] = 0;
if (outarray[5]> 1) outarray[5] = 1;

outarray[6] = (outarray[6] - .1) / .8 ;
if (outarray[6]< 0) outarray[6] = 0;
if (outarray[6]> 1) outarray[6] = 1;

outarray[7] = (outarray[7] - .1) / .8 ;
if (outarray[7]< 0) outarray[7] = 0;
if (outarray[7]> 1) outarray[7] = 1;

outarray[8] = (outarray[8] - .1) / .8 ;
if (outarray[8]< 0) outarray[8] = 0;
if (outarray[8]> 1) outarray[8] = 1;

outarray[9] = (outarray[9] - .1) / .8 ;
if (outarray[9]< 0) outarray[9] = 0;
if (outarray[9]> 1) outarray[9] = 1;

outarray[10] = (outarray[10] - .1) / .8 ;
if (outarray[10]< 0) outarray[10] = 0;
if (outarray[10]> 1) outarray[10] = 1;

outarray[11] = (outarray[11] - .1) / .8 ;
if (outarray[11]< 0) outarray[11] = 0;
if (outarray[11]> 1) outarray[11] = 1;
 outarray[12] = (outarray[12] - .1) / .8 ;
if (outarray[12]< 0) outarray[12] = 0;
if (outarray[12]> 1) outarray[12] = 1;
 outarray[13] = (outarray[13] - .1) / .8 ;
if (outarray[13]< 0) outarray[13] = 0;
if (outarray[13]> 1) outarray[13] = 1;
 outarray[14] = (outarray[14] - .1) / .8 ;
if (outarray[14]< 0) outarray[14] = 0;
if (outarray[14]> 1) outarray[14] = 1;
 }
```

Appendix E1: -Comparison of BPNN with makespans from optimal rule combinations for n=10 to100

n=10	Optimal	BPNN	n=10	Optimal	BPNN
N10M1	845	845	N10M26	701	712
N10M2	819	819	N10M27	872	890
N10M3	842	880	N10M28	604	661
N10M4	762	762	N10M29	793	793
N10M5	604	604	N10M30	747	865
N10M6	815	815	N10M31	834	838
N10M7	723	753	N10M32	722	781
N10M8	862	862	N10M33	547	567
N10M9	868	899	N10M34	847	922
N10M10	766	766	N10M35	734	774
N10M11	800	931	N10M36	786	825
N10M12	827	828	N10M37	822	825
N10M13	849	849	N10M38	785	795
N10M14	658	658	N10M39	763	811
N10M15	620	626	N10M40	540	573
N10M16	801	801	N10M41	694	733
N10M17	555	555	N10M42	743	770
N10M18	773	819	N10M43	731	736
N10M19	535	535	N10M44	690	690
N10M20	872	878	N10M45	722	722
N10M21	757	757	N10M46	923	958
N10M22	592	592	N10M47	777	820
N10M23	629	641	N10M48	716	737
N10M24	811	901	N10M49	843	862
N10M25	693	693	N10M50	723	731

n=15	Optimal	BPNN	n=15	Optimal	BPNN
N15M1	1221	1272	N15M26	1235	1235
N15M2	1151	1151	N15M27	1272	1463
N15M3	1127	1127	N15M28	1081	1111
N15M4	1014	1014	N15M29	1125	1125
N15M5	1184	1184	N15M30	1055	1113
N15M6	1238	1248	N15M31	1030	1051
N15M7	1216	1232	N15M32	1064	1102
N15M8	1205	1286	N15M33	1040	1040
N15M9	1103	1103	N15M34	1146	1146
N15M10	1078	1088	N15M35	1184	1190
N15M11	1175	1281	N15M36	1185	1219
N15M12	1124	1124	N15M37	1141	1166
N15M13	781	781	N15M38	1295	1295
N15M14	887	926	N15M39	1273	1285
N15M15	1141	1226	N15M40	1218	1218
N15M16	1083	1089	N15M41	1140	1140
N15M17	1293	1293	N15M42	957	957
N15M18	1218	1286	N15M43	1199	1280
N15M19	1089	1165	N15M44	1116	1170
N15M20	1169	1228	N15M45	1159	1207
N15M21	974	994	N15M46	979	979
N15M22	1067	1067	N15M47	1299	1300
N15M23	844	852	N15M48	876	918
N15M24	1093	1129	N15M49	1155	1181
N15M25	1042	1122	N15M50	1246	1246

n=20	Optimal	BPNN	n=20	Optimal	BPNN
N20M1	1532	1532	N20M26	1450	1450
N20M2	1426	1426	N20M27	1196	1196
N20M3	1167	1354	N20M28	1492	1492
N20M4	1566	1566	N20M29	1368	1370
N20M5	1476	1490	N20M30	1200	1307
N20M6	1710	1748	N20M31	1634	1641
N20M7	1772	1781	N20M32	1482	1482
N20M8	1468	1468	N20M33	1508	1543
N20M9	1455	1455	N20M34	1740	1827
N20M10	1639	1800	N20M35	1441	1441
N20M11	1137	1196	N20M36	1115	1167
N20M12	1643	1802	N20M37	1386	1386
N20M13	1476	1476	N20M38	1557	1557
N20M14	1194	1199	N20M39	1446	1448
N20M15	1351	1351	N20M40	1493	1493
N20M16	1617	1617	N20M41	1313	1313
N20M17	1435	1474	N20M42	1476	1476
N20M18	1462	1462	N20M43	1665	1666
N20M19	1496	1654	N20M44	1482	1627
N20M20	1441	1441	N20M45	1336	1336
N20M21	1327	1327	N20M46	1626	1697
N20M22	1445	1474	N20M47	1624	1682
N20M23	1270	1270	N20M48	1592	1592
N20M24	1059	1059	N20M49	1490	1490
N20M25	1288	1304	N20M50	1534	1534

n=25	Optimal	BPNN	n=25	Optimal	BPNN
N25M1	1982	2028	N25M26	1598	1598
N25M2	1646	1654	N25M27	2003	2003
N25M3	1870	1870	N25M28	2015	2039
N25M4	1949	1949	N25M29	1538	1538
N25M5	1903	1903	N25M30	2033	2050
N25M6	1680	1680	N25M31	2042	2042
N25M7	1838	1838	N25M32	1878	1878
N25M8	1978	1978	N25M33	2021	2021
N25M9	2023	2023	N25M34	1556	1556
N25M10	1727	1727	N25M35	1656	1656
N25M11	1817	1890	N25M36	1894	1895
N25M12	1810	1810	N25M37	1708	1708
N25M13	1373	1439	N25M38	1501	1542
N25M14	1980	1980	N25M39	1846	1864
N25M15	1693	1693	N25M40	1824	1824
N25M16	2019	2019	N25M41	1826	1881
N25M17	2058	2058	N25M42	1586	1586
N25M18	1889	1889	N25M43	1817	1817
N25M19	1788	1899	N25M44	1743	1743
N25M20	1950	1950	N25M45	1880	1888
N25M21	1985	1985	N25M46	1900	1900
N25M22	1864	1864	N25M47	1587	1587
N25M23	1611	1611	N25M48	1877	1877
N25M24	1769	1769	N25M49	1695	1695
N25M25	1709	1776	N25M50	1612	1612

n=30	Optimal	BPNN	n=30	Optimal	BPNN
N30M1	2270	2332	N30M26	2254	2254
N30M2	2097	2097	N30M27	2325	2325
N30M3	1788	1788	N30M28	2282	2282
N30M4	1971	1971	N30M29	2051	2057
N30M5	2280	2280	N30M30	2136	2136
N30M6	2229	2385	N30M31	1981	2075
N30M7	2016	2016	N30M32	1973	1973
N30M8	2426	2675	N30M33	2047	2047
N30M9	2219	2219	N30M34	2238	2238
N30M10	1778	1790	N30M35	2046	2050
N30M11	2113	2166	N30M36	2456	2456
N30M12	2149	2149	N30M37	2271	2271
N30M13	2171	2171	N30M38	2181	2319
N30M14	2151	2151	N30M39	2238	2299
N30M15	2053	2053	N30M40	2172	2309
N30M16	2009	2045	N30M41	2377	2377
N30M17	2104	2130	N30M42	2076	2091
N30M18	2367	2367	N30M43	2278	2391
N30M19	2227	2356	N30M44	2005	2005
N30M20	2251	2273	N30M45	2347	2628
N30M21	2049	2049	N30M46	2132	2192
N30M22	2359	2359	N30M47	2078	2078
N30M23	2300	2300	N30M48	2302	2391
N30M24	2444	2444	N30M49	2150	2252
N30M25	1650	1650	N30M50	1964	1964

n=35	Optimal	BPNN	n=35	Optimal	BPNN
N35M1	2787	2793	N35M26	2741	2741
N35M2	2561	2561	N35M27	2712	2712
N35M3	2745	2789	N35M28	2595	2595
N35M4	2361	2361	N35M29	2020	2020
N35M5	2727	2727	N35M30	2808	2808
N35M6	2707	2743	N35M31	2705	2705
N35M7	2602	2772	N35M32	2218	2218
N35M8	2768	2768	N35M33	2062	2174
N35M9	2666	3099	N35M34	2643	2685
N35M10	2622	2622	N35M35	2544	2544
N35M11	2749	2749	N35M36	2654	2654
N35M12	2334	2355	N35M37	2066	2166
N35M13	2716	2716	N35M38	2623	2623
N35M14	2743	2743	N35M39	2569	2569
N35M15	2482	2482	N35M40	2241	2241
N35M16	2459	2459	N35M41	2636	2636
N35M17	2654	2654	N35M42	2397	2502
N35M18	2308	2308	N35M43	2123	2129
N35M19	2585	2585	N35M44	2592	2749
N35M20	2670	2670	N35M45	2496	2496
N35M21	2308	2308	N35M46	2604	2604
N35M22	2254	2254	N35M47	2192	2192
N35M23	2555	2555	N35M48	2255	2255
N35M24	2725	2725	N35M49	2613	2613
N35M25	2405	2405	N35M50	2705	2705

n=40	Optimal	BPNN	n=40	Optimal	BPNN
N40M1	2797	2797	N40M26	3033	3033
N40M2	2621	2621	N40M27	3072	3072
N40M3	3079	3079	N40M28	3252	3456
N40M4	3040	3040	N40M29	2882	2882
N40M5	2674	2752	N40M30	2999	3005
N40M6	2852	2862	N40M31	3120	3120
N40M7	2111	2111	N40M32	2664	2681
N40M8	2153	2183	N40M33	3065	3065
N40M9	2670	2670	N40M34	2901	2906
N40M10	2364	2364	N40M35	2981	2981
N40M11	2952	2952	N40M36	2789	2789
N40M12	2314	2314	N40M37	3080	3138
N40M13	2445	2445	N40M38	2511	2511
N40M14	2693	2693	N40M39	2722	2722
N40M15	2078	2241	N40M40	1979	1979
N40M16	3062	3062	N40M41	2321	2321
N40M17	2636	2636	N40M42	3003	3003
N40M18	2706	2727	N40M43	3107	3123
N40M19	3112	3112	N40M44	3101	3101
N40M20	2843	2843	N40M45	2944	2944
N40M21	2900	2900	N40M46	3264	3450
N40M22	2232	2232	N40M47	2953	2976
N40M23	3059	3059	N40M48	3254	3254
N40M24	2710	2713	N40M49	2388	2566
N40M25	2512	2512	N40M50	2829	2829

n=45	Optimal	BPNN	n=45	Optimal	BPNN
N45M1	3674	4104	N45M26	3274	3455
N45M2	3348	3374	N45M27	3301	3301
N45M3	3298	3298	N45M28	2902	2902
N45M4	2883	2883	N45M29	3531	3531
N45M5	3251	3251	N45M30	3297	3297
N45M6	3456	3456	N45M31	2846	2931
N45M7	2899	2899	N45M32	3599	3637
N45M8	3446	3446	N45M33	3394	3400
N45M9	3206	3206	N45M34	3047	3047
N45M10	2484	2484	N45M35	3763	3922
N45M11	3034	3034	N45M36	3135	3158
N45M12	2532	2638	N45M37	2829	2829
N45M13	2953	2953	N45M38	3382	3382
N45M14	3256	3310	N45M39	3228	3228
N45M15	3727	3792	N45M40	3676	3676
N45M16	3399	3415	N45M41	2927	2927
N45M17	3060	3060	N45M42	2894	2894
N45M18	2749	2749	N45M43	3280	3352
N45M19	3425	3425	N45M44	2917	2917
N45M20	3060	3060	N45M45	3400	3400
N45M21	3456	3456	N45M46	2771	2781
N45M22	3268	3268	N45M47	3410	3845
N45M23	3580	4289	N45M48	3374	3374
N45M24	3469	3469	N45M49	3468	3481
N45M25	2504	2553	N45M50	3283	3283

n=55	Optimal	BPNN	n=55	Optimal	BPNN
N55M1	4111	4111	N605M26	3706	3706
N55M2	4382	4407	N55M27	4209	4209
N55M3	3786	3786	N55M28	3855	3855
N55M4	3898	4090	N55M29	4124	4124
N55M5	4084	4084	N55M30	4135	4138
N55M6	3742	4037	N55M31	4088	4173
N55M7	3956	3970	N55M32	3734	3753
N55M8	3531	3531	N55M33	4282	4282
N55M9	4015	4015	N55M34	4303	4303
N55M10	3625	3625	N55M35	3761	3761
N55M11	4113	4113	N55M36	4225	4225
N55M12	3964	3981	N55M37	3919	3919
N55M13	4413	4531	N55M38	3914	3914
N55M14	3586	3586	N55M39	4060	4060
N55M15	4321	4698	N55M40	3487	3487
N55M16	3771	3906	N55M41	3820	3820
N55M17	3950	4060	N55M42	3984	3984
N55M18	4205	4205	N55M43	3955	3955
N55M19	4281	4281	N55M44	4325	4325
N55M20	3670	3670	N55M45	3961	3965
N55M21	3920	3920	N55M46	4256	4256
N55M22	3461	3461	N55M47	4011	4011
N55M23	4282	4282	N55M48	4220	4220
N55M24	3600	3600	N55M49	3768	3768
N55M25	4451	4760	N55M50	4106	4106

n=60	Optimal	BPNN	n=60	Optimal	BPNN
N60M1	4001	4001	N60M26	4500	4500
N60M2	4556	4556	N60M27	4218	4218
N60M3	4562	4562	N60M28	3348	3348
N60M4	4197	4197	N60M29	4755	4811
N60M5	3927	3927	N60M30	4669	4669
N60M6	4801	4801	N60M31	4134	4134
N60M7	3915	3915	N60M32	4556	4556
N60M8	4523	4523	N60M33	4406	4420
N60M9	3733	3733	N60M34	3492	3495
N60M10	4697	4697	N60M35	4583	4583
N60M11	3876	3876	N60M36	3821	3821
N60M12	4546	4546	N60M37	4292	4292
N60M13	4148	4148	N60M38	4451	4451
N60M14	4585	4585	N60M39	4260	4260
N60M15	3186	3186	N60M40	4659	4659
N60M16	4599	4599	N60M41	4558	4627
N60M17	4225	4225	N60M42	4243	4243
N60M18	4443	4490	N60M43	4535	4535
N60M19	4135	4135	N60M44	4547	4773
N60M20	4060	4060	N60M45	4660	4660
N60M21	4770	4770	N60M46	3935	3935
N60M22	4460	4460	N60M47	4657	4657
N60M23	3591	3703	N60M48	4552	4552
N60M24	4304	4304	N60M49	4274	4441
N60M25	4574	4574	N60M50	4379	4379

n=75	Optimal	BPNN	n=75	Optimal	BPNN
N75M1	5568	5568	N75M26	5264	5264
N75M2	5536	5969	N75M27	5836	5836
N75M3	5246	5246	N75M28	4985	4985
N75M4	4936	4936	N75M29	5962	5962
N75M5	5544	5907	N75M30	3991	3991
N75M6	5820	6161	N75M31	5220	5220
N75M7	4026	4055	N75M32	5792	5792
N75M8	5602	5602	N75M33	5161	5161
N75M9	5890	5890	N75M34	5413	5436
N75M10	5433	5433	N75M35	5526	5526
N75M11	5794	5794	N75M36	5372	5372
N75M12	5503	5503	N75M37	5155	5157
N75M13	5153	5153	N75M38	4822	5010
N75M14	5244	5244	N75M39	5823	5823
N75M15	4750	5270	N75M40	5714	5714
N75M16	5737	5737	N75M41	5215	5215
N75M17	5947	5974	N75M42	5706	5751
N75M18	5712	5712	N75M43	5154	5154
N75M19	5479	5479	N75M44	5815	6093
N75M20	5364	5881	N75M45	4914	4914
N75M21	5870	5881	N75M46	4343	4343
N75M22	4265	4265	N75M47	5651	5651
N75M23	5619	5650	N75M48	5427	5427
N75M24	5692	5692	N75M49	5779	5779
N75M25	5470	5470	N75M50	5578	5578

n=85	Optimal	BPNN	n=85	Optimal	BPNN
N85M1	5950	5950	N85M26	6316	6316
N85M2	5819	5819	N85M27	6457	6457
N85M3	6523	6523	N85M28	5559	5559
N85M4	6274	6274	N85M29	5553	5553
N85M5	5745	5745	N85M30	6054	6054
N85M6	6642	6642	N85M31	5892	5892
N85M7	5748	5806	N85M32	6046	6046
N85M8	6781	6962	N85M33	5660	5660
N85M9	5835	5835	N85M34	5871	5871
N85M10	6740	6740	N85M35	6243	6243
N85M11	5526	5526	N85M36	6169	6169
N85M12	6446	6446	N85M37	6412	6504
N85M13	6242	6242	N85M38	6272	6272
N85M14	5973	5973	N85M39	6263	6263
N85M15	6231	6231	N85M40	4396	4464
N85M16	6485	6485	N85M41	6259	6259
N85M17	5512	5512	N85M42	6206	6206
N85M18	6100	6100	N85M43	6564	6564
N85M19	6345	6345	N85M44	6727	6741
N85M20	6447	6447	N85M45	5081	5081
N85M21	5815	5815	N85M46	5842	5842
N85M22	6603	6603	N85M47	6706	6706
N85M23	6513	6513	N85M48	5983	5983
N85M24	5571	6108	N85M49	4801	4801
N85M25	6056	6056	N85M50	6138	6138

n=100	Optimal	BPNN	n=100	Optimal	BPNN
N100M1	5379	5849	N100M26	7279	7279
N100M2	7571	7571	N100M27	7116	7116
N100M3	7187	7324	N100M28	6991	6991
N100M4	6445	6445	N100M29	5654	5654
N100M5	7727	7727	N100M30	7737	7737
N100M6	7382	7408	N100M31	6685	6685
N100M7	7605	7605	N100M32	7244	7244
N100M8	7616	7616	N100M33	7410	7411
N100M9	6866	6866	N100M34	7427	7427
N100M10	7940	7952	N100M35	6858	7113
N100M11	7810	7810	N100M36	7497	7497
N100M12	6931	6931	N100M37	7739	7739
N100M13	7705	7705	N100M38	7571	7571
N100M14	7183	7183	N100M39	7709	7709
N100M15	6204	6204	N100M40	5503	5516
N100M16	6479	6479	N100M41	7707	7707
N100M17	5926	5926	N100M42	7178	7178
N100M18	6189	6189	N100M43	6310	6491
N100M19	7839	7839	N100M44	7210	7210
N100M20	7360	7388	N100M45	5799	5799
N100M21	6124	6124	N100M46	7897	7901
N100M22	6901	6901	N100M47	7037	7037
N100M23	6862	6862	N100M48	7212	7212
N100M24	7667	7667	N100M49	7526	7526
N100M25	6276	6276	N100M50	7244	7244

Appendix E2: -Comparison of BPNN with Mean Flowtimes for optimal rule combinations for n=10 to 100

n=10	Optimal	BPNN	n=10	Optimal	BPNN
N10F1	560.5	563.6	N10F26	471.2	475.3
N10F2	518	554.6	N10F27	671.3	708.3
N10F3	559.2	569.1	N10F28	445.7	487.7
N10F4	539.7	588.6	N10F29	502.6	512.2
N10F5	400.8	409.4	N10F30	533.3	544.9
N10F6	599.7	599.7	N10F31	585.1	595.8
N10F7	510.7	519.8	N10F32	525.6	555.7
N10F8	621.2	633	N10F33	371.2	386.5
N10F9	605.9	605.9	N10F34	590.2	619.6
N10F10	534.6	534.8	N10F35	514.7	544.1
N10F11	569.5	579.9	N10F36	592.3	612.1
N10F12	535.4	562.7	N10F37	539.8	550.8
N10F13	583.1	591.5	N10F38	560.4	591
N10F14	475.1	479.9	N10F39	523.1	527.5
N10F15	464.7	469.2	N10F40	391.1	391.1
N10F16	536.7	536.7	N10F41	453.5	459
N10F17	404.7	419.6	N10F42	546.3	555.5
N10F18	554.6	573.4	N10F43	531.4	531.4
N10F19	394.1	406.2	N10F44	461.2	503.2
N10F20	633	648.8	N10F45	508.9	522.5
N10F21	508.2	583	N10F46	641.7	658.9
N10F22	403.4	417.9	N10F47	560.6	560.6
N10F23	387.1	397.7	N10F48	524	541.5
N10F24	558	590.8	N10F49	582	609.2
N10F25	500.4	511.6	N10F50	489.6	497.4

n=15	Optimal	BPNN	n=15	Optimal	BPNN
N15F1	782.69	794.67	N15F26	791.67	807.73
N15F2	678.67	679.27	N15F27	798.8	814.33
N15F3	754.87	781.67	N15F28	670.53	689.4
N15F4	623.13	631.2	N15F29	659.33	659.33
N15F5	750.53	756.53	N15F30	704.13	748.67
N15F6	770.13	776.67	N15F31	649.33	649.33
N15F7	774.13	783.73	N15F32	737.73	737.73
N15F8	816.87	823.33	N15F33	687.6	707.27
N15F9	674.67	702.93	N15F34	770.67	757.73
N15F10	701.33	707	N15F35	672	798.93
N15F11	709.2	732.4	N15F36	586.53	758.2
N15F12	656.47	656.47	N15F37	600.8	732.07
N15F13	492.13	515.2	N15F38	612.4	799.6
N15F14	536.4	538.53	N15F39	819	785.93
N15F15	713.53	720.6	N15F40	651.13	830.67
N15F16	687.07	698.87	N15F41	707.33	713.67
N15F17	798.13	834.07	N15F42	799.13	637.67
N15F18	741.2	744.2	N15F43	594.2	816.67
N15F19	688.93	720.13	N15F44	721.6	669.33
N15F20	762.47	800.6	N15F45	632.67	775.87
N15F21	625.87	646.47	N15F46	716.13	646.93
N15F22	659.73	684.33	N15F47	726.47	815.2
N15F23	564.6	568.93	N15F48	509.13	583
N15F24	721.33	744.13	N15F49	558.07	731.4
N15F25	633.87	656.73	N15F50	807.93	826.6

n=20	Optimal	BPNN	n=20	Optimal	BPNN
N20F1	933.35	945.95	N20F26	867.25	876.15
N20F2	896.5	897.8	N20F27	703.8	703.8
N20F3	705.75	738.45	N20F28	923.1	923.1
N20F4	864.75	867.45	N20F29	826.55	846.6
N20F5	827.85	852.7	N20F30	777.1	803.95
N20F6	969.2	1050.05	N20F31	952.35	955.3
N20F7	1052.35	1052.35	N20F32	902.9	950.9
N20F8	824.4	830.15	N20F33	862	866.3
N20F9	848.05	923.05	N20F34	1019.8	1053.4
N20F10	1013.83	1051.9	N20F35	823.55	823.55
N20F11	710.45	713.55	N20F36	660.6	665.75
N20F12	1000.95	1018.25	N20F37	780.35	787.9
N20F13	881.3	932.2	N20F38	973.85	983.55
N20F14	747.4	747.4	N20F39	819.2	821.55
N20F15	766.55	767.15	N20F40	870.7	882.85
N20F16	966.95	976.65	N20F41	756.4	767.15
N20F17	870.6	870.6	N20F42	858.35	858.35
N20F18	857.15	859.45	N20F43	1014.4	1051.8
N20F19	917.7	917.7	N20F44	908.6	910.9
N20F20	859.3	873.9	N20F45	787.4	808.5
N20F21	746.75	753.7	N20F46	1034.45	1064.1
N20F22	907.95	932.35	N20F47	860.7	873.2
N20F23	743.8	743.8	N20F48	910.55	910.55
N20F24	640.7	649.3	N20F49	900.35	900.35
N20F25	802.25	876.2	N20F50	903.35	910.9

n=25	Optimal	BPNN	n=25	Optimal	BPNN
N25F1	1186	1187.4	N25F26	963.16	966.4
N25F2	961.2	974.48	N25F27	1152.88	1168.36
N25F3	1025.92	1026.92	N25F28	1112	1120.12
N25F4	1142.08	1210.16	N25F29	824.96	827.2
N25F5	1107.44	1125.52	N25F30	1185.56	1209.96
N25F6	972.08	972.08	N25F31	1083.92	1098.64
N25F7	1101.2	1106.04	N25F32	1080.92	1086.4
N25F8	1082.12	1092.48	N25F33	1259.16	1297.28
N25F9	1148.68	1160.12	N25F34	866.6	870
N25F10	1023.4	1038.76	N25F35	958.72	970.8
N25F11	1025.56	1025.56	N25F36	989.76	1030
N25F12	979.28	992.44	N25F37	996.12	1002.04
N25F13	820.08	837.04	N25F38	846.08	880.6
N25F14	1191	1226.6	N25F39	1018.76	1028.96
N25F15	967.72	967.72	N25F40	1000.52	1000.68
N25F16	1145.16	1147.64	N25F41	1041.16	1056.24
N25F17	1144.44	1163.16	N25F42	908.8	927.16
N25F18	1029.08	1075.64	N25F43	946.68	999.04
N25F19	1031.52	1034.12	N25F44	1036.6	1055.48
N25F20	1096.28	1096.28	N25F45	1056.16	1109.72
N25F21	1094	1101	N25F46	1090	1135.52
N25F22	1006.16	1014	N25F47	948.72	979.84
N25F23	933.92	933.92	N25F48	1058	1082.72
N25F24	991.76	991.76	N25F49	951.84	958.4
N25F25	1034.84	1044.8	N25F50	898.56	909.32

n=30	Optimal	BPNN	n=20	Optimal	BPNN
N30F1	1221.53	1234.8	N30F26	1216.9	1228.13
N30F2	1087.5	1092.3	N30F27	1230.47	1234.4
N30F3	1045.73	1174.57	N30F28	1264.03	1266.67
N30F4	1127.37	1127.37	N30F29	1099	1099
N30F5	1251.67	1270.87	N30F30	1189.37	1192.7
N30F6	1258.93	1334.33	N30F31	1128.33	1128.4
N30F7	1058.77	1062.97	N30F32	1166.37	1172.33
N30F8	1289.8	1305.67	N30F33	1127.9	1149.87
N30F9	1292.3	1294.5	N30F34	1277.67	1277.67
N30F10	970.67	970.67	N30F35	1168.63	1197.93
N30F11	1132.1	1168.67	N30F36	1337.2	1349.87
N30F12	1165.13	1176.07	N30F37	1205.57	1205.57
N30F13	1130	1148.23	N30F38	1147.43	1192.17
N30F14	1184.73	1206	N30F39	1262.7	1266.5
N30F15	1116.33	1116.8	N30F40	1194.5	1195.83
N30F16	1100	1110.5	N30F41	1306.77	1342
N30F17	1219.7	1250.63	N30F42	1189.57	1229.03
N30F18	1362.57	1368.73	N30F43	1239.2	1251.37
N30F19	1270.97	1328.4	N30F44	1065.47	1065.47
N30F20	1343.87	1350.87	N30F45	1292.53	1316.83
N30F21	1113.03	1113.03	N30F46	1141.07	1180.23
N30F22	1306.07	1306.17	N30F47	1089.97	1111.5
N30F23	1229.43	1235.97	N30F48	1236.2	1288.9
N30F24	1374.87	1388.67	N30F49	1148.43	1161.77
N30F25	932.17	938.43	N30F50	1076.3	1076.3

n=35	Optimal	BPNN	n=25	Optimal	BPNN
N35F1	1455.86	1463.54	N35F26	1547.09	1551.91
N35F2	1369.2	1377.2	N35F27	1390.77	1390.77
N35F3	1530.29	1575.51	N35F28	1466.71	1480.31
N35F4	1302	1306.31	N35F29	1062.49	1070.43
N35F5	1537.71	1557.34	N35F30	1507.97	1507.97
N35F6	1492.11	1528.63	N35F31	1452.63	1464.26
N35F7	1545.17	1545.17	N35F32	1129.31	1135.46
N35F8	1449.03	1462.89	N35F33	1043.17	1049.11
N35F9	1393.49	1430.4	N35F34	1482.43	1521.03
N35F10	1431.09	1460.29	N35F35	1337.23	1363.14
N35F11	1491.49	1512.31	N35F36	1567	1570.06
N35F12	1273.09	1308.4	N35F37	1184.31	1199.46
N35F13	1497.54	1527.37	N35F38	1353.23	1353.23
N35F14	1472.49	1481.06	N35F39	1381.89	1455.71
N35F15	1318.34	1325.09	N35F40	1214.06	1260
N35F16	1329	1365.71	N35F41	1483.97	1517.6
N35F17	1367.89	1373.83	N35F42	1303.6	1360.29
N35F18	1298.31	1348.09	N35F43	1136.23	1138.4
N35F19	1382.11	1384.29	N35F44	1443.6	1459.37
N35F20	1421.09	1540.4	N35F45	1334.06	1390.03
N35F21	1209.46	1209.46	N35F46	1414.83	1422.31
N35F22	1241.66	1286.06	N35F47	1149.26	1159.4
N35F23	1372.74	1391.91	N35F48	1262.11	1305.97
N35F24	1447.29	1456.03	N35F49	1400.86	1469.4
N35F25	1320.83	1320.83	N35F50	1471.14	1489.69

n=40	Optimal	BPNN	n=40	Optimal	BPNN
N40F1	1497.88	1529.03	N40F26	1535.4	1543.82
N40F2	1441.12	1457.12	N40F27	1664.8	1697.7
N40F3	1579.18	1579.18	N40F28	1685.53	1699
N40F4	1633.62	1647.25	N40F29	1578.85	1628.7
N40F5	1419.3	1505.28	N40F30	1604.07	1776.43
N40F6	1487.9	1603.5	N40F31	1629.6	1629.6
N40F7	1062.43	1062.43	N40F32	1512.62	1547.32
N40F8	1210.72	1248.15	N40F33	1638.45	1719.32
N40F9	1367.9	1380.8	N40F34	1504.07	1506.7
N40F10	1216	1216	N40F35	1638.35	1716.32
N40F11	1479.68	1481.12	N40F36	1551.5	1603.8
N40F12	1224.03	1271.3	N40F37	1729.65	1763.22
N40F13	1279.03	1281.3	N40F38	1211.12	1223.6
N40F14	1435.18	1448.35	N40F39	1407.9	1434.53
N40F15	1117.43	1132.47	N40F40	1058.75	1065.72
N40F16	1693.2	1716.45	N40F41	1199.78	1202.78
N40F17	1410.09	1457.03	N40F42	1579.95	1590.5
N40F18	1475.78	1502.85	N40F43	1632.35	1668.97
N40F19	1649.03	1671.22	N40F44	1633	1649.25
N40F20	1554	1585.2	N40F45	1540.35	1540.35
N40F21	1509.62	1532.5	N40F46	1766.22	1821.53
N40F22	1198.72	1261.85	N40F47	1552.55	1554.35
N40F23	1567.7	1568.57	N40F48	1726.55	1726.82
N40F24	1516.28	1540.7	N40F49	1214.72	1216.7
N40F25	1340.5	1357.8	N40F50	1410.88	1410.88

n=45	Optimal	BPNN	n=45	Optimal	BPNN
N45F1	2025.58	2053.73	N45F26	1719.53	1741.91
N45F2	1745.09	1771.33	N45F27	1779.11	1792.64
N45F3	1638.73	1642.69	N45F28	1530.4	1540.64
N45F4	1464.33	1464.33	N45F29	1827.78	1827.78
N45F5	1727.36	1729.22	N45F30	1699.36	1729.6
N45F6	1826.71	1849.04	N45F31	1607.87	1677.02
N45F7	1508.8	1527.11	N45F32	1852	1875.42
N45F8	1914.78	1947.49	N45F33	1839.73	1845.76
N45F9	1623.09	1624.98	N45F34	1605.84	1651.53
N45F10	1371.76	1378.56	N45F35	1986.38	1986.38
N45F11	1661.07	1721.38	N45F36	1590.53	1594.31
N45F12	1358.78	1424.36	N45F37	1495.73	1530.24
N45F13	1639.4	1644	N45F38	1661.24	1669.29
N45F14	1732.96	1757.8	N45F39	1689.51	1689.73
N45F15	1862.67	1892.04	N45F40	1979.67	2028.33
N45F16	1758.11	1799.38	N45F41	1611.31	1642.44
N45F17	1655.44	1697.91	N45F42	1509.69	1571.18
N45F18	1370.36	1376.64	N45F43	1740.67	1788.6
N45F19	1835.53	1849.73	N45F44	1566.33	1615.44
N45F20	1605.04	1617.78	N45F45	1821.58	1831.22
N45F21	1823.44	1843.67	N45F46	1491.69	1527.8
N45F22	1696.71	1712.38	N45F47	1872.42	1877.51
N45F23	1920.98	1956.64	N45F48	1867.93	1922.84
N45F24	1799.69	1809.87	N45F49	1875.16	1895.2
N45F25	1384.47	1384.47	N45F50	1725.31	1744.11

n=55	Optimal	BPNN	n=55	Optimal	BPNN
N55F1	2097.36	2130.89	N55F26	1942.71	2025.45
N55F2	2272.09	2288.35	N55F27	2272.96	2352.35
N55F3	1949.64	1949.64	N55F28	2063.78	2113.98
N55F4	2103.84	2115.02	N55F29	2141.93	2240.38
N55F5	2154.02	2159.02	N55F30	2137.78	2137.78
N55F6	1980.65	1980.65	N55F31	2201.27	2212.6
N55F7	1970.53	2040.47	N55F32	1970.87	1990.71
N55F8	1766.71	1766.71	N55F33	2272.6	2279.85
N55F9	2062.84	2070.75	N55F34	2114.49	2117.05
N55F10	1892.93	1892.93	N55F35	1920.04	1924.13
N55F11	2187.91	2211.98	N55F36	2281.33	2281.33
N55F12	2028.45	2028.45	N55F37	2046.85	2087.64
N55F13	2375.44	2475.16	N55F38	1945.4	1947.93
N55F14	1936.8	2016.31	N55F39	2085.22	2086.2
N55F15	2334.51	2356.69	N55F40	1751.87	1773.6
N55F16	2046.67	2111.53	N55F41	1959.18	1960.85
N55F17	1959.47	1972.11	N55F42	1940.84	1972.98
N55F18	2180.58	2180.58	N55F43	1954.76	1970.87
N55F19	2227.65	2283.24	N55F44	2290.8	2294.85
N55F20	1934.2	1967.02	N55F45	2104.71	2104.71
N55F21	1950.02	1954.8	N55F46	2216.45	2216.45
N55F22	1924.73	1925	N55F47	2059.38	2060.58
N55F23	2252.87	2244.16	N55F48	2321.11	2351.49
N55F24	1857	1857.55	N55F49	1900.27	1900.27
N55F25	2358.55	2379.04	N55F50	2083.02	2083.02

n=60	Optimal	BPNN	n=60	Optimal	BPNN
N60F1	2086.93	2126.2	N60F26	2242.73	2244.63
N60F2	2236.13	2238.17	N60F27	2101.72	2116.62
N60F3	2400.28	2400.28	N60F28	1724.43	1760.8
N60F4	2147.48	2150.03	N60F29	2368.58	2380
N60F5	2054.78	2059.48	N60F30	2507.32	2555.77
N60F6	2465.25	2471.83	N60F31	2169.57	2239.82
N60F7	1922.17	1922.17	N60F32	2318.53	2429.42
N60F8	2204.28	2204.28	N60F33	2386.35	2477.6
N60F9	1910.17	2031.88	N60F34	1885.15	1902.03
N60F10	2463.3	2475.5	N60F35	2260.57	2262.43
N60F11	1925.38	1925.38	N60F36	2011	2012.07
N60F12	2320.52	2320.52	N60F37	2163.82	2164.8
N60F13	2201.5	2312.03	N60F38	2312.53	2474.52
N60F14	2247.55	2247.55	N60F39	2197.25	2199.28
N60F15	1718.22	1737.62	N60F40	2373.08	2386.22
N60F16	2250.02	2258.02	N60F41	2428.48	2438.78
N60F17	2278.28	2279.2	N60F42	2256	2335.1
N60F18	2434.25	2502.22	N60F43	2361.22	2363.17
N60F19	2136.57	2147.62	N60F44	2225	2227.68
N60F20	1966.78	1966.78	N60F45	2294.48	2296.52
N60F21	2397.02	2397.43	N60F46	1973.8	1974.53
N60F22	2278.28	2344.5	N60F47	2406.68	2416.08
N60F23	1921.82	1952.1	N60F48	2357	2378.88
N60F24	2141.77	2143.7	N60F49	2274.28	2309.03
N60F25	2383.47	2401.17	N60F50	2272.98	2484.62

n=75	Optimal	BPNN	n=75	Optimal	BPNN
N75F1	2851.16	2991.17	N75F26	2766.87	2884.39
N75F2	2888.11	2901.88	N75F27	3013.07	3061.61
N75F3	2748.17	2766.13	N75F28	2570.76	2588.56
N75F4	2511.31	2541.51	N75F29	3018.77	3027.73
N75F5	2890.39	2998.4	N75F30	1921.05	1924.57
N75F6	2969.59	3030.85	N75F31	2699.79	2719.08
N75F7	2138.59	2141.04	N75F32	2848.44	2848.65
N75F8	2887	2887	N75F33	2565.6	2620.24
N75F9	2931.55	2974.96	N75F34	2694.89	2694.89
N75F10	2832.8	2854.69	N75F35	2804.84	2804.84
N75F11	2987.37	2993.48	N75F36	2678.88	2686.32
N75F12	2734.88	2735.2	N75F37	2617.49	2695.67
N75F13	2700.99	2730.56	N75F38	2482.28	2584.56
N75F14	2603.77	2612	N75F39	3022.65	3022.65
N75F15	2494.01	2528	N75F40	2823.95	2828.83
N75F16	2908.52	3025.49	N75F41	2642.83	2712.24
N75F17	2986	3003.8	N75F42	2881.43	2883.49
N75F18	2805.21	2805.21	N75F43	2592.92	2602.95
N75F19	2698.67	2698.67	N75F44	2848.63	2882.16
N75F20	2769.4	2793.55	N75F45	2391.95	2396.99
N75F21	3021.55	3129.35	N75F46	2172.41	2175.57
N75F22	2088.49	2088.49	N75F47	2736.45	2736.45
N75F23	3021.55	2880.2	N75F48	2582.29	2583.37
N75F24	2932.59	2956.39	N75F49	2949.23	3048.27
N75F25	2834.67	2954.33	N75F50	2722.45	2722.89

n=85	Optimal	BPNN	n=85	Optimal	BPNN
N85F1	3047.98	3089.78	N85F26	3317.27	3348.74
N85F2	2961.82	3006.07	N85F27	3097.36	3110.96
N85F3	3330.11	3402.32	N85F28	2798.93	2803.38
N85F4	3181.13	3229.29	N85F29	2765.48	2766.25
N85F5	2815.17	2828.29	N85F30	2939.81	3114.24
N85F6	3370.56	3370.56	N85F31	3002.96	3115.6
N85F7	2951.12	2962.91	N85F32	3118.47	3283.88
N85F8	3414.59	3414.79	N85F33	2959.16	2979.38
N85F9	3014.12	3019.28	N85F34	2877.71	2887.12
N85F10	3445.29	3507.76	N85F35	3084.39	3107.14
N85F11	2862.02	2872.59	N85F36	3203.53	3241.13
N85F12	3295.66	3295.66	N85F37	3169.92	3172.2
N85F13	3244.08	3305.55	N85F38	3077.67	3095.65
N85F14	2891.46	3021.85	N85F39	3191.71	3250.72
N85F15	3142.85	3186.96	N85F40	2162.98	2214.61
N85F16	3305.38	3357.52	N85F41	3208.14	3251.32
N85F17	2817.04	2884.08	N85F42	3119.93	3188.22
N85F18	3112.8	3182.36	N85F43	3175.6	3177.66
N85F19	3078.24	3080.74	N85F44	3428.38	3432.4
N85F20	3182.84	3313.68	N85F45	2586.18	2606.24
N85F21	2965.42	2977.95	N85F46	2878.28	2878.28
N85F22	3296.13	3309.12	N85F47	3301.33	3301.33
N85F23	3155.89	3228.99	N85F48	3093.54	3093.54
N85F24	2856.11	2872.75	N85F49	2417.16	2433.12
N85F25	3075.45	3118.69	N85F50	3149.61	3215.18

n=100	Optimal	BPNN	n=100	Optimal	BPNN
N100F1	2585.32	2655.24	N100F26	3458.92	3458.92
N100F2	3678.62	3679.99	N100F27	3429.26	3430.57
N100F3	3461.73	3464.72	N100F28	3469.93	3473.85
N100F4	3218.08	3218.08	N100F29	2853.93	2861
N100F5	3923.19	4086.99	N100F30	3919.49	3955.75
N100F6	3646.68	3646.68	N100F31	3305.1	3305.1
N100F7	3827.34	3926.24	N100F32	3643.62	3709.39
N100F8	3718.65	3718.65	N100F33	3583.86	3593.1
N100F9	3457.39	3495.47	N100F34	3709.51	3711.48
N100F10	3971.52	4000.86	N100F35	3433.84	3444.6
N100F11	4022.08	4062.28	N100F36	3671.19	3678.15
N100F12	3397.91	3520.62	N100F37	3775.62	3775.62
N100F13	3803.01	4071.1	N100F38	3850.58	3861.1
N100F14	3546	3616	N100F39	3770.58	3770.58
N100F15	3084.2	3137.01	N100F40	2721.59	2835.38
N100F16	3128.92	3143.04	N100F41	3809.51	3812.03
N100F17	3063.13	3075.42	N100F42	3603.32	3614.64
N100F18	3170.24	3175.86	N100F43	3237.57	3241.3
N100F19	3792.36	3808.72	N100F44	3614.89	3702.32
N100F20	3734.7	3763.04	N100F45	2764.21	2777.81
N100F21	2913.15	2913.15	N100F46	4021.82	4093.29
N100F22	3267.38	3267.38	N100F47	3508.76	3577.05
N100F23	3305.38	3312.41	N100F48	3509.49	3517.79
N100F24	3933.6	4016.9	N100F49	3790.38	3821.67
N100F25	2985.09	2990.76	N100F50	3602.44	3648.29