

2024-08-01

## Flow-Shop Scheduling Optimization Through Discrete-Event Simulation And Neural Networks.

Jesus Ricardo Herrera Garfio  
*University of Texas at El Paso*

Follow this and additional works at: [https://scholarworks.utep.edu/open\\_etd](https://scholarworks.utep.edu/open_etd)



Part of the [Engineering Commons](#)

---

### Recommended Citation

Herrera Garfio, Jesus Ricardo, "Flow-Shop Scheduling Optimization Through Discrete-Event Simulation And Neural Networks." (2024). *Open Access Theses & Dissertations*. 4183.  
[https://scholarworks.utep.edu/open\\_etd/4183](https://scholarworks.utep.edu/open_etd/4183)

This is brought to you for free and open access by ScholarWorks@UTEP. It has been accepted for inclusion in Open Access Theses & Dissertations by an authorized administrator of ScholarWorks@UTEP. For more information, please contact [lweber@utep.edu](mailto:lweber@utep.edu).

FLOW-SHOP SCHEDULING OPTIMIZATION THROUGH DISCRETE-EVENT  
SIMULATION AND NEURAL NETWORKS.

JESUS RICARDO HERRERA GARFIO

Master's Program in Industrial Engineering

APPROVED:

---

Tzu-Liang (Bill) Tseng, Ph.D., Chair

---

Ivan A. Renteria Marquez, Ph.D., Co-Chair

---

Yirong Lin, Ph.D.

---

Stephen L. Crites, Jr., Ph.D.  
Dean of the Graduate School

Copyright ©

by

JESUS RICARDO HERRERA GARFIO

2024

## **Dedication**

To my family and my girlfriend, for your love, patience, and belief in me.

FLOW-SHOP SCHEDULING OPTIMIZATION THROUGH DISCRETE-EVENT  
SIMULATION AND NEURAL NETWORKS

by

JESUS RICARDO HERRERA GARFIO

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Industrial, Manufacturing and Systems Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

August 2024

## **Acknowledgements**

I would like to thank my advisor, Dr. Renteria, for his constant support and mentorship. His knowledge and guidance have been essential to the success of this work.

I also thank Dr. Tseng for his financial support and dedication to industrial projects. His contributions made this project possible.

## **Abstract**

Industrial developments over the past few decades have transformed manufacturing processes, emphasizing the necessity for efficient scheduling systems. The shift from manual scheduling to automated and optimized systems has underscored the need for innovative approaches to improve production efficiency.

Scheduling optimization of manufacturing systems is critical to improve operational efficiency. Typically, these scheduling problems become challenging because they involve coordinating various production tasks, resource management, and makespan minimization. These challenges are particularly evident in flow-shop manufacturing systems, where the operation sequencing significantly impacts overall performance. The main objective of this manuscript is to present the methodology followed to optimize the schedule of a flow-shop manufacturing system through the combination of discrete-event simulation and neural networks. With a specific focus on enhancing operational efficiency, the study aimed to employ the built-in neural network capabilities of Simio simulation software to optimize the makespan, consequently reducing the overall production time required for processing manufacturing orders. By exploring the integration of advanced simulation techniques with neural network functionalities, this research highlights the advantages of this powerful tool to model and optimize complex manufacturing processes. Moreover, the study compared the Simio with neural networks approach and Palmer Heuristic method to assess their respective advantages and disadvantages and emphasized the potential benefits for the manufacturing industry by implementing these methodologies to facilitate informed decision-making and drive improvements in productivity and resource utilization within the context of manufacturing operations.

The findings from this research offer valuable insights into the effectiveness of combining discrete-event simulation with neural networks, providing a framework for future studies and practical applications in manufacturing systems. The demonstrated improvements in makespan and overall operational efficiency highlight the potential for these advanced techniques to be adopted in different industrial settings.



## Table of Contents

Dedication .....	iii
Acknowledgements .....	v
Abstract .....	vi
Table of Contents .....	viii
List of Tables .....	x
List of Figures .....	xi
Nomenclature .....	xii
Chapter 1: Introduction .....	1
1.1 Literature review .....	2
1.1.1 Overview of flow-shop scheduling problems .....	2
1.1.2 Traditional scheduling methods .....	4
1.1.3 Advances in discrete-event simulation .....	5
1.1.4 Application of neural networks in production scheduling .....	6
1.1.4.1 The artificial Neural Network Technology .....	7
1.2 Thesis objectives .....	8
Chapter 2: Case Study and Problem Definition .....	10
2.1 Description of the industrial flow-shop case study .....	10
2.2 Processing times and job sequences .....	12
2.3 Identification of key performance indicators .....	12
2.3.1 Makespan .....	12
2.4 Notation definition .....	13
Chapter 3: Proposed Methodology .....	15
3.1 Research design and framework .....	15
3.2 Discrete-event simulation integration .....	16
3.2.1 Simio software features .....	16
3.2.1.1 Simio Key Components .....	16
3.2.2 Neural networks integration .....	20
3.2.3 Plan for data collection and processing .....	23

3.3	Palmer Heuristic integration .....	24
3.4	performance metrics and evaluation criteria.....	25
Chapter 4: Simulation Model Development .....		26
4.1	Design and setup of the simulation model in Simio .....	26
4.2	Definition of input parameters and constraints.....	27
4.2.1	Entities .....	27
4.2.2	Sources.....	29
4.2.3	Servers.....	30
4.3	Training and implementation of the neural network model.....	30
4.3.1	Setting the neural network properties .....	30
4.3.2	Collecting data for training .....	33
4.3.3	Training of the neural network .....	34
4.3.4	Implementation of neural network.....	34
Chapter 5: Results and Conclusions .....		36
5.1	Performance of the Simio neural network optimization .....	36
5.2	Results of the Palmer Heuristic algorithm .....	37
5.3	Conclusion .....	38
References.....		40
Appendix.....		42
A1.1	Python code for Palmer's Heuristic implementation .....	42
Vita		45

## **List of Tables**

Table 2.1 Processing times for each job on each machine .....	12
Table 2.2 Nomenclature for completion time calculation of FSSP .....	13
Table 3.1 Key components of FSSP simulation .....	17
Table 5.1 Average time in system (hours) .....	36

## List of Figures

Figure 1.1 Representation of an Artificial Neural Network (ANN) .....	8
Figure 2.1 Processing flow of parts in flow shop .....	11
Figure 2.2 Illustrative examples of final product's assembly levels.....	11
Figure 3.1 Simio NN integration process .....	20
Figure 3.3 Training parameters .....	24
Figure 4.1 2D view for the simulation model .....	27
Figure 4.2 Entity properties .....	28
Figure 4.3 Sequence table .....	29
Figure 4.4 Source properties .....	29
Figure 4.5 Server properties.....	30
Figure 4.6 Neural network properties .....	31
Figure 4.7 Input value expressions .....	32
Figure 4.8 Save inputs trigger .....	32
Figure 4.9 Save actual trigger .....	33
Figure 4.10 Training data.....	33
Figure 4.11 NN trainer properties .....	34
Figure 4.12 NN implementation .....	35
Figure 5.1 Schedule without optimization. ....	36
Figure 5.2 Schedule optimized by NN.....	37
Figure 5.3 Schedule optimized by Palmer heuristics.....	38

## **Nomenclature**

### **Acronyms**

DES: Discrete-Event Simulation

FSSP: Flow-Shop Scheduling Problem

NN: Neural Networks

ANN: Artificial Neural Networks

GA: Genetic Algorithms

SA: Simulated Annealing

PSO: Particle Swarm Optimization

B&B: Branch and Bound

MILP: Mixed-Integer Linear Programming

OR: Operations Research

BOM: Bill of Materials

SUA: Sub-Assemblies

FIFO: First-In, First-Out

## **Chapter 1: Introduction**

Efficient scheduling in manufacturing systems is crucial for optimizing operational efficiency, particularly in the context of flow-shop environments. The challenges of coordinating diverse production tasks and minimizing makespan have led to the exploration of innovative methodologies. Hence, this research employs the powerful combination of discrete-event simulation and Simio neural networks to optimize scheduling in flow-shop manufacturing systems.

The integration of Simio's neural networks served as a cornerstone of the methodology. By utilizing the built-in neural network capabilities within Simio simulation software, the study aims to optimize the makespan and reduce the overall production time required for processing manufacturing orders. Successful implementations in related fields inspired this approach, where the integration of simulation and neural networks had proven to be a potent tool for process optimization. Furthermore, the Palmer Heuristic method was incorporated as a complementary approach to scheduling optimization. This method offered a systematic approach to job sequencing, known for its effectiveness in finding near-optimal solutions for flow-shop scheduling problems.

Utilizing both Simio's neural networks and the Palmer Heuristic method allowed for a comparative analysis to determine if both approaches could achieve similar results, which is particularly crucial in flow-shop scheduling problems, where attaining an optimal schedule is paramount for minimizing makespan. Each method presents distinct limitations and challenges. The limitations of Simio NN primarily arise from its black box nature, hindering interpretability and diagnostic capabilities, and posing challenges in domains with limited data availability. On the other hand, implementing the Palmer heuristic can be challenging due to the need to accurately translate its algorithmic steps into code, handle complex data structures, and optimize computation

efficiency. By evaluating the efficiency and effectiveness of each method, the study seeks to offer insights into their individual strengths and weaknesses in optimizing flow-shop scheduling. Recent studies have demonstrated the effectiveness of integrating neural networks into simulation models for optimizing manufacturing processes. Their findings have showcased notable improvements in makespan reduction and operational efficiency, providing a basis for the exploration of this combined approach. By building upon the insights gained from prior research, this study contributes to the understanding of the application of advanced simulation techniques and heuristic methods in the context of flow-shop scheduling optimization.

## **1.1 LITERATURE REVIEW**

### **1.1.1 Overview of flow-shop scheduling problems**

Flow-shop scheduling problems (FSSP) are a class of production scheduling problems where  $n$  jobs  $\{J_1, J_2, \dots, J_n\}$  must pass through a series of  $m$  machines  $\{M_1, M_2, \dots, M_m\}$  in a specific order. Each job has an identical flow pattern, must visit each machine exactly once, and the order of machines is the same for all jobs.

The processing of job  $J_j$  on machine  $M_i$  is called an operation, denoted by  $O_{ij}$ . For each operation  $O_{ij}$ , there is an associated processing time  $t_{ij}$ . In addition, there may be a ready time (or release date)  $r_j$  associated with each job, at which time  $J_j$  is available for processing, and/or a due date  $d_j$ , by which time  $J_j$  should be completed. A schedule in this context is an assignment of jobs over time onto machines. The scheduling problem is to find a schedule that optimizes some performance measure. The following assumptions appear frequently in scheduling theory literature:

1. Machines are always available and never break down.
2. Each machine can process at most one job at any time.
3. Any job can be processed on at most one machine at any time.

4. Ready times of all jobs are zero, i.e., all jobs are available at the commencement of processing.
5. No pre-emption is allowed; once an operation is started, it is continued until complete.
6. Setup times are independent of the schedules and are included in processing times.
7. Processing times and technological constraints are deterministic and known in advance, and similarly for due dates, where appropriate [1].

The main objective in FSSP is typically to minimize the makespan, which is the total time required to complete all jobs. For a comprehensive review of flow shop scheduling problems, one can refer to Linn and Zhang (1999) [2]. FSSP is characterized by the following elements:

- Jobs and Machines: A finite set of jobs, each requiring a sequence of operations performed on a finite set of machines.
- Processing Order: Each job must follow a specific sequence of operations across the machines. The order is predefined and identical for all jobs.
- Processing Times: Each operation has a specific processing time that can vary for each job and machine.
- Constraints: Common constraints include no job splitting, where a job cannot be interrupted once started on a machine, and machine availability, where each machine can process only one job at a time.

FSSP is considered NP-hard [3], meaning that finding an optimal solution requires computational time that increases exponentially with the problem size. This complexity arises from the large number of possible job sequences and the need to account for various constraints and processing times. Key challenges in FSSP include:

- Makespan Minimization: Achieving the shortest possible total elapsed time required to process all jobs on all machines [4].
- Resource Allocation: Efficiently assigning jobs to machines to minimize idle times and maximize utilization.



- Sequencing and Scheduling: Determining the optimal order in which jobs should be processed on each machine to minimize delays and bottlenecks.

### **1.1.2 Traditional scheduling methods**

Like many OR application areas, the study of scheduling theory began in the early 1950s. Johnson's article (Johnson 1954) [5], is acknowledged as a pioneering work. Jackson (1955) [6] and Smith (1956) [7] provided various optimal rules for single-machine problems. These early works formed the basis for much of the development of classical scheduling theory.

In the 1970s, theoretical work on problem complexity began. It was found that most problems are NP-hard (Lenstra et al. 1977) [8]. Fast optimal algorithms are unlikely to exist for these problems. The effectiveness of heuristic algorithms was studied by theoretical analysis and computational experiments [1].

There are many scheduling problems that are intrinsically very hard, i.e., NP-hard. They cannot be formulated as linear programs and there are no simple rules or algorithms that yield optimal solutions in a limited amount of computer time. It may be possible to formulate these problems as integer or disjunctive programs, but solving these to optimality may require an enormous amount of computer time [9].

One of the traditional methods for solving scheduling problems is the use of heuristic methods, which provide good solutions in a reasonable time frame. These methods, as discussed by Thomas L. Saaty, are often based on rules or strategies that guide the search for feasible solutions in complex optimization problems [10]. Examples include Johnson's rule, which provides an optimal solution for two-machine flow-shop problems, and the Palmer Heuristic, which uses priority rules to sequence jobs. One advantage of heuristic methods like the Palmer Heuristic is their efficiency and quick processing, making them suitable for real-time applications and providing satisfactory solutions.

Metaheuristic algorithms are advanced strategies that guide the search process to explore the solution space more effectively. Examples include Genetic Algorithms (GA), Simulated Annealing (SA), and Particle Swarm Optimization (PSO). These methods are often used for larger, more complex problems due to their ability to find near-optimal solutions within acceptable computational times. However, a disadvantage of metaheuristic algorithms is that they do not guarantee an optimal solution and their performance can be sensitive to the choice of parameters and the specific problem instance.

Exact algorithms guarantee an optimal solution by exhaustively exploring all possible solutions. Examples include Branch and Bound (B&B) and Mixed-Integer Linear Programming (MILP). These methods are typically limited to smaller problem instances due to their high computational demands, making them impractical for larger, more complex scheduling problems.

### **1.1.3 Advances in discrete-event simulation**

Discrete-event simulation (DES) has evolved significantly in recent years, particularly in its application to flow-shop scheduling problems. DES involves modeling systems as a series of discrete events over time, where each event represents a change in the system's state. This method allows for a detailed representation of complex manufacturing processes, enabling researchers and practitioners to analyze system behavior, optimize processes, and test scenarios under controlled conditions.

In the context of flow-shop scheduling, advancements in DES have led to more robust tools and methodologies for optimizing scheduling decisions. Researchers such as Benjamin W. Johnson and Jeffrey S. Smith have leveraged DES not only to simulate and analyze existing scheduling strategies but also to develop and test novel approaches. This includes the integration of machine learning algorithms and optimization heuristics, thereby pushing the boundaries of what can be achieved in terms of scheduling efficiency and operational effectiveness.

Simulation has been extensively utilized to address various short-term decision-making challenges in manufacturing. Thomas and Charpentier [11] illustrate the benefits of constructing simplified models with reduced elements, connections, or calculations to enhance scheduling efficiency in manufacturing systems. Lejmi and Sabuncuoglu [12], on the other hand, employ simulation coupled with statistical analysis to assess how variations in load, processing time, and due dates impact scheduling system performance within manufacturing environments. These studies underscore the versatility of simulation techniques in optimizing manufacturing operations by providing insights into system behavior under different conditions [13].

Advances in discrete-event simulation (DES) have played a crucial role in the development of software platforms like Simio LLC, which specializes in simulation software widely used for addressing complex flow-shop scheduling problems (FSSP). Simio is a leading platform in discrete-event simulation, offering innovative solutions for modeling manufacturing processes, logistics, healthcare, and aerospace industries. The software integrates advanced algorithms to effectively manage uncertainty and variability in FSSP, significantly enhancing operational efficiency and decision-making across various industries.

#### **1.1.4 Application of neural networks in production scheduling**

In recent years, the application of neural networks has gained significant attention in the manufacturing sector, revolutionizing traditional approaches to optimizing production processes.

The manufacturing industry has experienced an unprecedented degree of change, including global competition, shortened product life cycles, shifts in management, increasing quality demands, higher customer expectations, rapid advances in complex technologies, and a vast array of options in materials and processes. Today's companies are challenged not only to adapt to these evolving conditions but also to harness them strategically to maintain a competitive advantage [14].

Neural networks have become widely recognized in manufacturing due to their capability to handle complex patterns and nonlinear relationships in production systems. In flow-shop scheduling, neural networks are employed to optimize scheduling decisions by learning from historical data and real-time inputs. This approach enables neural networks to predict processing times more accurately than traditional methods, thereby improving the overall efficiency of manufacturing operations.

#### ***1.1.4.1 The artificial Neural Network Technology***

Artificial Neural Networks (ANN) Are a type of machine learning algorithm that is inspired by the structure and function of biological neurons in the human brain. The fundamental concept of ANNs is the structure of the information processing system. Composed of a large number of highly interconnected processing units, “neurons” connected into networks, a neural network system uses the human-like technique of learning by example to resolve problems [15].

Neural networks (NN) are models designed to detect and predict complex relationships within data sets, making them highly effective in tasks such as forecasting future trends, classifying data, and optimizing decision-making processes. NNs are designed to model and predict complex relationships between input data and output data (see Figure 1.1), making them useful in applications such as forecasting, classification, and optimization. These networks function by systematically adjusting parameters known as weights, which determine the strength and direction of connections between individual neurons. During the training phase, neural networks learn from

historical data inputs and corresponding outputs, iteratively refining their weights to minimize the disparity between predicted outcomes and actual results.

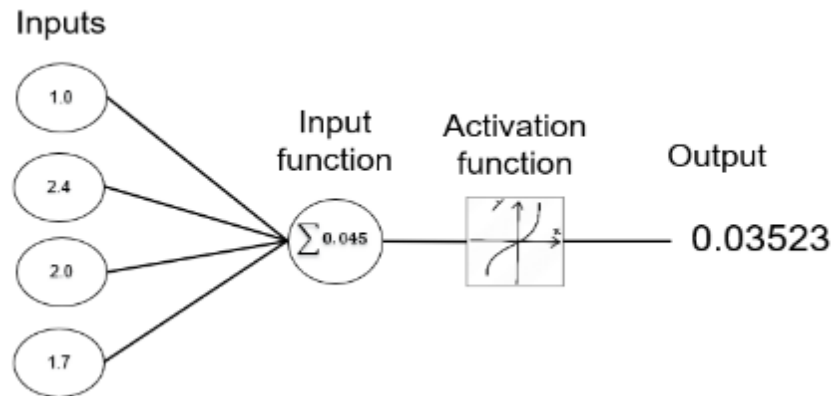


Figure 1.1 Representation of an Artificial Neural Network (ANN)

Through this iterative learning process, neural networks gain an understanding of the patterns and structures present in the data. This acquired knowledge enables them to make accurate predictions when presented with new, previously unseen data. In the context of manufacturing and flow-shop scheduling, neural networks enable the optimization of production schedules by leveraging historical performance data and real-time operational inputs. By continuously adapting and refining their predictive capabilities, neural networks contribute significantly to enhancing efficiency and decision-making within manufacturing environments.

One significant application of neural networks in manufacturing is their integration with simulation tools like Simio. By incorporating neural networks within simulation models, researchers can simulate various scheduling scenarios and optimize schedules dynamically.

## 1.2 THESIS OBJECTIVES

The primary objective of this thesis is to apply Simio software's neural network capabilities in a production scheduling case study to evaluate the efficiency of discrete-event simulation and neural network integration. This research will utilize Simio's advanced features to minimize the makespan in a specific manufacturing system, thereby enhancing overall operational efficiency.

By comparing the performance of Simio's neural network approach with the Palmer Heuristic method, the study aims to assess their effectiveness in addressing the complexities of flow-shop scheduling and providing practical solutions applicable to real-world manufacturing environments.

Additionally, this thesis seeks to provide insights into the strengths and limitations of employing neural networks and Palmer Heuristic for scheduling optimization. Through a detailed documentation of the integration process and an evaluation of their impact on scheduling performance, the research will analyze how these techniques can be optimally utilized. The goal is to emphasize the potential advantages of these approaches, including potential reductions in production times and enhancements in resource utilization. Ultimately, this study aims to establish a foundation for further research and practical implementation in the field of manufacturing operations.

## **Chapter 2: Case Study and Problem Definition**

Efficient scheduling is crucial in improving operational performance within manufacturing processes, offering advantages such as reduced lead times, optimized resource utilization, and, ultimately, cost savings. Moreover, schedule optimization holds the potential to streamline workflows, ensuring tasks are completed in a timely and cost-effective manner. The presented case study aims to optimize the schedule of an industrial flow-shop, which is described in the following sections.

### **2.1 DESCRIPTION OF THE INDUSTRIAL FLOW-SHOP CASE STUDY**

In this study, each machine serves as an assembly operation station responsible for a specific part of the final product's manufacturing process. Upon completion of processing through all three machines, the final product is considered finished.

The case study centers around four jobs that must pass through three machines in a predefined order (see Figure 2.1). In this FSSP, the machine stages are represented by Machine 1, Machine 2, and Machine 3. Each job has a different processing time for each machine. The goal is to identify an optimal job completion sequence, reduce the makespan, and enhance overall operational efficiency.

Once each job passes through the three machines, it will be considered a finished product. Finished products consist of several components that need to be assembled according to a hierarchical assembly structure. These components are sub-assembly operations, called sub-assemblies (SUA). Figure 2.2 presents an illustrative example of the final products with a hierarchical assembly structure. The assembly structure has a tree configuration, where the final node (machine 3) represents the final assembly operation. Final products have simple hierarchical assembly structures involving just one assembly operation in each machine. They are completed

in the assembly stage, where each product is built by assembling the processed parts according to the Bill of Materials (BOM). For this case study, it is assumed that each assembly stage (machine operation) has infinite BOM capacity, meaning material capacity constraints are not considered as a factor in optimizing the FSSP. The goal is to find the assembly operations sequence that optimizes a specified performance measure, such as the makespan, which means determining the order in which jobs should be processed for maximum efficiency.

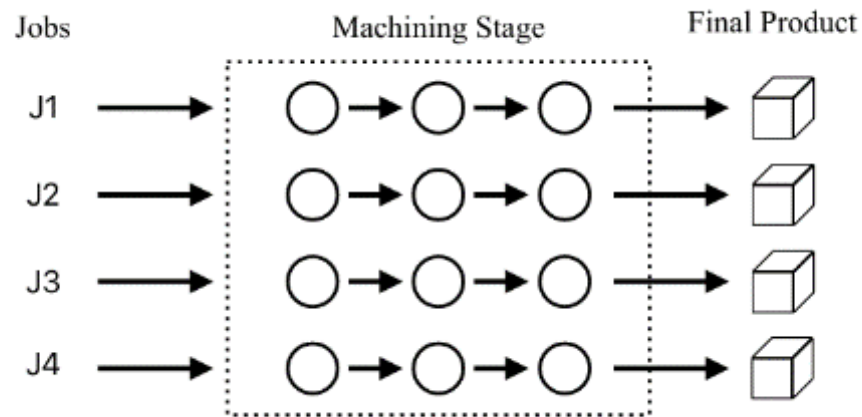


Figure 2.1 Processing flow of parts in flow shop

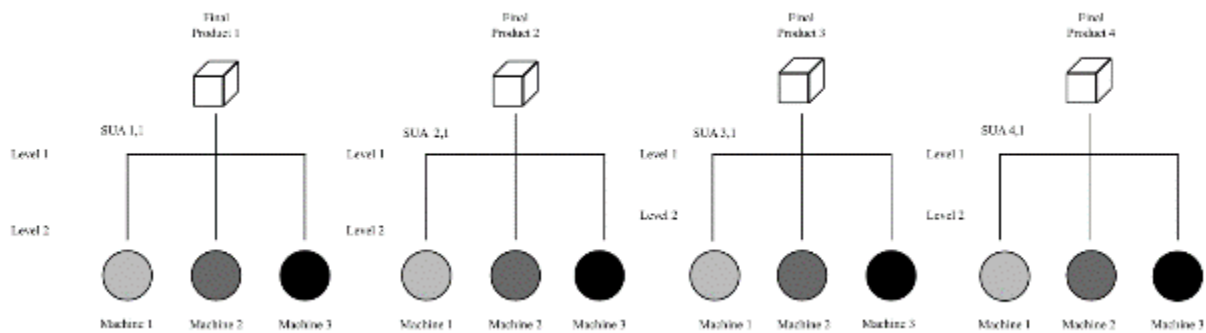


Figure 2.2 Illustrative examples of final product's assembly levels.



## 2.2 PROCESSING TIMES AND JOB SEQUENCES

The variability in processing times for each job across Machine 1, Machine 2, and Machine 3 adds complexity to the scheduling problem. The distinct processing times for different jobs on each machine imply that the sequence in which jobs are processed directly influences the makespan and overall efficiency of the flow-shop scheduling system. The production sequences and processing times are specified in Table 2.1. This table specifies the required processes per job type, where  $j$  and  $k$ , specifies the machine where the part must be processed on, and the task processing time.

Table 2.1 Processing times for each job on each machine

Job ( $j$ )	Machine processing times in minutes ( $k$ )		
	M1	M2	M3
1	6	5	10
2	8	9	7
3	3	8	6
4	4	7	9

Effective job sequencing balances the workload across machines, reduces idle time, and ensures the timely completion of all jobs by determining the optimal order in which jobs should be processed through the machines to minimize makespan. In this study, methods for estimating processing times and strategies for sequencing jobs are explored to improve the operational performance of flow-shop manufacturing systems.

## 2.3 IDENTIFICATION OF KEY PERFORMANCE INDICATORS

### 2.3.1 Makespan

By minimizing the makespan, manufacturers can ensure that all jobs are completed in the shortest possible time, reducing idle times, improving machine utilization, and increasing overall efficiency. This leads to cost savings and higher customer satisfaction due to timely deliveries.

Given its significance, optimizing the makespan has been the focal point of this study. Different techniques have been developed and employed to achieve this optimization. Among these techniques, discrete-event simulation (DES) and heuristic methods stand out. Discrete-event simulation provides a powerful tool for modeling complex manufacturing systems and evaluating the impact of different scheduling strategies on the makespan. By simulating various scenarios, DES helps in identifying bottlenecks and testing alternative solutions without disrupting the actual production process.

Heuristic methods, on the other hand, offer practical and often computationally efficient approaches to find near-optimal solutions for minimizing the makespan. These methods, such as the Palmer Heuristic, are designed to generate good scheduling sequences based on certain rules or heuristics.

## 2.4 NOTATION DEFINITION

The notation definition used for describing the scheduling problem in a flow-shop manufacturing environment is presented in table 2.2.

Table 2.2 Nomenclature for completion time calculation of FSSP

<b>Indices</b>	<b>Parameters:</b>
$j$ : Index of jobs, where: $j = 1, 2, \dots, n$	$n$ : Number of jobs
$k$ : Index of machines, where: $k = 1, 2, \dots, m$	$m$ : Number of machines
<b>Decision variables:</b>	<b>Processing time:</b>
$C_{j,k}$ : Completion time of job $j$ on machine $k$	$t_{j,k}$ : Time taken for job $j$ to be processed on machine $k$

Assuming  $(j_1, j_2, \dots, j_n)$  as jobs' sequence, the completion time  $C_{j,k}$  for each job  $j$  on each machine  $k$  is calculated as follows:

- Start with the first job on the first machine:

$$C_{1,1} = t_{1,1}$$

- For the first job on subsequent machines:

$$C_{1,k} = C_{1,k-1} + t_{1,k} \text{ for } k = 2, \dots, m$$

- For the remaining jobs on the first machine:

$$C_{j,1} = C_{j-1,1} + t_{j,1} \text{ for } j = 2, \dots, n$$

- For the remaining jobs on the remaining machines:

$$C_{j,k} = \max(C_{j-1,k}, C_{j,k-1}) + t_{j,k} \text{ for } j = 2, \dots, n \text{ and } k = 2, \dots, m$$

The makespan, which represents the completion time of the last job on the last machine, is given by:

$$\text{Makespan} = C_{n,m}$$

This notation and calculation method form the basis for analyzing scheduling efficiency and optimizing flow-shop manufacturing systems.

### **Chapter 3: Proposed Methodology**

The proposed methodology for optimizing the flow-shop schedule involves a systematic and iterative process to compare the effectiveness of Simio with neural networks and Palmer's Heuristic.

#### **3.1 RESEARCH DESIGN AND FRAMEWORK**

The process begins with simulating the flow-shop schedule without NN optimization using Simio, establishing a baseline makespan.

Next, neural networks are integrated into the Simio environment and trained using relevant data to understand system patterns. The trained neural network is then applied to optimize the schedule, and its impact on performance is evaluated. Subsequently, Palmer's Heuristic is implemented in Python to find an optimal routing solution, which is then used to optimize the schedule, and its performance is assessed.

A comparative analysis is conducted between the results obtained from the Simio with neural networks optimization and Palmer's Heuristic. The analysis considers computational efficiency, accuracy, and ease of implementation. Validation and sensitivity analysis are performed to confirm the results and identify significant factors affecting each method's performance.

Finally, the entire methodology, including simulation parameters, neural network parameters, and the Python code for Palmer's Heuristic, is documented. A comprehensive report is prepared to highlight findings and provide recommendations.

The following sections will detail the tools and techniques used in this research, including the functionalities of Simio, the plan for data collection and processing, the integration of neural networks, and the implementation of Palmer's Heuristic.

## **3.2 DISCRETE-EVENT SIMULATION INTEGRATION**

### **3.2.1 Simio software features**

Simio software is a powerful tool for discrete-event simulation, offering features that make it highly useful for modeling and optimizing manufacturing systems. One of its advantages is its object-oriented modeling environment, which allows users to create detailed and realistic representations of their systems. This is particularly beneficial for flow-shop scheduling, where understanding the impact of different job sequences on overall system performance is crucial.

Another significant feature of Simio is its neural network capabilities. This tool enables the utilization of data derived from validated simulations to train the neural network, thereby optimizing production processes. This approach ensures higher reliability and accuracy in the training process, leading to more effective and efficient production schedule optimizations.

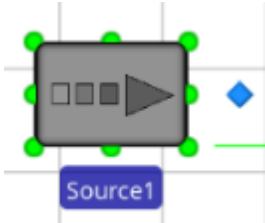
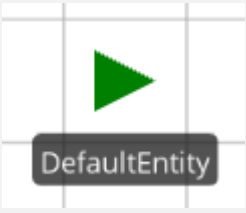
Simio software includes built-in templates and libraries with objects that can significantly reduce the time required to develop and test simulation models. The following section defines the objects and tools employed in this simulation of FSSP.

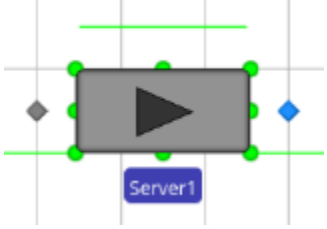
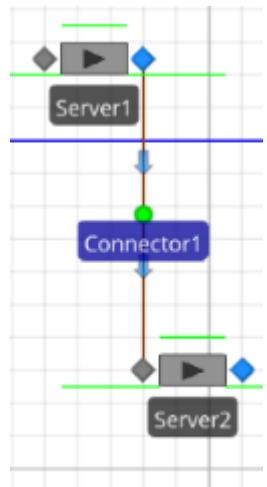
#### ***3.2.1.1 Simio Key Components***

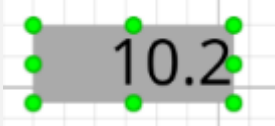
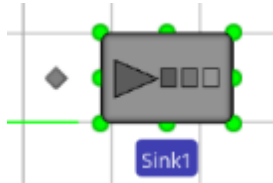
Simio intelligent objects are key components in modeling and simulating complex systems. These objects, along with tools like sequence tables, enable the simulation of system workflows and facilitate optimization efforts.

Table 3.1 presents each of the key components utilized for the simulation of the FSSP.

Table 3.1 Key components of FSSP simulation

<p><b>Source Objects</b></p>	<p>Their primary role is to generate or "create" entities within the model, which represent units of work, orders, products, or other items flowing through the simulated system. These objects can be configured with specific rules that govern the arrival rate of entities into the system, as well as other parameters related to the generation and initial behavior of simulated entities.</p>	
<p><b>Entity objects</b></p>	<p>Represents an object that moves through a simulation model. Entities typically represent items, such as parts, orders, or jobs that flow through a system being modeled. They can carry attributes and follow paths defined by the simulation logic. Entities are fundamental for simulating how real-world objects move and interact within a simulated environment, allowing analysts to study system behavior, optimize processes, and evaluate performance metrics such as throughput, utilization, and waiting times.</p>	

<p><b><i>Server object</i></b></p>	<p>Represents a resource or a work center that manages the processing of entities according to defined schedules or rules within a simulation model. These resources can be configured to handle different types of tasks, processing times, and capacities, which influence the flow and efficiency of entities through the system being modeled.</p>	 <p>The diagram shows a grey rectangular box with a black right-pointing triangle in the center, representing a server. It is positioned between two horizontal green lines. On the left side, there is a grey diamond. On the right side, there is a blue diamond. Below the box, the text 'Server1' is written in a blue box.</p>
<p><b><i>Sequence table</i></b></p>	<p>It is a tool used to define and manage the order in which entities move through a simulation model. It specifies the sequence in which tasks or operations are performed on entities as they progress through various stages of a system or process.</p>	
<p><b><i>Connector object</i></b></p>	<p>These objects define pathways through which entities, such as products or resources, travel from one location to another. They are configured with attributes like capacity and speed and integrated with logic for routing and decision-making. These pathways connect different parts of the model, representing movement between stations, machines, or process steps.</p>	 <p>The diagram shows a vertical pathway connecting two server objects. At the top is a grey box with a right-pointing triangle labeled 'Server1'. At the bottom is a similar grey box labeled 'Server2'. A vertical line with a downward-pointing arrow connects them. In the middle of this line is a blue box labeled 'Connector1'. The entire pathway is flanked by horizontal green lines at the top and bottom, with grey and blue diamonds at the ends.</p>

<p><b><i>Status labels</i></b></p>	<p>Label associated with an entity, object, or state within a simulation model. It is used to indicate the current status, condition, or state of an entity or component in the simulated system. This label can dynamically change as the simulation progresses based on predefined conditions or events modeled within Simio. It helps in monitoring and analyzing the behavior and performance of the simulated system by providing real-time or near-real-time updates on the status of various entities or processes.</p>	
<p><b><i>Sink object</i></b></p>	<p>Used to represent the final destination or end point for entities in a model. Its primary function is to collect entities that have completed their processing within the simulation. Sink objects are typically placed at the end of process flows or pathways to capture and record data related to completed entities, such as arrival times, processing times, and other relevant metrics.</p>	



### 3.2.2 Neural networks integration

Simio Software offers a no-code neural network development process. This no-code process empowers developers to create NN models without having to encounter complex coding challenges. Once created, the user has the option to train and evaluate the NN model directly using the Simio Trainer feature [16].

This study utilizes Simio's recent feature, which enables the generation of training data, conducting neural network training, and implementing the trained model directly within the Simio interface. This process involves creating a feed-forward neural network directly in Simio without any coding required.

The way Simio's neural networks operate is illustrated in Figure 3.1. The process starts by acquiring information from within the simulation, which then enters a black box with the output function based on the defined feed-forward neural network model. The neural network then learns patterns from the data, making predictions or classifications based on the inputs. The neural networks can analyze the simulated outcomes to suggest optimal scheduling sequences, reducing the makespan improving operational efficiency.

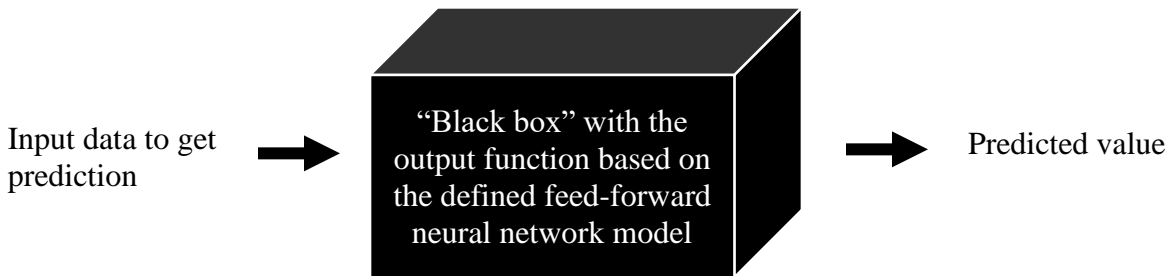


Figure 3.1 Simio NN integration process

The neural network model in Simio is a fully connected feedforward neural network regression model with a fixed number of numeric inputs and a single numeric output.

This type of neural network can have one or more hidden layers. The number of hidden layers and the number of nodes in each hidden layer are hyperparameters that can be set before training the model.

The hyperbolic tangent function (tanh) is used as the activation function for hidden layers, while the identity function (linear) is used as the activation function for the output layer.

Table 3.2 outlines the specific features that enable Simio to create and define a neural network element. The neural network is trained using Simio's built-in trainer (refer to Table 3.3 for detailed concepts) and subsequently implements the model logic. This approach utilizes Simio's capabilities to improve process efficiency and optimize the flow-shop scheduling problem.

Table 3.2 Simio concepts to create the neural network element.

<b><i>Neural Network element</i></b>	This element is used to integrate a neural network regression model into simulation logic
<b><i>Input Value Expressions</i></b>	The expressions used to get a set of input values for the associated neural network model. These expressions should always match the order of the input values provided during the training of the model.
<b><i>Save Input Triggers</i></b>	Optional event-driven triggers that will save a set of input values for the associated neural network model.

<b><i>Save Actual Triggers</i></b>	Optional event-driven triggers that will save an actual observed value for the associated neural network model.
<b><i>Actual Value Expression</i></b>	The expression used to record an actual observed value for the associated neural network model when a Save Actual Trigger occurs.

Table 3.3 Simio concepts to train the neural network.

<b><i>Number of Input Nodes</i></b>	The number of nodes in the neural network's input layer.
<b><i>Record Training Data</i></b>	Indicates whether to record training data for the neural network model.
<b><i>Maximum Training Record Limit</i></b>	Indicates the maximum number of training records that may be stored in the neural network models training data repository. The oldest records will be overwritten by new ones then the specified maximum record limit is reached.

### 3.2.3 Plan for data collection and processing

One of the advantages of using Simio software is its ability to utilize simulation for data sampling, which can then be used to train the neural network. The plan for optimization using neural networks in Simio involves running simulations and using the generated data to train the neural network. Specifically, the FSSP simulation is run 10,000 times to generate a comprehensive dataset. The purpose is to capture a wide range of possible system states and outcomes, helping the neural network to generalize better and optimize the process more effectively

During training, the neural network adjusts its internal weights and biases to minimize the difference between predicted and actual outputs. To create a more efficient trained model, 100,000 data points are collected from the nine specified input nodes. This extensive dataset ensures that the neural network has sufficient data to accurately identify the optimal routing solution, thereby optimizing the production schedule for this flow-shop problem.

Figure 3.3 shows the parameters set within the neural network configuration, specifying that 100,000 records are extracted from the simulations for training. This number is chosen to ensure that the neural network has a comprehensive dataset, which enhances the model's ability to learn and adapt to the complexities of the system, leading to more precise and reliable optimization outcomes. By using a large dataset from multiple simulation runs, the neural network can better learn and adapt to the system's intricacies, resulting in a more efficient optimization process and ensuring that the solutions generated are based on a thorough understanding of the system dynamics.

Properties: NeuralNetworkModel (Feedforward Neural Network Model)	
▲ <b>Input/Output</b>	
Number Input Nodes	9
Force Negative Outputs To Zero	False
▲ <b>Training Data Repository</b>	
Current Records Count	100000
Automatic Data Recording	True
Maximum Records Limit	100000
▲ <b>General</b>	
Name	NeuralNetworkModel
Description	
Is Trained	True
Last Update Time (UTC)	11/9/2023 10:18:05 PM

Figure 3.3 Training parameters

### 3.3 PALMER HEURISTIC INTEGRATION

Palmer Heuristic is a well-established algorithm utilized in scheduling optimization, known for its effectiveness in solving complex scheduling problems. The Palmer Heuristic uses heuristic rules and strategies to determine an optimal job processing sequence to minimize the makespan and improve operational efficiency. This study employs the Palmer Heuristic as an alternative optimization approach to measure the performance of the Simio simulation with neural networks. Therefore, a custom Python code is developed to implement the Palmer's Heuristic and determine the optimal scheduling route for this particular problem. This section outlines the steps required to implement the Palmer Heuristic for the FSSP involving three machines and four jobs. The general formula for calculating  $A_1$ , which represents the completion time for job  $j$ , is used to determine the optimal job sequence.

The completion time  $A_j$  for  $j$  using the Palmer Heuristic is calculated as follows:

$$A_j = - \sum_{i=1}^m ((3 - (2i - 1)) * p_{ij})$$

Where  $m$  is the number of machines and  $p_{ij}$  is the processing time of job  $j$  on machine  $i$ .

This formula calculates  $A_j$  by summing the product of the processing time of job  $j$  on each machine, multiplied by a specific factor determined by the Palmer Heuristic. Once  $A_j$  values are calculated for all jobs, sort the jobs in descending order based on their  $A_j$  values. This sorted order represents the optimal job sequence according to the Palmer Heuristic.

### **3.4 PERFORMANCE METRICS AND EVALUATION CRITERIA**

The makespan obtained from both Simio's neural network-based approach and the traditional Palmer Heuristic method will be used to compare and evaluate their respective performances. The comparative analysis will involve assessing the makespan values from both methods to determine their advantages and disadvantages. The key evaluation criteria will include:

- Efficiency: Evaluating which method produces a lower makespan, thus indicating a more efficient scheduling process.
- Computational Effort: Comparing the time and resources required to achieve results using both methods.
- Implementation Complexity: Assessing the ease of implementing each method, considering the need for custom coding versus using built-in simulation tools.
- Flexibility and Scalability: Analyzing how each method adapts to changes in the scheduling problem, such as varying the number of jobs or machines.

## Chapter 4: Simulation Model Development

### 4.1 DESIGN AND SETUP OF THE SIMULATION MODEL IN SIMIO

In this simulation model, Simio *source* objects were defined to generate entity objects with programmed parameters such as quantity, type, and frequency. Four *source* objects were allocated in the model, each producing one of the four *entities* created that share identical characteristics. Each *entity* represents an order and must pass through the three machines necessary for completion. The machines are simulated using *server* objects in the model, which are utilized to model the machines required for processing each order. These *server* objects represent processes with finite capacity. A *sequence table* was also employed, specifying the routes and processing times required for each *entity* to pass through each machine.

*Connector* objects were used to define the connections between different areas of the simulation through which orders travel. These *connector* objects are unidirectional since orders always follow a specific route (e.g., machine 1, machine 2, machine 3). *Dynamic labels* were placed next to the *sources* to monitor the average time in the system for each order from each *source*. *Dynamic labels* are variables used to display dynamic information about *entities*, objects, or the system state. Finally, a *sink* object was employed to denote the completion of the simulation, signifying when an order has been fulfilled. The 2D view of the simulation is shown in Figure 4.1.

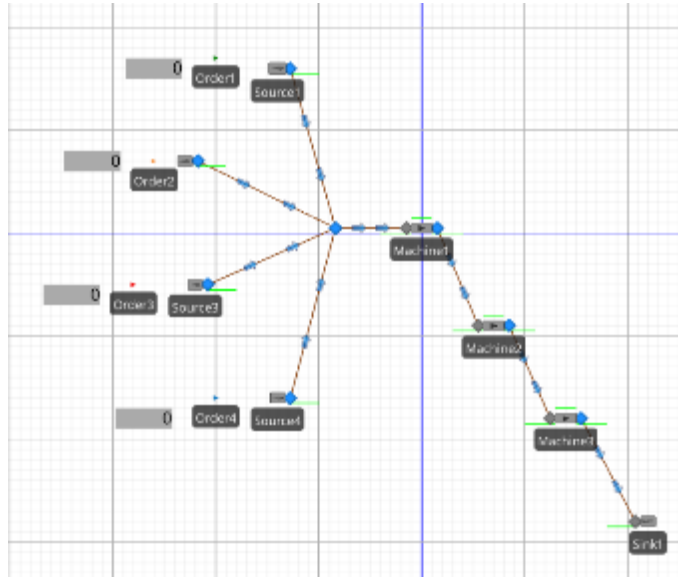


Figure 4.1 2D view for the simulation model

## 4.2 DEFINITION OF INPUT PARAMETERS AND CONSTRAINTS

In this section, the parameters of each object used to create the baseline model for this simulation in the FSSP case study will be described, defining and explaining each parameter and the reasons for these choices. It is important to highlight that all the parameters established in this section are for the creation and development of the model without optimization through neural networks.

### 4.2.1 Entities

Four entities were established in this model, each representing one of the jobs or "orders" as named in the Simio simulation. All entities in this model have identical characteristics (refer to Figure 4.2 for detailed entity properties). The free space steering behavior is defined as "direct to destination," meaning that the order will always travel through the connectors directly to the next object in the simulation, whether it is the next machine to be processed or the *sink*. The routing



logic for all entities is the same and is defined with the same processing priority, meaning all orders have equal importance in the simulation and none are prioritized over another.

At time zero of the simulation, the initial number of entities in the system is zero. Upon running the model, the four entities are simultaneously created at their respective *sources*. The key and important difference between these four orders or jobs is that each has a different processing time for each of the three machines they must pass through to be considered a finished product. All orders must follow the same path: machine 1, then machine 2, and finally machine 3.

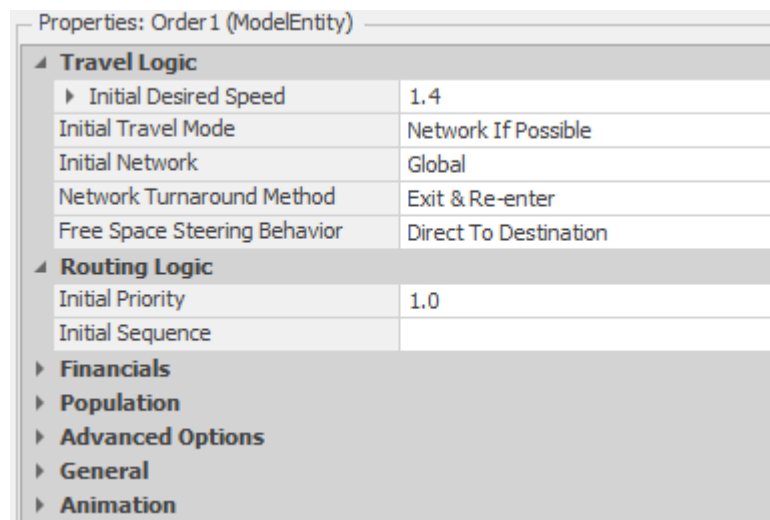


Figure 4.2 Entity properties

Additionally, the entities were configured with specific parameters to ensure accurate modeling of the production process. The *sequence table*, which details the processing times required for each entity at each machine, is presented in Figure 4.3. This table ensures that each entity's process through the system is defined and allows for control over the simulation's workflow.

In the sequence table, processing times for each machine are listed. For instance, Entity 1 might have a processing time of 6 minutes on Machine 1, 5 minutes on Machine 2, and 4 minutes

on Machine 3. Entity 2 could have different times, reflecting variability in job requirements. This setup provides a realistic depiction of the production environment and its constraints.

Order Table					
	Order Type	Mix	Machine1ProcessingTime (Minutes)	Machine2ProcessingTime (Minutes)	Machine3ProcessingTime (Minutes)
1	Order1	1	6	5	4
2	Order2	1	8	1	4
3	Order3	1	3	5	4
4	Order4	1	4	4	2
*					

Figure 4.3 Sequence table

## 4.2.2 Sources

Four sources were defined in the model, each tasked with creating one of the four entities representing the orders. The interarrival time for the four sources is set to 26 minutes, meaning the interval at which entities are created. This timing ensures that each new order is generated only after the previous four orders have been processed, thereby preventing a new entity from being delayed by an order from the previous set still being processed by the three machines.

Figure 4.4 details the properties of the four sources, all of which share the same defined properties.

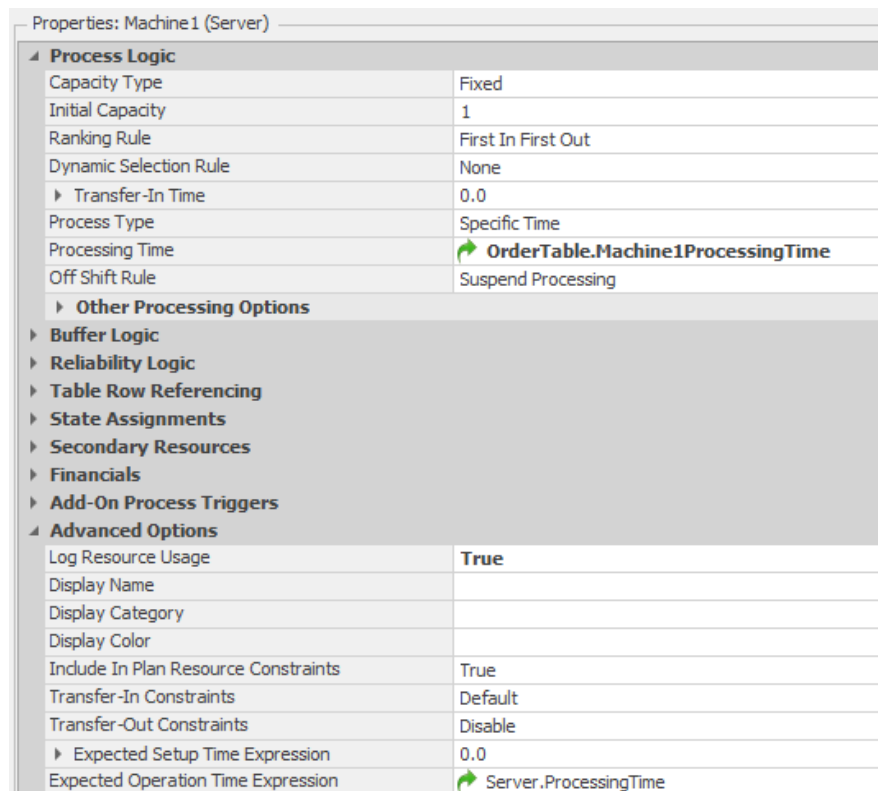
Properties: Source 1 (Source)

- Entity Arrival Logic
  - Entity Type: Order1
  - Arrival Mode: Interarrival Time
    - Time Offset: 0.0
    - Interarrival Time: 26
    - Entities Per Arrival: 1
- Stopping Conditions
- Buffer Logic
- Table Row Referencing
  - Before Creating Entities
    - Action Type: Reference Existing Row
    - Table Name: OrderTable
    - Row Number: 1
  - On Created Entity
- State Assignments
- Financials
- Add-On Process Triggers
- Advanced Options
- General

Figure 4.4 Source properties

### 4.2.3 Servers

Each server represents a machine that processes the orders with the times specified in Figure 4.3. All machines share the same properties. It is important to note that the ranking rule for these servers is set to first-in, first-out (FIFO), meaning the machines process entities in the order they arrive without prioritizing any order over another. Additionally, the log resource usage feature is activated, which allows for obtaining the visual schedule at the end of the simulation. Figure 4.5 shows the properties of the servers.



Properties: Machine 1 (Server)	
<b>Process Logic</b>	
Capacity Type	Fixed
Initial Capacity	1
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Process Type	Specific Time
Processing Time	OrderTable.Machine1ProcessingTime
Off Shift Rule	Suspend Processing
<b>Other Processing Options</b>	
<b>Buffer Logic</b>	
<b>Reliability Logic</b>	
<b>Table Row Referencing</b>	
<b>State Assignments</b>	
<b>Secondary Resources</b>	
<b>Financials</b>	
<b>Add-On Process Triggers</b>	
<b>Advanced Options</b>	
Log Resource Usage	True
Display Name	
Display Category	
Display Color	
Include In Plan Resource Constraints	True
Transfer-In Constraints	Default
Transfer-Out Constraints	Disable
Expected Setup Time Expression	0.0
Expected Operation Time Expression	Server.ProcessingTime

Figure 4.5 Server properties

## 4.3 TRAINING AND IMPLEMENTATION OF THE NEURAL NETWORK MODEL

### 4.3.1 Setting the neural network properties

Implementing a neural network at the input node of the server allowed the machines to determine which order to process first to achieve an optimal makespan and complete all four

orders, resulting in informed routing decision. In this case study, it was assumed that the predicted makespan depended only on the current load at each of the servers. Figure 4.6 shows the properties of the neural network element created.

Properties: NeuralNetwork (Neural Network Element)	
<b>Basic Logic</b>	
Neural Network Model Name	NeuralNetworkModel
Input Value Expressions	9 Rows
Untrained Predicted Value Expression	Random.Uniform(0,1)
<b>Training Data Recording</b>	
Save Inputs Triggers	1 Row
Save Actual Triggers	1 Row
Actual Value Expression	TimeNow - ModelEntity.TimeCreated
Training Key Expression	

Figure 4.6 Neural network properties

A neural network element was created in this simulation model and connected to other elements to receive input data and send output data. The *Input Value Expressions* used to train the neural network are shown in Figure 4.7. These parameters include:

- Associated Station Load: This is the sum of the number of entities currently routed to that server, the number of entities currently waiting at the server, and the number of entities currently being processed at the server. This parameter helps identify potential bottlenecks in the system.
- Resource State: Indicates the status of servers, whether they are busy, available, or in another state, assisting in resource allocation management.
- Schedule Utilization: Indicates how efficiently the planned schedules are being utilized in the model.

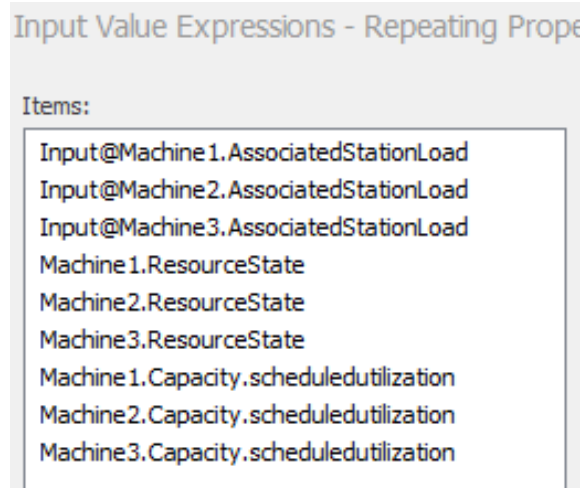


Figure 4.7 Input value expressions

*Save Inputs Triggers* determines when to start collecting training data. In this study, the chosen point for collecting training data was when the orders entered the system. To ensure that all four orders had been created by the time of data collection, transfer node 1 was selected as the save input trigger, see Figure 4.8.

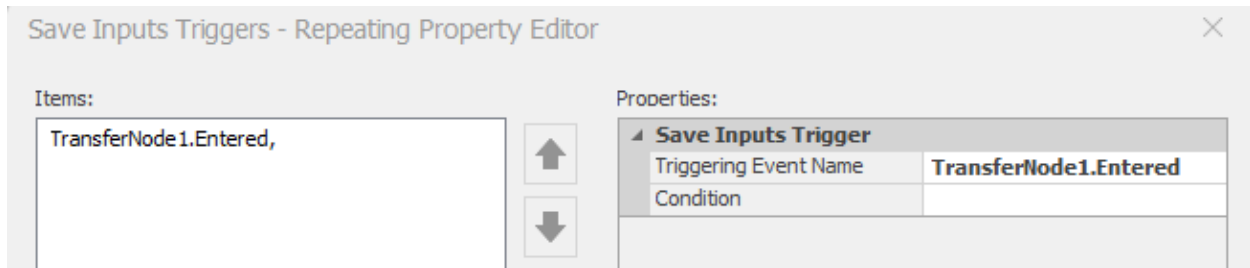


Figure 4.8 Save inputs trigger

*Save Actual Trigger* determines when to stop collecting training data. This trigger was activated when the orders finished being processed, specifically when machine 3 had completed the processing of the orders (see Figure 4.9).

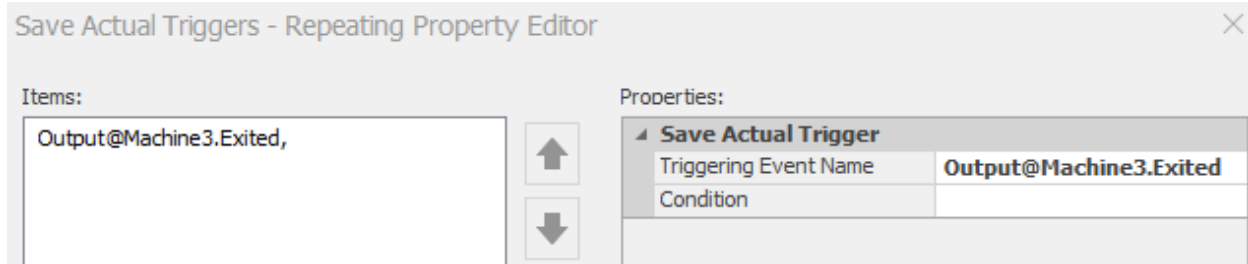


Figure 4.9 Save actual trigger

The *Actual Value Expression* calculates the current time minus the time when the order was created in the model. This helps determine the processing time and efficiency of the system.

### 4.3.2 Collecting data for training

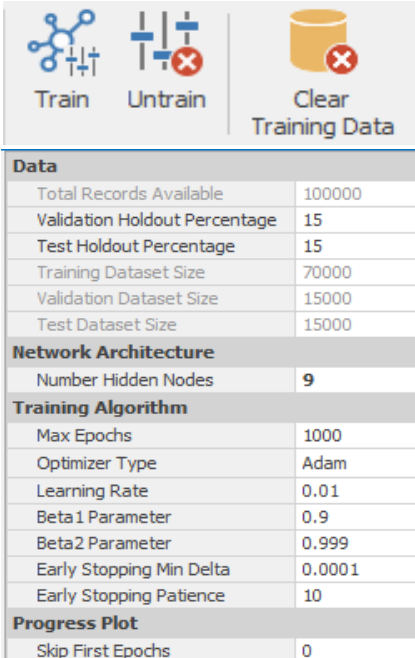
Once the parameters and properties of the neural network were established, the next step was to run the simulation to collect the training data. Simio automatically gathered the specified number of records, set to 100,000 data points for this study. As mentioned in section 3.2.3, "Plan for Data Collection and Processing," these data points were obtained from the nine input nodes. Figure 4.10 illustrates an example of the data collected by Simio during the simulation.

Time Stamp (UTC)	Actual Value	Input Values
11/9/2023 9:23:04 PM	0.4333	2.0000, 0.0000, 1.0000, 0.0000, 0.0000, 1.0000, 80.7692, 57.6923, 46.1538
11/9/2023 9:23:04 PM	0.4667	3.0000, 0.0000, 1.0000, 0.0000, 0.0000, 1.0000, 80.7692, 57.6923, 46.1538
11/9/2023 9:23:04 PM	0.2500	0.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 50.0000
11/9/2023 9:23:04 PM	0.3167	1.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 50.0000
11/9/2023 9:23:04 PM	0.4333	2.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 50.0000
11/9/2023 9:23:04 PM	0.4667	3.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 50.0000
11/9/2023 9:23:04 PM	0.2500	0.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 51.2821
11/9/2023 9:23:04 PM	0.3167	1.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 51.2821
11/9/2023 9:23:04 PM	0.4333	2.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 51.2821
11/9/2023 9:23:04 PM	0.4667	3.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 51.2821
11/9/2023 9:23:04 PM	0.2500	0.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 51.9231
11/9/2023 9:23:04 PM	0.3167	1.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 51.9231
11/9/2023 9:23:04 PM	0.4333	2.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 51.9231
11/9/2023 9:23:04 PM	0.4667	3.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 51.9231
11/9/2023 9:23:04 PM	0.2500	0.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 52.3077
11/9/2023 9:23:04 PM	0.3167	1.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 52.3077
11/9/2023 9:23:04 PM	0.4333	2.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 52.3077
11/9/2023 9:23:04 PM	0.4667	3.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 52.3077
11/9/2023 9:23:04 PM	0.2500	0.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 52.5641
11/9/2023 9:23:04 PM	0.3167	1.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 52.5641
11/9/2023 9:23:04 PM	0.4333	2.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 52.5641
11/9/2023 9:23:04 PM	0.4667	3.0000, 1.0000, 1.0000, 0.0000, 1.0000, 1.0000, 80.7692, 57.6923, 52.5641

Figure 4.10 Training data

### 4.3.3 Training of the neural network

To train the neural network, the collected data is used by selecting the training option in the Simio interface. Figure 4.11 shows the neural network trainer window, detailing the specified properties for training the neural network in this case study. It is important to note that the number of hidden nodes equals the number of input value expressions.



The screenshot displays the Simio Neural Network Trainer interface. At the top, there are three buttons: 'Train' (with a blue neural network icon), 'Untrain' (with a blue neural network icon and a red 'X'), and 'Clear Training Data' (with a yellow database cylinder icon and a red 'X'). Below these buttons is a table of training properties organized into four sections: Data, Network Architecture, Training Algorithm, and Progress Plot.

Data	
Total Records Available	100000
Validation Holdout Percentage	15
Test Holdout Percentage	15
Training Dataset Size	70000
Validation Dataset Size	15000
Test Dataset Size	15000

Network Architecture	
Number Hidden Nodes	9

Training Algorithm	
Max Epochs	1000
Optimizer Type	Adam
Learning Rate	0.01
Beta1 Parameter	0.9
Beta2 Parameter	0.999
Early Stopping Min Delta	0.0001
Early Stopping Patience	10

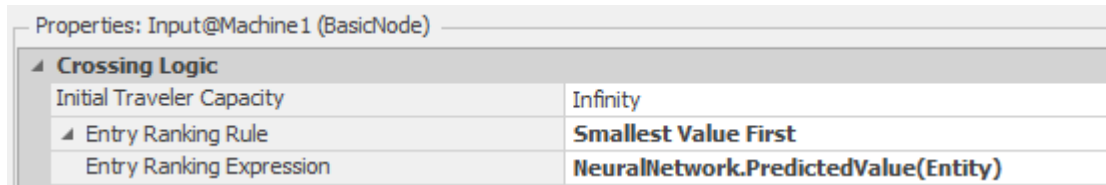
Progress Plot	
Skip First Epochs	0

Figure 4.11 NN trainer properties

### 4.3.4 Implementation of neural network

After the neural network parameters were determined and adequately trained, the trained neural network was implemented by adjusting the properties of the input node for the servers (see figure 4.12). The property *entry ranking rule* of the servers was set to "smallest value first" to be determined with the data previously obtained from the established neural network, and the entry ranking expression was defined as **NeuralNetwork.predictedvalue(entity)**. Instead of processing orders as they arrive (i.e., first in, first out), the model employs the neural network's insights to

make informed decisions on the optimal sequence for order processing, aiming to achieve the production schedule with the minimum possible makespan.



Properties: Input@Machine1 (BasicNode)	
<b>▲ Crossing Logic</b>	
Initial Traveler Capacity	Infinity
▲ Entry Ranking Rule	<b>Smallest Value First</b>
Entry Ranking Expression	<b>NeuralNetwork.PredictedValue(Entity)</b>

Figure 4.12 NN implementation

#### 4.4 DEVELOPMENT OF THE PALMER HEURISTIC ALGORITHM IN PYTHON

Palmer Heuristics uses heuristic rules and strategies to determine an optimal job processing sequence to minimize the makespan and improve operational efficiency. This study employed Palmer Heuristics as an alternative optimization approach to measure the performance of Simio simulation with neural networks. To achieve this, a custom Python code was developed to implement Palmer's Heuristic and determine the optimal scheduling route for this specific problem. The complete Python code can be found in Appendix A1.1.



## Chapter 5: Results and Conclusions

### 5.1 PERFORMANCE OF THE SIMIO NEURAL NETWORK OPTIMIZATION

To assess Simio's capacity for optimizing scheduling problems using neural networks, a simulation lasting 10 hours was conducted without the application of neural networks. The obtained results are detailed in Table 5.1. It is notable that without parameter modification, the model processes orders in a "first in, first out" manner, indicating that the model does not make decisions regarding the sequence of order processing. Consequently, as orders arrive at the server, the machine processes them, leading to idle times and a suboptimal schedule. Consequently, the orders were processed in the sequence of 1, 2, 3, and 4, resulting in the schedule depicted in Figure 5.1.

It's important to point out that order four has an average time in the system of 0.466 hours, approximately 28 minutes longer than the other three orders. This is because order four is processed last and must wait for the previous orders to finish before it can be processed. Consequently, the makespan derived from the simulation, without any optimization to complete all four orders, is 0.466 hours.

Table 5.1 Average time in system (hours)

Order	Average time in system (Hours)
1	0.249
2	0.316
3	0.433
4	0.466

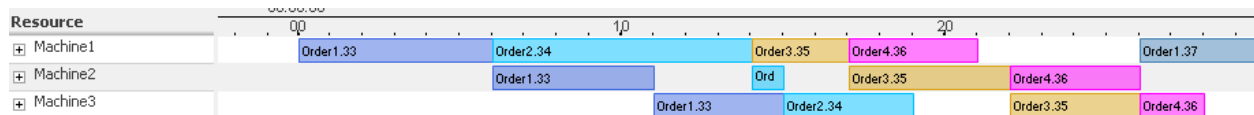


Figure 5.1 Schedule without optimization.

Upon integration of the neural network, the makespan was significantly reduced to 26 minutes, with the order processing revised to job3-job4-job1-job2. This outcome underscores the effectiveness of utilizing neural networks to enhance scheduling decisions and streamline operational efficiency within the flow-shop environment. Furthermore, downtime has been notably reduced through the neural network optimization process, as evidenced by the schedule presented in Figure 5.2. Consequently, both the average time in the system and the makespan have been decreased.

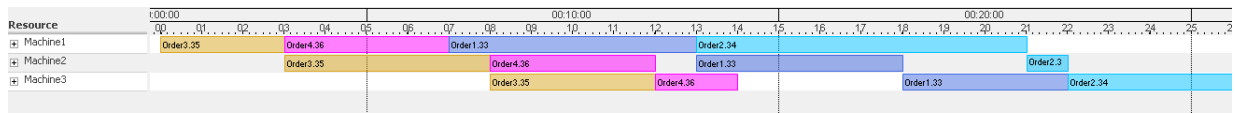


Figure 5.2 Schedule optimized by NN

## 5.2 RESULTS OF THE PALMER HEURISTIC ALGORITHM

The optimization results obtained through Palmer Heuristics are the same as those achieved with the Simio software. The optimal job sequence identified using Palmer Heuristics was job3, job1, job4, and job2, resulting in a total makespan of 26 minutes for processing all four orders. Figure 5.3 shows the schedule obtained through Palmer Heuristics. This comparison between Simio simulation with neural networks and Palmer Heuristics emphasizes the effectiveness of both methods in optimizing flow-shop.

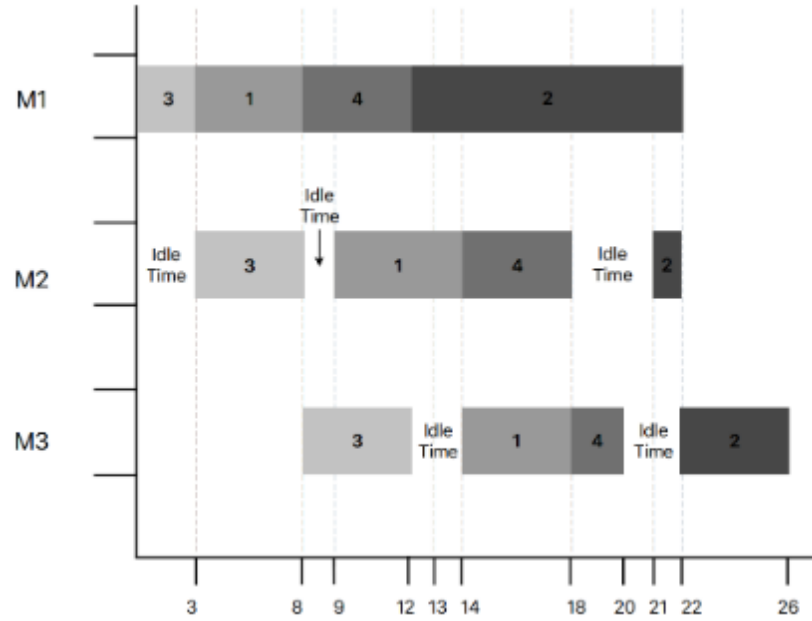


Figure 5.3 Schedule optimized by Palmer heuristics.

### 5.3 CONCLUSION

In conclusion, the implementation of both methodologies produced remarkably similar results. Both methods successfully optimized the flow-shop scheduling problem by providing the optimal routing sequence. This similarity in results highlights the effectiveness of the neural network capabilities within Simio software. Moreover, the results confirm that Simio with neural networks effectively optimizes scheduling processes and highlights its ease of implementation. The integrated approach within Simio offers a user-friendly platform that facilitates the implementation of neural networks for scheduling optimization. In contrast, the complexity associated with coding and implementing a heuristic method, as demonstrated by Palms Heuristic in Python, adds a layer of complexity to the process.

This comparison emphasizes the practical advantages of leveraging Simio with neural networks as a powerful tool for flow-shop scheduling optimization. The user-friendly nature of Simio, combined with the efficiency of neural networks, positions it as a persuasive solution for manufacturing systems seeking to improve operational efficiency. The findings suggest that Simio's integrated neural network capabilities provide a viable alternative to traditional heuristic

methods, offering a more accessible and efficient avenue for achieving optimal scheduling outcomes.

The limitations of this study include the specific context of the flow-shop scheduling problem and the predefined parameters within the simulation model. Future work could address these limitations by testing the methodologies in more diverse and dynamic settings.

## References

1. MacCarthy, B. L., & Liu, J. (1993). Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, 31(1), 59–79.
2. Linn, R., & Zhang, W. (1999). Hybrid flow shop scheduling: a survey. *Computers & Industrial Engineering*, 37(1–2), 57–61.
3. Hall, N. G., & Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3), 510–525.
4. Ding, J.-Y., Song, S., Gupta, J. N. D., Wang, C., Zhang, R., & Wu, C. (2016). New block properties for flowshop scheduling with blocking and their application in an iterated greedy algorithm. *International Journal of Production Research*, 54(16), 4759-4772.
5. Johnson, S. M. (1954). Optimal two- and three-stage production schedules with set-up times included. *Naval Research Logistics Quarterly*, 1, 61-68.
6. Jackson, J. R. (1955). Scheduling a production line to minimize maximum tardiness (Research Report No. 43). Management Science Research Project, University of California at Los Angeles.
7. Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3, 59-66.
8. Lenstra, J. K., Rinnooy Kan, A. H. G., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1, 343-302.
9. Pinedo, M. L. (2009). *Planning and Scheduling in Manufacturing and Services* (2nd ed. 2009.). Springer New York. <https://doi.org/10.1007/978-1-4419-0910-7>

10. Saaty, T. L. (1982). Decision making for leaders : the analytical hierarchy process for decisions in a complex world. Lifetime Learning Publications.
11. Thomas, A., & Charpentier, P. (2005). Reducing simulation models for scheduling manufacturing facilities. *European Journal of Operational Research*, 161(1), 111–125.
12. Lejmi, T., & Sabuncuoglu, I. (2002). Effect of load, processing time and due date variation on the effectiveness of scheduling rules. *International Journal of Production Research*, 40(4), 945–974.
13. Negahban, A., & Smith, J. S. (2014). Simulation for manufacturing system design and operation: Literature review and analysis. *Journal of Manufacturing Systems*, 33(2), 241–261.
14. Kostas Metaxiotis & John Psarras (2003). Neural networks in production scheduling: Intelligent solutions and future promises, *Applied Artificial Intelligence*, 17:4, 361-373, DOI: 10.1080/713827140
15. Arbib, M. (1998). *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA: The MIT Press.
16. Simio LLC. (2024). *Overview of neural networks with simulation modeling*. Simio. <https://www.simio.com/applications/industry-40/Overview-of-Neural-Networks-with-Simulation-Modeling.php>

## Appendix

### A1.1 PYTHON CODE FOR PALMER'S HEURISTIC IMPLEMENTATION

```
import time
startTime = time.time()

job = [1,2,3,4]
# processing times for each machine

m1 = [6,8,3,4]

m2 = [5,1,5,4]

m3 = [4,4,4,2]

# A1 calculations

Indx_m1 = m1[0]
Indx_m2 = m2[0]
Indx_m3 = m3[0]

A0 = (3-((2)*(job[0])-1))*(Indx_m1)
A1 = (3-((2)*(job[1])-1))*(Indx_m2)
A2 = (3-((2)*(job[2])-1))*(Indx_m3)

print("A1 values: ", A0,A1,A2)

A1_Sum = ((A0 + A1 + A2)*(-1))

print("A1 Sum is: ", A1_Sum)

JOB1 = A1_Sum

#A2 calculations

Indx_m1_A2 = m1[1]
Indx_m2_A2 = m2[1]
Indx_m3_A2 = m3[1]

A2_0 = (3-((2)*(job[0])-1))*(Indx_m1_A2)
A2_1 = (3-((2)*(job[1])-1))*(Indx_m2_A2)
A2_2 = (3-((2)*(job[2])-1))*(Indx_m3_A2)

print("A2 values: ", A2_0,A2_1,A2_2)

A2_Sum = ((A2_0 + A2_1 + A2_2)*(-1))

print("A2 Sum is: ", A2_Sum)

JOB2 = A2_Sum
```

```

#A3 calculations

Indx_m1_A3 = m1[2]
Indx_m2_A3 = m2[2]
Indx_m3_A3 = m3[2]

A3_0 = (3-((2)*(job[0])-1))*(Indx_m1_A3)
A3_1 = (3-((2)*(job[1])-1))*(Indx_m2_A3)
A3_2 = (3-((2)*(job[2])-1))*(Indx_m3_A3)

print("A3 values: ", A3_0,A3_1,A3_2)

A3_Sum = ((A3_0 + A3_1 + A3_2)*(-1))

print("A3 Sum is: ", A3_Sum)

JOB3 = A3_Sum

#A4 calculations

Indx_m1_A4 = m1[3]
Indx_m2_A4 = m2[3]
Indx_m3_A4 = m3[3]

A4_0 = (3-((2)*(job[0])-1))*(Indx_m1_A4)
A4_1 = (3-((2)*(job[1])-1))*(Indx_m2_A4)
A4_2 = (3-((2)*(job[2])-1))*(Indx_m3_A4)

print("A4 values: ", A4_0,A4_1,A4_2)

A4_Sum = ((A4_0 + A4_1 + A4_2)*(-1))

print("A4 Sum is: ", A4_Sum)

JOB4 = A4_Sum

#Sorting The JOB sequence

JOBS = [JOB1, JOB2, JOB3, JOB4]

JOBS.sort(reverse = True)

#print(JOBS)

#allows to sort values in dictionarys

mydict = {'job1': JOB1, 'job2': JOB2, 'job3': JOB3, 'job4': JOB4}

sorted_values = sorted(mydict.items(), key =lambda x:x[1], reverse = True)

converted_dict = dict(sorted_values)

```



```
#print(converted_dict)

#values arranged in decending order

print("The Job sequence using palmers heuristic is:" )
for keys, values in converted_dict.items():
    print(keys, values)

executionTime = (time.time() - startTime)
print('Execution time in seconds: ' + str(executionTime))
```

## **Vita**

Jesús Ricardo Herrera Garfio was born in Ciudad Juárez, Chihuahua, México. He obtained his Bachelor's degree in Industrial Engineering from the Universidad Autónoma de Ciudad Juárez in 2020. He then pursued a Master's in Industrial Engineering at The University of Texas at El Paso, during this time, he became a Research Assistant in the Department of Industrial, Manufacturing, and Systems Engineering, while also working as a Quality Engineer intern at Cummins, a global leader in engine technology. His research focuses on Flow-Shop Scheduling Optimization through Discrete-Event Simulation and Neural Networks.