MANUFACTURING PLANNING AND OPERATIONS OPTIMISATION FOR MASS CUSTOMISATION MANUFACTURING USING COMPUTATIONAL INTELLIGENCE

Louwrens Johannes Butler

Submitted in fulfilment of the academic requirements for the degree of PhD in the School of Engineering, University of KwaZulu-Natal, Durban

December 2015

Supervisor: Prof Glen Bright
LIBRARY COPY

COLLEGE OF AGRICULTURE, SCIENCE AND ENGINEERING

DECLARATION 1 – PLAGIARISM

I, **Louwrens Johannes Butler** declare that:

1. The research reported in this thesis, except where otherwise indicated, is my original research;

2. This thesis has not been submitted for any degree or examination at any other university;

3. This thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons;

4. This thesis does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

    a. Their words have been re-written but the general information attributed to them has been referenced

    b. Where their exact words have been used, then their writing has been placed in italics and inside quotation marks, and referenced

5. This thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the thesis and in the References sections.

Signed: ……………………… JButler

COLLEGE OF AGRICULTURE, SCIENCE AND ENGINEERING

DECLARATION 2 – PUBLICATIONS

DETAILS OF CONTRIBUTION TO PUBLICATIONS that form part and/or include research presented in this thesis are as follows:

Publication 1

L. J. Butler and G. Bright, "Computational Intelligence for Advanced Manufacturing System Management: A Review," *International Journal of Intelligent Systems, Technologies, and Applications*, vol. 13, no. 4, p. 258, 2014.

Acknowledgements

Firstly, I would like to thank my supervisor, Professor Glen Bright, for providing the environment to further my education in Engineering.

I would also like to thank my wife Morag Butler, for her frequent pep talks, and unwavering support and confidence in me. I would never have been able to complete this work without her.

Lastly, I would like to thank my parents for their support and understanding throughout my academic career. They instilled in me an inquisitive nature which has brought me to this point in my life.

Abstract

This study determined whether an Advanced Manufacturing System could be optimised, more effectively than by traditional methods, using new and novel computational intelligence techniques. An Advanced Manufacturing System can be described as highly automated and highly complex systems that strive for global competitiveness. In the context of this study, these systems aim to compete in a Mass Customisation Manufacturing market. Traditional optimisation methods refer to methods based on mathematical models, experience, or industry best practice. Computational Intelligence refers to computational methods inspired by natural systems and processes. This includes, but is not limited to, evolutionary intelligence, Artificial Neural Networks, swarm intelligence, and fuzzy systems.

This study investigated the optimisation of the manufacturing system from both a planning and an operations perspective. Research was carried out to identify Computational Intelligence paradigms and algorithms for Advanced Manufacturing System planning and operations optimisation. Static and dynamic simulation models of an Advanced Manufacturing System, for the respective perspectives, have been developed in order to simulate a manufacturing system designed to produce a hypothetical range of customisable men's wristwatches on a mass scale at a competitive cost.

A new Biogeography-Based Optimisation algorithm was developed to optimise an aggregate production plan using static simulation models. This algorithm was implemented to find the lowest production cost for the wristwatch production system case study. This algorithm produced a lower cost plan than a Simulated Annealing algorithm with a lower impact on workforce. A new Distributed Dynamic Selection Rule Strategy was developed for optimising production scheduling using dynamic simulation models. This new strategy was inspired by the Harmony Search principle and was based on traditional selection rules for scheduling. This strategy was able to produce statistically significantly lower average order lead times than three out of four traditional selection rules tested.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

This chapter provides research background and introduces the problem spaces and research approaches taken in this research study. Research outputs produced during this study are listed, and an outline of this thesis is provided.

1.1 Research background

A large proportion of the global manufacturing sector is striving to achieve Mass Customisation Manufacturing [1]. The driver for this is the demand from the markets for unique custom products at affordable prices and reasonable lead times [1]. However, producing custom products at the same cost as high volume low variety manufacturing are conflicting objectives. Many researchers have devoted their attention to this problem from many perspectives. However, two main research areas have been identified as being fundamental and sufficiently different to focus on separately in an attempt to improve manufacturing system performance in order to approach full Mass Customisation Manufacturing - production planning and production scheduling. Computational Intelligence has emerged as a popular instrument to apply in the pursuit of optimising manufacturing systems for both maximising profitability as well as achieving Mass Customisation Manufacturing for maximum market share.

An important advantage of the Computational Intelligence paradigm, in this context, is that the techniques involved produce sets of optimal and near-optimal solutions, from which the most acceptable solution can be selected. This way the decision makers still have make the final decision. The nature of most Computational Intelligence algorithms make them robust and adaptable to variations in system parameters and characteristics. For this reason they are well suited to applications in environments based on paradigms that make use of Computer-Integrated Manufacturing infrastructure, and are designed to be flexible. The application of Computational Intelligence optimisation techniques will assist modern manufacturing systems to be more responsive and adaptable to variations in product design due to changing customer demands. This is an important consideration for existing manufacturing enterprises in order to maintain competitive production rates.

1.2 Research overview

The research question posed was: Can an Advanced Manufacturing System, striving for Mass Customisation Manufacturing, be optimised more effectively than by traditional methods, using novel Computational Intelligence principles? An Advanced Manufacturing System can be described as systems that strive for economic competitiveness in Mass Customisation

Manufacturing through high levels of automation. Traditional optimisation methods refer to methods based on industry best practice.

1.2.1 Aim and objectives

The aim of this study was to research, develop, and test novel Computational Intelligence-based optimisation methods built on simulation models for Advanced Manufacturing Systems, at the planning and scheduling levels. The objectives identified at the outset of this study were:

1. Research the state of manufacturing strategies and manufacturing system types for compatibility with the research approach and the state of Computational Intelligence as a technology.

2. Develop manufacturing system models for implementation and testing of planning and scheduling optimisation techniques.

3. Research and develop a Computational Intelligence-based optimisation technique for optimising production planning activities within the manufacturing system type identified in research.

4. Research and develop a Computational Intelligence-based optimisation technique for optimising production scheduling within the manufacturing system type identified in research.

5. Deploy Computational Intelligence optimisation techniques in computer simulations of a case study production system to evaluate the performance of the techniques against traditional planning and scheduling methods, by analysing and interpreting computer simulation results, and draw conclusions.

1.2.2 Method

The methods used in this study relied on the use of simulation modelling for testing and experimentation. Static and dynamic simulation models were created based on a hypothetical product family-based product range designed for Mass Customisation Manufacturing. These models were developed using a classical simulation model development process which included verification and validation of the model behaviour. Simulation models were used due to the fact that no real-world system was available for this study. Two static system models were developed for investigating the production planning problem, and a dynamic simulation model was developed for investigating the production scheduling problem.

Analysis of the results achieved by the implementation of the newly developed optimisation methods for the planning and scheduling problems, was carried out by comparison. An additional optimisation algorithm was written based on an established Computational Intelligence paradigm as found in literature, in order to measure the performance of the newly developed planning optimisation algorithm. The performance of the newly developed scheduling strategy was compared to the performance of traditional scheduling rules commonly used for the same application as found in literature.

1.2.3 Research contribution

The contribution of this research study was threefold. The first was the development of a dynamic manufacturing system simulation model designed for investigating Mass Customisation Manufacturing through single unit order handling. In this study this model was used as the basis for the development of a new distributed dynamic scheduling strategy.

The second contribution made in this study was the development of a new optimisation algorithm for Aggregate Production Planning. This algorithm was based on the principles of Biogeography-Based Optimisation, which has never been used for this specific application. This algorithm was able to produce lower cost aggregate production plans than traditional planning methods.

The third contribution made in this study was the development of a new distributed dynamic scheduling strategy to address the scheduling problem caused by the pursuit of Mass Customisation Manufacturing through single unit order processing by a flexible flow shop manufacturing system. The implementation of this strategy achieved lower average order lead times than traditional scheduling methods.

1.3 Thesis outline

Chapter 2 gave more detailed research background and reviewed literature relevant to the aims and objectives of this research study. Chapter 3 developed three simulation models, two static and one dynamic, which formed the basis of further development of the research toward improving system performance from the perspectives of production planning and scheduling. Chapter 4 documented the development of a new optimisation algorithm for Aggregate Production Planning based on the two static simulation models of Chapter 3. Chapter 5 presented the development of a new approach for dynamic manufacturing scheduling applied to the dynamic simulation model of Chapter 3. Finally, Chapter 6 summarised the findings and conclusions drawn during this research study, related these back to the objectives stated here, and offered topics for future work

.

Chapter 2 Research Context and Literature Review

This chapter highlights past research into the facilitation of Mass Customisation Manufacturing (MCM), and the application of Computational Intelligence (CI) in manufacturing system optimisation research. The purpose of this chapter is to review and interpret the relevant literature to set the context and direction of this research study.

2.1 Mass Customisation Manufacturing

This section presents background information and literature focussed on the facilitation of MCM from an economic and a practical perspective. Economically producing custom products on a mass scale is analysed from a holistic perspective and the suitability of manufacturing strategies discussed. The evolution of a manufacturing system from the perspective of a product platform is also discussed and put into context for the purposes of this study.

2.1.1 Definition of Mass Customisation

Mass Customisation (MC) was first defined by Joseph Pine in 1993 [2] as the production of individually customised goods through the use of flexible and highly responsive manufacturing systems at a cost near that of mass produced goods. This is a broad definition which can take many forms at a practical level.

In the pursuit of MC the most popular approaches have been based on Delayed Product Differentiation (DPD), also known as process postponement [3]. Many different implementations of DPD exist. Each is characterised by the point in production at which differentiation occurs, from engineer-to-order, where each instance of the product is designed to the customer's requirements, to package-to-order, where differentiation only occurs at the packaging stage.

In the context of this study, mass customisation is viewed as differentiation of the product during production, in other words differentiation is restricted to variants in the product family and is achieved while the product is in the manufacturing system. This is analogous to the mode of MC described as fixed resource design-to-order MC, according to MacCarthy, Brabazon, & Bramham [4]. This selection was made because it is believed that this is the best avenue for achieving MCM effectively.

2.1.2 Considerations in the pursuit of Mass Customisation

Fixed resource design-to-order MC is dependent on the inevitability that there will be a certain lead time involved in the transaction, as opposed to selecting a product off the shelf. This is due to the fact that the product only comes into being after the customer has set their specifications.

As a matter of opinion, a large proportion of customer satisfaction should be linked to the lead time for delivering the product to the customer. The other consequence of this situation is that, although the interface between manufacturer and customer can be made more direct through direct sales by the manufacturer to the end user, the experience of shopping for, and purchasing a product becomes more indirect with the wider implementation of online product configurators as has been found by Fogliatto et al. [1]. That is, unless demonstration products have been made available for the customer to experience before making a purchase. The value added by customisation and short lead time must outweigh the value lost by distancing the customer from experiencing the product before making a purchase.

The fundamental aim for manufacturers in selling their goods is to maximise the value of the product as perceived by the end user, irrespective of whether the manufacturer produces parts or subassemblies of the end product, or the end product itself. It is vital for the manufacturer to focus on the factors that contribute to customer satisfaction. Research has shown that too wide a variety of feature options in a product family can have a detrimental effect on the supply chain [5]. It is important for a manufacturer to establish an optimal range of product variants and in turn feature variety. However, defining the best variety level is still a challenge for many manufacturing enterprises [6]. In this study, the assumption is made that the product variety on offer is designed in such a way that it effectively addresses the variation in customer requirements.

Two more factors that traditionally relate to customer satisfaction are perceived added value through quality and functionality [7]. Mass Customisation Manufacturing attempts to address the latter by tailoring each unit to the preferences of the customer. Quality in the sense of defect detection and avoidance has a wide and active field of research associated with it, and as such will not be addressed in this study. It is assumed in this research study that product and process design are performed such that the quality of the end product is sufficient, and does not affect customer satisfaction.

2.1.3 Systems for achieving Mass Customisation

Important systems and concepts have been developed in research into achieving MC. Fundamentally, these are aimed at increasing the flexibility of the physical manufacturing system, and increasing the flexibility of the system governing the organisation of product range information. These are discussed in this section.

2.1.3.1 Product information management

Primary driving forces for a manufacturing system flexibility, in the pursuit of economic competitiveness, include the evolution of product variety and demand levels, along with advances in machine technology, which are considered as external forces [8]. It is proposed that the manufacturing system needs to evolve with the products, both in terms of the product variety and demand levels within the product variety to maintain optimal operation. The timescale of manufacturing system evolution can be coupled to the timescale of product evolution and, for maximum competitiveness, changes in the manufacturing system must track changes in product demand and variety [6]. This study investigates the preparation for and reaction to this behaviour.

In order to rationalise and keep track of the changes to the product variety, the concept of product family architecture is implemented through the use of product platforms during the development of the product range [9]. Product platforms are in turn linked to process platforms in the manufacturing system through the development of a Bill of Materials and Operations (BOMO) [10]. This approach is also an effective method for developing flexible BOMOs, also referred to as Generic Bills of Materials and Operations (GBOMOs), for facilitating scheduling of the same operations to be performed by different workstations depending on the load distribution and demand, in a Flexible Manufacturing System (FMS). This is a valuable characteristic which has been incorporated in this research study, in the case study.

2.1.3.2 Production system architecture

Manufacturing system configuration approaches showing the most potential in facilitating MC are FMSs, and Reconfigurable Manufacturing Systems (RMSs) [8]. Dedicated Manufacturing Systems are not considered due to the fact that they do not meet the requirements for volume as well as variety imposed by MC [11]. Flexible Manufacturing Systems possess the necessary flexibility and agility to facilitate MCM, but these tend to possess more capability and/or capacity than absolutely necessary, which implies that the capital investment may be higher than necessary [8]. Proper planning could limit this over-capitalisation.

Reconfigurable Manufacturing Systems are defined to be tailored to exact processing and capacity requirements through modularity and scalability of workstations [8]. This is aimed at cutting cost at the setup of the manufacturing facility. However, whenever additional modules for functionality or capacity are required these will still need to be purchased, and whenever modules are required to be removed for similar reasons these will need to be stored at the manufacturer's cost. There would also be considerable time and cost involved in the physical reconfiguration of such a system since workstation modules may be large and cumbersome.

An early definition of the Flexible Manufacturing System concept was suggested by J.A. Collins [12]: "*FMS combines the existing technology of NC manufacturing, automated material handling, and computer hardware and software to create an integrated system for the automatic random processing of palletized parts across various work stations in the system.*".

Assuming this definition of FMS, it can be argued that a manufacturing system can have characteristics of both FMS and RMS approaches integrated. Workstations can be modular to facilitate reconfiguration, while multiple workstations can be configured to perform similar operations if product demand requires this. This variation would only be necessary in the case where demand for a certain operation becomes inordinately high compared to the demand for the other operations involved in production, and can only be facilitated if the necessary base workstations are available for reconfiguration. To facilitate this, capabilities for other operations based on the same workstation could be reduced to increase the capability for the operation with increased demand. In this research study the basic manufacturing system configuration approach assumed is FMS, but may include an element of reconfigurability.

2.2 Computational Intelligence

Computational Intelligence has an active field of research in many different areas of application including medicine, financial analysis, and ecology [13]–[15]. Although traditional CI methods such as Genetic Algorithms (GAs), fuzzy systems, and Artificial Neural Networks (ANNs) have been applied in manufacturing systems research, they have yet to be fully accepted by industry [16]. The definition of CI adopted in this research study can be stated as: Methods of problem solving using computational technology incorporating intelligence derived from natural systems and phenomena. This definition aligns well with the definition held by researchers in the pure CI field [17], [18], as well as researchers and champions of applied CI [19]. The following subsections set the context of CI in manufacturing system research and review recent relevant developments.

2.2.1 Computational Intelligence in context

The concept of CI is viewed as a development and extension of the concept of Artificial Intelligence (AI), as such this may include topics that are regarded as AI such as ANNs, evolutionary computation, and swarm intelligence [17]. However, in order to avoid limiting the scope of the research to the more established CI/AI paradigms, the definition used here also included algorithms and methods based on fields such as biology, physics, and chemistry [18]. In order to effectively review the literature of CI, the context of CI was investigated first.

A literature search analysis, carried out in March 2014, showed that the keywords “artificial intelligence” along with keywords relating to this study such as “flexible manufacturing systems” started receiving attention in the 1980s, with 10 % of the total search results coming from this decade. The prevalence climbed steeply from there with 28 % in the 1990s, 35 % in the 2000s, and 27 % in the 2010s to date. This analysis also produced on average eight times more results when using the keywords “artificial intelligence” versus “computational intelligence”. From this it was clear that much more attention has been given to AI with regard to manufacturing systems research over CI, which is an indication that CI is not quite mature as a field of research in manufacturing systems. The literature most relevant in the context of this study has been reviewed.

The optimisation of manufacturing systems has an active field of research with as many focus areas as there are subsystems. Renzi et al. [20] and Dias Ferreira [21] present exhaustive lists of literature of CI methods used in optimising AMSs. From the literature it is clear that some of these focus areas have received much attention in research through application of different methods that can be described as CI, among other methods. The different areas of application have been classified in two broad categories namely, manufacturing planning, and manufacturing operations. These two categories have been used as the basis for this study, and will be used as basis for the discussion of the relevant literature in the following subsections. In manufacturing planning, the emphasis was on Aggregate Production Planning (APP), and in manufacturing system operations, the emphasis was on scheduling. This was done to focus the research direction as much as possible.

2.3 Manufacturing planning for Mass Customisation Manufacturing

Manufacturing planning involves planning the use of resources for optimal production. This includes capacity planning and APP for variation in overall demand [3]. In order to focus attention on the manufacturing planning alone, the assumption was made that the facility layout is determined and fixed, in other words travel times between workstations are known and can be modelled by probability distributions. Material Requirements Planning (MRP) represents the link between the production plans, and the operational control system. However, it is more closely related to operational control and scheduling, and thus will be discussed later in this chapter.

2.3.1 Capacity planning

Capacity planning relies on many considerations, most important of which are maintaining system balance, frequency of capacity additions or adjustments, and the use of external capacity [3]. The capacity planning problem is trivial for a system which is required to produce a constant supply of a fixed set of products, but it becomes more complex when overall demand and product variety demand varies over time [22]. According to Ceryan & Koren [23], it is critical for long-term profitability, for a manufacturing enterprise to invest in the optimum quantities and types of capacity at the beginning of a planning horizon.

In this study, the capacity planning activity represented an initial step towards APP, and was used to determine a range of average order arrival rates for the case study manufacturing system, described in Section 3.1, that produced acceptable system utilisation levels. According to Stecke

[24], the available capacity should be related to the aggregate production requirements as part of the solution to the production planning problem.

2.3.2 Aggregate Production Planning

Aggregate Production Planning is classified as a medium range planning activity, and is primarily used for intermediate time length adjustments to the system. This includes production rates, workforce levels, and inventory levels. A formal statement of APP has been suggested by Chase et al. [3]: "*Given the demand forecast F_t for each period t in the planning horizon that extends over T periods, determine the production level P_t, inventory level I_t, and workforce level W_t for periods t = 1, 2, . . . , T that minimize the relevant costs over the planning horizon.*". Production planning strategies exist that are based around the variables mentioned in the definition statement above. The three standard planning strategies, according to Chase et al. [3], are as follows:

1. Chase strategy, in which the production rates are matched to the order arrival rate exactly by the hiring and laying off of employees as the order arrival rates vary.
2. Stable workforce – variable work hours, in which production is varied by varying the number of production hours worked, by implementing flexible shifts or overtime.
3. Level strategy, in which the workforce is kept stable with constant output rates allowing for inventory build-up or shortages.

These can be implemented individually as standard strategies, or they can be combined in various ways. It is often the case that a combination, or mixed strategy produces the best results, where mixed strategies are usually found through a manual charting process known as cut-and-try [3]. Emphasis can be placed on a single cost parameter, or the objective can be to minimise the overall cost over the planning period. Mathematical techniques such as linear programming, goal programming, and mixed integer programming have also been proposed for solving the APP problem [25]. One major obstacle to finding optimal plans using analytical models is the inherent imprecision and uncertainty in the input data such as demand forecasts [3].

Researchers have been occupied with the APP problem since the late 1960s, implementing goal programming models in order to account for the divergent objectives of the various system variables [26], [27]. A review of production planning models by Mula, et al. [28] revealed that fuzzy modelling has been the most popular technique used for addressing uncertainty in demand forecasting. Wang and Fang [29] attempted to account for uncertainty and imprecision using a fuzzy linear programming model for solving the APP problem with multiple objectives.

Baykasoglu [25] proposed a model for solving the APP problem using a multiple objective Tabu Search (TS) algorithm in order to address a lack of solutions that addressed multiple criteria.

These criteria included finding optimal production levels, inventory levels, and overtime and subcontracting. Their model also included strict requirements such as satisfying forecasted demand in each planning period. Baykasoglu concluded that the multiple objective TS algorithm is a promising candidate for the APP problem. However, out of six tests using different parameter sets for the TS algorithm, only a single set produced a lower cost plan than the ideal solution produced by a pre-emptive goal programming method. This led one to believe that the TS algorithm lacks robustness in the setup of its parameters. This may have been due to an overcomplicated solution technique.

Baykasoglu and Gocken [30] suggested a solution method aimed at solving the fuzzy multiple objective APP problem, using ranking methods of fuzzy members and TS. The authors found no significant improvement between the proposed method and the comparison method used. They concluded that the proposed fuzzy solution found was in the vicinity of the crisp solution and did not require defuzzification to improve the results. This is further evidence that attempting to match the complexity of the problem by adding layers to the already complex solution from [25] was not worthwhile.

Leung and Wu [31] developed a model for stochastic APP optimisation to account for uncertainty by considering multiple scenarios and producing options to be made use of by decision makers. However, though it accounted for multiple scenarios, it was found to be valid only for the parameter values used, and could not be extrapolated. This model matched the complexity of Baykasoglu [25] very closely in terms of number of parameter and objectives considered. Although this is proposed as a robust model for use by production planners, it still allows for a certain level of under-fulfilment of demand, which is a major compromise in the realm of production planning.

An imprecise goal programming model for aggregate planning was developed by Mezghani et al. [32]. Special consideration was given to the manager's preferences in the form of a satisfaction function. The model attempted to account for imprecision in input data and technological parameters through the use of fuzzy member functions. The model was able to produce improved results over a fuzzy linear programming model. However, the results were still dependent on the decision maker's knowledge with regards to the form of the satisfaction function and their own preferences.

The literature reviewed here indicated that most attempts at solving the APP problem have been aimed at fitting the problem to the nature of the solution, or attempting to match the complexity of the solution to the complexity of the problem. Based on the findings of this literature review, it has been hypothesised that a better solution could be found if an approach was taken of tailoring a new algorithm to the nature of the problem, and basing this solution on tried-and-trusted methods. This has been addressed here with the implementation of a Biogeography-Based Optimisation algorithm.

2.3.3 A new algorithm for Aggregate Production Planning optimisation

The Biogeography-Based Optimisation (BBO) algorithm has been identified for the optimisation of aggregate production plans. The BBO paradigm was selected because it had not been implemented in this application area before. It is also believed that the operation of the BBO principle aligns well with the nature of the APP problem. Specifically, the discrete population-based nature of the algorithm suits the discrete nature of the aggregate production plan structure [33].

Aggregate Production Planning is a time consuming activity to perform a trial-and-error search of possible optimal solutions. A manual search method described by Chase et al. [3] is known as the cut-and-try method, which is an approach based on costing various planning alternatives and selecting the best one. The BBO algorithm replicates and accelerates this process. Further details on the algorithm can be found in Chapter 4.

2.4 Control for Mass Customisation Manufacturing

The main objective of research effort in the area of MCM has been in achieving the optimal balance between economy of scale and economy of scope [6]. The method suggested for approaching economy of scale, while managing the scope on the shop floor is through the implementation of an effective operational control strategy. Irrespective of the manufacturing strategy implemented for achieving MC, the objectives remain the same: maximising profit through minimising cost and maximising sales, with the defining characteristic of providing customised products. Several strategies for manufacturing system control exist [34]. However, not all strategies meet the requirements imposed by MCM as most were developed before the advent of MCM. Manufacturing system control philosophies as well as shop floor scheduling strategies have been investigated here.

2.4.1 Manufacturing system control philosophies

The collective term for the implementation of a manufacturing system control strategy is Manufacturing Execution System (MES).The main task of an MES is scheduling operations on the shop floor, in order to maximise throughput, minimise Work-In-Process (WIP), and minimise operating expense [3]. Traditional MESs implemented for mass production, characterised by high volume and low variety, are linked with Material Requirements Planning (MRP) systems and schedule operations based on a Master Production Schedule (MPS) [3]. This is an inflexible system for controlling operations due to the fact that it is dependent on high certainty in product demand which enables large order and batch sizes. Large batch sizes are also beneficial in a mass production environment, as this reduces the frequency of setups during production.

In MCM, large batch sizes are infeasible due to the variety in product configuration in consecutive orders released into the manufacturing system. Each consecutive order in MCM can be unique and different from all previous orders. This may necessitate frequent setups to be performed at each workstation. The lean manufacturing philosophy was developed for high volume low variety production, and also relies on the link with the MRP system for generating feasible operating schedules in a synchronous manner [35]. However, Stump & Badurdeen [36] suggested that some lean manufacturing principles can be applied to MCM, depending on the level of customisation. The defining characteristics of this strategy are to minimise levels of raw material, WIP, and finished goods held, as well as lead time.

Shorter lead times can be directly linked to customer satisfaction [7]. Lower levels of inventory held across the shop floor could effectively improve responsiveness of the system in the sense that more variety can be built into raw materials and parts ordering. This can be done due to the fact that shorter planning horizons are necessitated by the lower inventory levels, in other words orders are only released when the materials or parts are required on the shop floor. This is known as a pull system traditionally implemented in the form of Just-In-Time (JIT) or Kanban systems [37]. This approach fits the mould of MCM in the sense that customer orders pull the required parts and materials into and through the manufacturing system. This does not account for the possible variety that can occur in consecutive customer orders, but this can be accounted for in other ways.

Implementation of the agile manufacturing philosophy has been proposed to account for frequent fluctuations in product demand and frequent product changeover [38]. According to Yusuf et al. [39] the main objectives of agile manufacturing align very well with the requirements of MC. These include, among others, highly customised products, mobilisation of core competencies, and response to change and uncertainty. Furthermore, Goldman & Nagel, [40] argued that agile manufacturing integrates a comprehensive range of flexible manufacturing technologies, and includes lessons learned from total quality management (TQM), JIT and lean manufacturing. Within the scope of this research study, the applicable concepts of the agile manufacturing strategy match those identified in previously discussed strategies. These include limiting the inventory on the shop floor, establishing a pull mechanism for material flow, and flexible process plans for enabling customisation.

The strategy of Synchronous Manufacturing (SM) is based on Eli Goldratt's [41] Theory of Constraints (TOC), and more specifically Optimised Production Technology (OPT) [3]. Synchronous Manufacturing refers to the manufacturing system as a whole working in unison toward the goal of generating profit for the firm. However, according to Chase et al. [3], many control strategies focus on maximising the utilisation of all workstations in order to achieve maximum throughput. These strategies use resource utilisation as a measure of performance which can encourage the overuse of non-bottleneck stations and result in excess inventory. This is one of the primary motivations for SM to focus on balancing the flow of a product rather than balancing the capacities of workstations, through managing the inevitable bottlenecks in the system.

In SM, shop floor resources are categorised according to their capacity relative to the demand for their outputs, if a resource has less capacity than the demand for its output it is classified as a bottleneck. The converse of that is classified as a non-bottleneck, and a capacity-constrained resource is classified as a resource whose capacity is slightly higher than the demand for its output. Chase et al. [3] state that improving the performance of a bottleneck has a direct positive effect on the throughput of the system. Thus identifying these categories improves the efficiency of the control strategy by focussing control efforts only on those resources, or work stations, which are behaving as bottlenecks. This is a universal effect that can be translated to MCM, with a single limitation. That is, in MCM there is a probability that bottlenecks may shift during operation of the manufacturing system. Although this is also a possibility in mass production environments, the frequency at which bottlenecks may shift in MCM would be much higher, which would necessitate a level of control for handling these shifts.

Combinations of the strategies discussed in this section can be formed in order to achieve the required flexibility and agility depending on the market demands. Figure 2.1 has been adapted from Shell and Hall, Eds. [37] to indicate the area of manufacturing control approaches MCM is striving towards. Ultimately, the planning for and operation of manufacturing are governed by the product being manufactured in terms of the paradigms to implement, and the specific types of workstations to install. The product being manufactured also determines the market to be serviced, which influences the paradigms to be implemented in terms of scale, flexibility, and agility. A case study has been formulated in this research study that is believed to enable MCM using FMS-based technology and a pull production system.

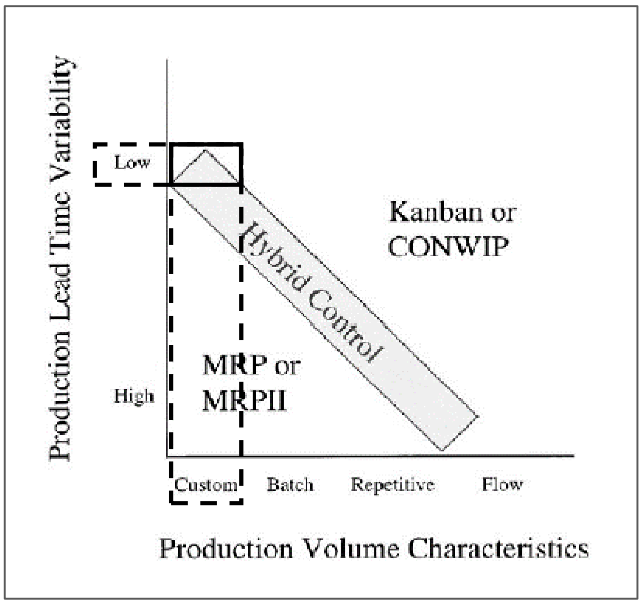


Figure 2.1. Manufacturing system control approaches. Adapted from [37].

2.4.2 Scheduling for Mass Customisation Manufacturing

From a practical perspective, effecting the control strategies necessary to maintain the desired levels of manufacturing lead time variability in response to the production volume characteristics requires effective control policies at the shop floor level. Many approaches have been proposed for solving shop floor control problem. These have been classified according to the underlying technology [42]:

1. Analytical approaches
2. Heuristic approaches
3. Simulation-based approaches
4. Artificial Intelligence-based approaches

Sharifnia et al. [43] suggested a control theoretic construct that deconstructed the flow control problem into multiple sub-problems. Lan et al. [44] investigated a single server system producing multiple products for the effects of setup cost of the performance of the system. Mathematical modelling has also been proposed for solving the shop floor control problem. Problem formulations for FMS control operation were presented by Stecke [45]. However, these were large nonlinear integer problems which were computationally intensive. It is the opinion of the Author that the control theoretic and mathematical model approaches require simplifying assumptions beyond reasonable practical application.

Online simulation has been used to select appropriate selection rules for the manufacturing system based on the state of the system. This approach has been found to be cumbersome, as the simulator needs to run multiple simulations for the state at hand in order to select the appropriate rule [46].

Chiu and Yih [46] also found that the scheduling interval, or rate at which the selections are made should be determined at machine level and not at system level. Traditionally, the simulation-based approach has been used in conjunction with well-established AI paradigms such as ANNs [42], [47]. Machine learning technology suits this application due to the large amount of data produced by such simulation-based scheduling systems. According to a review by Negahban and Smith [48], AI principles including TS and fuzzy systems have also been applied, in conjunction with simulation, to the flow-shop scheduling problem.

Another approach that has received research attention is the application of heuristics to the scheduling problem, in the form of dispatching rules, also referred to as selection rules. Panwalkar and Iskander [49] surveyed a large number of selection rules and techniques, and classified these according to their complexity and performance targets. They recommended using a combination of simple priority rules, or a combination of heuristics with a simple priority rule, in order to avoid ambiguity at selection. Scheduling problems with setup times or costs were surveyed by Allahverdi et al. [50], in which the problems were categorised according to system type, sequence dependence, and whether batching was considered. The authors classified the system under investigation in the case study developed in this research study as a flexible flow-shop, non-batching, sequence-dependent scheduling problem. Although setup times are not one of the primary considerations in this study, this is a useful classification. Allahverdi et al. also found that GAs and Simulated Annealing (SA) commonly found application to problems in the same category as the one at hand.

2.4.2.1 Dynamic scheduling for Mass Customisation Manufacturing

Park et al. [51] developed an adaptive scheduling policy for FMSs, that selected an appropriate selection rule depending on the state of the system. This application included an inductive learning component employed for constructing selection rule selection heuristics at the system level by simulating training example scenarios. This policy performed well compared to a single system-wide selection rule for systems with frequent disruptions such as machine breakdowns. This contribution had value in its adaptive nature of the policy. However, this was in the selection for system-wide implementation of selection rules only.

In contrast to Park et al. [51], an analytical process model combined with a multiple criteria ranking technique was employed by Reddy et al. [52] to develop a two-stage group heuristic selection rule based dynamic scheduling framework for FMSs. This system assigned part families to machine groups rather than individual parts to individual machines. A family selection heuristic and a selection rule made up the two-stage process. The proposed system was an exhaustive group scheduling system, in other words, it exhausted one part family queue before making the decision of which part family to empty next. Although it was dynamic, it was similar to the contribution of Park et al. [51] in that it was also a system-wide scheduling framework.

Kapanoglu and Alikalifa [53] proposed a learning priority rule scheduling system based on a GA for job shops. This system built a common state priority rule based on queue length intervals. This resulted in condition-action rules for choosing a selection rule, based for different ranges of queue lengths at a system level. Selection occurred between two traditional selection rules. The proposed system produced lower flow times when compared to nine traditional selection rules implemented individually across the system. The simplicity of this approach was an attractive attribute. However, as with previous literature, the rules were applied at the system level which could limit the effectiveness.

Subramaniam et al. [54] proposed a system which dynamically chose selection rules based on fuzzy logic for a job shop environment. Their system used relative workloads in the system as inputs to the fuzzy scheduler to decide on which selection rule to select. The fuzzy scheduler showed marginal improvements over traditional selection rules implemented individually across the system. An advantage of their fuzzy scheduler cited by the authors was that it was a single pass method and required the same amount of computation as one of the traditional selection rules. This was also valuable insight which has been taken advantage of in this study.

Three machine learning algorithms for dynamic scheduling of FMSs were compared by Priore et al. [55]. These were an inductive learning algorithm, an ANN algorithm, and a Case-Based Reasoning (CBR) algorithm in the form of a nearest neighbour (k-NN) algorithm. Of these three the k-NN algorithm produced the lowest makespan. However, these results were in the order of 5 % better than the best performing traditional selection rules. Multiple monitoring periods were tested as it was cited that this period determined the performance of the system. This period determined the frequency at which control attributes were tested to decide whether to change the selection rules. The authors made the point that although their test FMS was of a generic configuration their results could not be generalised to any type of FMS.

According to Nakasuka and Yoshida [56], a scheduling system attempting to dynamically modify selection rules should meet two requirements:

1. Real-time operations must not be delayed by choosing of selection rules
2. A variety of system information must be considered in real-time by the selection operation

The purpose of this section has been to obtain a view of both the philosophical and practical aspects of manufacturing system scheduling and control theory, in order to bring these two together in a novel way to achieve the objectives of maximising customer satisfaction through minimising order lead times for mass produced custom products. The instrument that has been identified for this purpose is CI. Literature proved that dynamic scheduling is not a new topic. However, the simplicity of selection rules both in their structure and in their implementation made them attractive as a starting point for further development of a new dynamic scheduling strategy.

2.4.2.2 A new scheduling approach

A relatively newly developed paradigm in CI in the form of the Harmony Search (HS) algorithm shows potential for implementation in manufacturing system scheduling. Parallels can be drawn between the metaphor on which HS is based and the nature of the manufacturing system scheduling problem. The HS paradigm is based on the metaphor of a group of musicians playing together by improvising their tune according to what the other musicians are playing [57]. The objective of these musicians is to keep a pleasing harmony while playing together. Each musician has a memory of musical notes from which to select the one to play at any given time in response to changes by the other musicians. The music is played at a specific rate or tempo, but this tempo may change while the musicians are playing. Each musician also decides at which rate to change their note or pitch, to complement the overall aesthetic of the piece.

Analogous to the HS metaphor, the flexible flow shop manufacturing system consists of a number of workstations working together to produce a certain product range. The objective of the manufacturing system is to produce that product in as short a time as possible. Each workstation performs a different process to achieve the finished product, and each process possesses different characteristics. Most important of which are the time required to complete the process and the order in which to process the parts entering the workstation. The tempo of the manufacturing system is dictated by the arrival rate of orders to the system, and the rate at which the workstations can respond to the changes in this tempo is dictated by their setup and processing times. These parallels have inspired the development of a new scheduling strategy based on workstations individually responding to the order arrival process, system state, and local state based on their own response characteristics.

The literature review has shown that HS has received limited attention in the area of manufacturing scheduling research. Wang et al. [58] offered an HS algorithm for the blocking flow shop scheduling problem. The proposed algorithm was a hybrid modified global-best HS algorithm used for optimising the processing sequence of jobs to minimise makespan. The algorithm performs well in comparison with algorithms such as GA, TS, and a greedy search algorithm. The operation of the algorithm was predicated on knowledge of the jobs requiring processing at the start, i.e. it was a static optimisation algorithm for a static problem space.

Yuan et al. [59] proposed another hybrid HS algorithm for solving the flexible job shop scheduling problem. The HS solution vector was converted to a new discrete vector form. The algorithm was modified to include a local search component to improve its exploitation ability. The goal here was also to minimise makespan. The performance of this algorithm was compared to other techniques found in literature on multiple datasets, and it produced lower cost plans. These techniques included SA, TS, and Particle Swarm Optimisation (PSO) algorithms. The operation of the optimisation algorithm also depended on knowledge of the complete set of jobs and their processing requirements to develop an optimised schedule.

In 2010 the novelty of HS as a nature-inspired optimisation methodology was brought into question. Research to this effect has been published by Weyland [60] which proposes that HS represents a special case of Evolution Strategies (ES). Geem [61], the original contributor of HS, published a rebuttal in response to this paper highlighting the uniqueness of HS compared to ES and commenting on the value of a new methodology and from where this value should be derived. Geem proposed that the value of a new methodology should be derived from its performance, and not purely from its novelty.

It is the Author's view that both researchers have valid arguments. Weyland's evidence that HS is a special case of the (μ + 1) ES is convincing. Both algorithms are population-based search methods incorporating crossover and mutation operations. Weyland proved from an analytical perspective that the operations of the two algorithms were equivalent and that the best ES is at least as effective as HS. His advice for caution in blindly accepting allegedly novel optimisation algorithms is wise. Geem stated that the HS algorithm was originally developed for discrete optimisation problems, whereas ES was originally developed for continuous optimisation problems. He also points out that the generation operation of HS uses all known harmonies, or candidate solutions, whereas the crossover operation of ES requires two 'parent' solutions.

The Author is of the opinion that research in the field of optimisation algorithms is littered with areas where multiple variations of the same theme have been offered in publication. In the book by Xing and Gao [18], various themes are described under which multiple algorithms are outlined which offer only minor variations on the original concepts. One example is the section on Bees Algorithms (BAs). The main principles of the majority of BAs are based on scout bees searching for promising sources of nectar, returning to the hive and communicating their exploits to worker bees through a sequence of moves known as a waggle dance [18]. Each of the BAs includes these principles with minor variations, tailored for a specific type of optimisation problem.

Whether HS is called HS or Modified Evolutionary Strategy, it is a contribution none the less, and it can give rise to further research building on the same theme. In light of this, the Author offers a new application and new adaptation of HS as research contribution in the hope to further the field of research into manufacturing systems scheduling optimisation.

Chapter 5 presents a dynamic scheduling strategy that has been designed based on useful aspects of the manufacturing system control philosophies discussed in Section 2.4 and inspired by the functionality of HS. This strategy has been built on the structure of scheduling policies based on heuristics and selection rules for their tried-and-trusted status.

2.5 Chapter summary

This chapter highlighted past research into the facilitation of MCM, and the application of CI in manufacturing system optimisation research. Manufacturing has been discussed from a system philosophy perspective as well as a practical shop floor control perspective. Past research into manufacturing planning and manufacturing scheduling was critically reviewed and interpreted to give guidance for the direction of this research study. Important points were highlighted for further consideration in this study.

Chapter 3 Production System Model Development

To investigate the performance of new optimisation paradigms and algorithms, models representing a manufacturing system were developed. This was necessary due to lack of access to a real-world manufacturing system and its data. This chapter describes a hypothetical custom wristwatch product platform from a Mass Customisation Manufacturing perspective. It documents the development of two static production planning models and a dynamic system model based on a manufacturing system for producing this family of products.

3.1 Product Family Architecture

A product family architecture for a range of men's wristwatches was the basis for the development of the hypothetical production system. The range allows customisation both in terms of discrete feature selection and continuous feature customisation. The men's wristwatch is an accessory that can be viewed as an expression of one's personality. It also offers a large scope in terms of customisability. For these reasons the men's wristwatch was selected for the case study.

3.1.1 Wristwatch product range

The wristwatch product range was divided into two categories based on the type of movement selected by the customer. This selection determined the parts required as well as the assembly of the watch. The two options available in the proposed product range were "standard automatic", which only performs the primary function of keeping time, and "chronograph", which has a stopwatch function in addition to the timekeeping function. An additional variant could be included in the form of a quartz movement. However, this does not affect the functioning of the manufacturing system, as it would require the same processing as the standard automatic movement, and so was not considered in this case study.

Figure 3.1 shows a set of wristwatches to illustrate the level of variations possible in wristwatch production and identify the less intuitively named components. A standard movement only requires a crown for setting the time, while a chronograph also requires pushers and their associated seals for start-stop and reset functions of the stopwatch function. The chronograph model also requires assembly of three small hands to the movement for the stopwatch. Apart from these differences the same components are required for the final product. Components that are customisable include the bracelet, case, bezel, crown, back case, dial, and all hands.

3.1.2 Bill of Materials

The product range required the manufacture of a subset of parts, and the purchase of the rest of the parts. These parts were then assembled into sub-assemblies and finally into the complete assembly of the wristwatch. The BOM structure can be seen in Figure 3.2. The total number of discrete parts in the structure was 23, six of which were manufactured according to customer specification in the system. The rest of the parts included in the final assembly were purchased from various suppliers based on orders received from customers. The movement, although an assembly, is treated as a single purchased part.

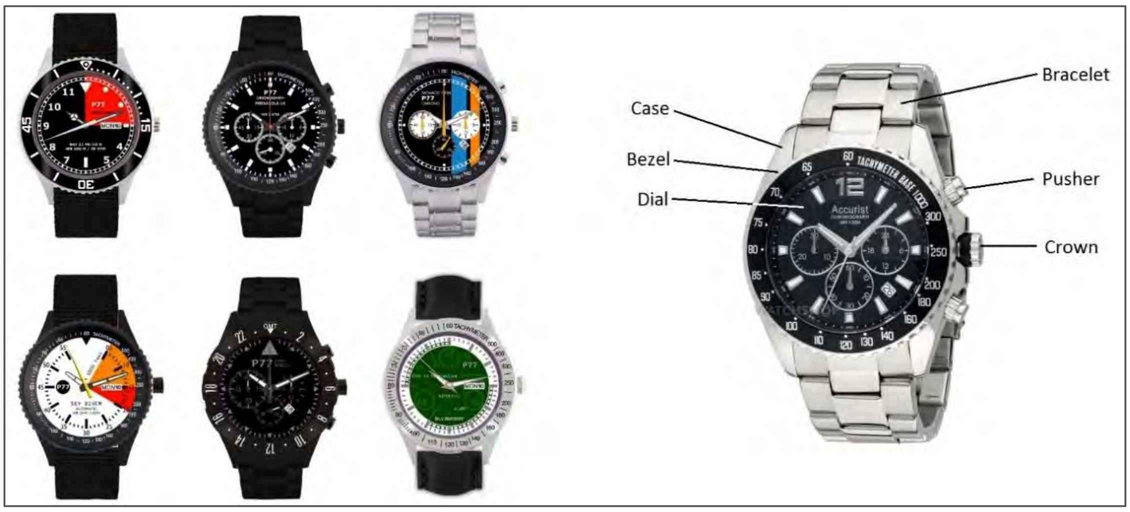


Figure 3.1. Variations in men's wristwatches. Images adapted from [62] **and** [63]**.**

3.1.2.1 Assembly

Starting from the end product, the wristwatch is assembled from the bracelet, two bracelet pins, and the body sub-assembly. The body sub-assembly consists of the bezel, glass, glass gasket, case sub-assembly, time sub-assembly, crown sub-assembly, spacer, back case gasket, and finally the back case. The case sub-assembly is made up of the case, two pusher case tubes, two pusher springs, two pusher seals, two pushers, two pusher screws, and a crown case tube. As indicated in Figure 3.2 by dashed borders, all parts for the pushers, i.e. case tubes, springs, seals, screws, and the pushers themselves are only included if the customer selected a chronograph movement.

The time sub-assembly consists of the dial, dial holding pin, hour hand, minute hand, second hand, three small hands, and the movement. Here, the three small hands are also only included if the customer selected a chronograph movement, as shown in Figure 3.2. Lastly, the crown sub-assembly is made up of the crown, stem, and crown seal.

The six parts which are fabricated in-house are the bezel, case, crown, dial, spacer, and back case. All of these parts are customisable by the customer setting preferences and specifications. Other parts which are customisable either depend on the specifications of the fabricated parts, or are selected from a range. The case, bezel, and crown may also be coloured, if the customer selects this option.

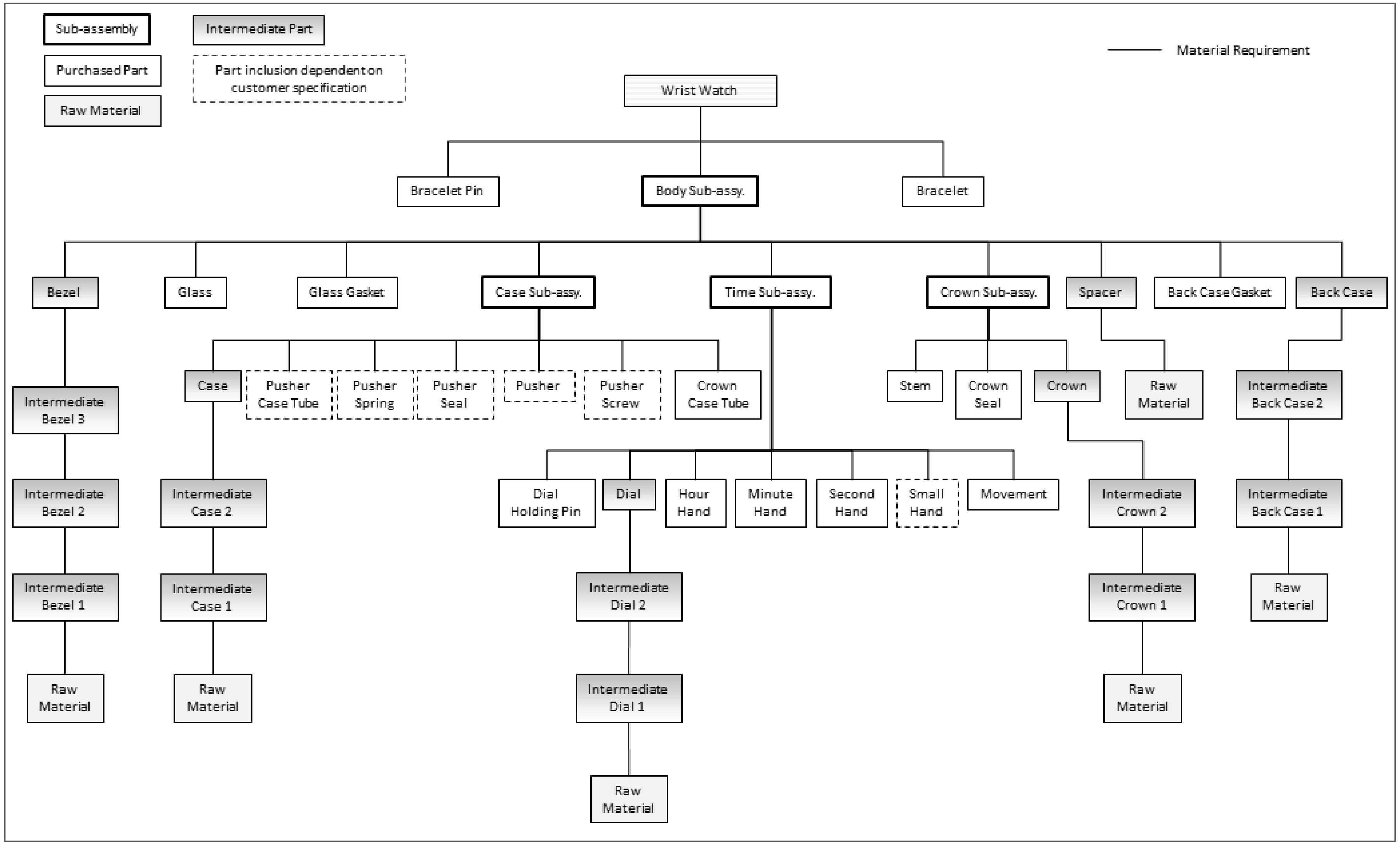


Figure 3.2. Bill of Materials Structure for generic product of case study manufacturing system.

3.1.2.2 Product differentiation

Product differentiation comes from various sources in the fabrication and assembly of the wristwatch. Table 3.1 shows the sources and types of product differentiation available to the customer, as well as the effects of selections on other parts of the product. These represent the hypothetical wristwatch product range.

Table 3.1. Product differentiation sources and types.

Part Family Name	Variants	Customer Input	Source of variation	Type of variation	Component Type
Bezel	Y	Y	Determined by selection of dial size	Discrete (Modular)	Stainless Steel Tube Stock
Glass	N	N	Determined by selection of dial size	Discrete (Modular)	Purchased
Glass gasket	N	N	Determined by selection of dial size	Discrete (Modular)	Purchased
Pusher	N	N	Determined by selection on movement	Yes/No	Purchased
Pusher screw	N	N	Determined by selection on movement	Yes/No	Purchased
Pusher spring	N	N	Determined by selection on movement	Yes/No	Purchased
Pusher case tube	N	N	Determined by selection on movement	Yes/No	Purchased
Pusher seal	N	N	Determined by selection on movement	Yes/No	Purchased
Crown case tube	N	N	N/A	N/A	Purchased
Case	Y	Y	Selection from range (design & colour)	Discrete (Modular)	Stainless Steel Billet
Dial	Y	Y	layout determined by movement & design & colour selection from range or own design & size selected from range	Continuous	Stainless Steel sheet metal
Dial holding pin	N	N	N/A	N/A	Purchased
Hour hand	N	N	Determined by selection of dial size	N/A	Purchased
Minute hand	N	N	Determined by selection of dial size	N/A	Purchased
Second hand	N	N	Determined by selection of dial size	N/A	Purchased
Small hand	N	N	Determined by selection on movement	Yes/No	Purchased
Movement	Y	Y	Selection of type from a range	Discrete (Modular)	Purchased
Crown seal	N	N	N/A	N/A	Purchased
Stem	Y	N	Determined by selection of movement	Discrete (Modular)	Purchased
Crown	N	N	Designed by customer	Continuous	Stainless Steel Bar Stock
Spacer	Y	N	Size determined by dial size and height determined by case height	Continuous	Purchased
Back case gasket	N	N	Determined by selection of dial size	Discrete (Modular)	Purchased

Part Family Name	Variants	Customer Input	Source of variation	Type of variation	Component Type
Back case	Y	Y	Size determined by selection of dial size, & design input from customer	Continuous	Stainless Steel Bar Stock
Bracelet	Y	Y	Material selection by customer and size determined by case design	Discrete (Modular)	Purchased
Bracelet pin	Y	N	Size determined by case design	Discrete (Modular)	Purchased

From the discrete selection type differentiation options in the assembly of the wristwatch alone 65 536 variants are possible. Including the continuous specifications, such as dial design and engraving, there is an infinite number of possible variants.

3.1.3 Bill of Operations

The production system has been specified to include metal fabrication processes, 3D printing, surface treatment, and discrete part assembly operations. This was done to study a manufacturing system that is representative of modern manufacturing technologies. The following subsections describe the generic processes required to produce the case study wristwatch.

3.1.3.1 Fabrication processes

Five of the parts that are fabricated in-house are machined from metal stock, which is mainly stainless steel, and the sixth is 3D printed from ABS plastic. Table 3.2 shows the processing and material requirements for each of the in-house fabricated parts. These setup and processing times are hypothetical.

Table 3.2. Processing and material requirements for in-house fabricated parts.

Case		Total Processing Times (min)	Setup Time /machine	Processing Time /machine
Raw Material:	60 x 60 x 20 mm Stainless Steel Billet	58.5	25.5	33.0
Processing Machine Type 1:	4-Axis CNC Milling Machine	48.0	21.0	27.0
Processing Machine Type 2:	Multi-purpose Grinding Machine	5.5	2.5	3.0
Processing Machine Type 6:	Physical Vapour Deposition (PVD) Machine	5.0	2.0	3.0
Dial		**Total Processing Times (min)**	**Setup Time /machine**	**Processing Time /machine**
Raw Material:	0.5 x 50 mm Stainless Steel Sheet Metal Roll	19.0	9.5	9.5
Processing Machine Type 3:	Piercing and Blanking Machine	3.5	2.5	1.0
Processing Machine Type 4:	Pad Printing Machine	12.5	5.0	7.5

Processing Machine Type 7:	Stencil Gluing Machine	3.0	2.0	1.0
Crown		**Total Processing Times (min)**	**Setup Time /machine**	**Processing Time /machine**
Raw Material:	Dia. 10 mm Stainless Steel Bar Stock	32.0	15.0	17.0
Processing Machine Type 5:	Feed-through CNC Lathe	17.0	7.0	10.0
Processing Machine Type 1:	4-Axis CNC Milling Machine	10.0	6.0	4.0
Processing Machine Type 6:	PVD Machine	5.0	2.0	3.0
Back Case		**Total Processing Times (min)**	**Setup Time /machine**	**Processing Time /machine**
Raw Material:	Dia. 60 mm Stainless Steel Bar Stock	33.0	16.0	17.0
Processing Machine Type 5:	Feed-through CNC Lathe	20.0	10.0	10.0
Processing Machine Type 1:	4-Axis CNC Milling Machine	9.0	4.0	5.0
Processing Machine Type 2:	Multi-purpose Grinding Machine	4.0	2.0	2.0
Bezel		**Total Processing Times (min)**	**Setup Time /machine**	**Processing Time /machine**
Raw Material:	OD 50 mm x ID 20 mm Stainless Steel Tube Stock	36.0	19.5	16.5
Processing Machine Type 5:	Feed-through CNC Lathe	20.5	11.0	9.5
Processing Machine Type 1:	4-Axis CNC Milling Machine	6.0	4.0	2.0
Processing Machine Type 2:	Multi-purpose Grinding Machine	4.5	2.5	2.0
Processing Machine Type 6:	PVD Machine	5.0	2.0	3.0
Spacer		**Total Processing Times (min)**	**Setup Time /machine**	**Processing Time /machine**
Raw Material:	ABS plastic filament roll	27.0	2.0	25.0
Processing Machine Type 8:	3D Printing Machine	27.0	2.0	25.0

Processing times are nominal times and probability distributions of these times were used in the simulation models described later in this chapter. The PVD machine processing steps are only included if the customer selects a colour for their watch other than raw stainless steel.

3.1.3.2 Assembly processes

Similarly to the processing times, nominal assembly times were used to model the delay incurred by the various assembly processes. Table 3.3 shows the nominal setup times and assembly times for the five assembly operations performed during the course of the manufacturing process. These setup and assembly times are hypothetical.

Table 3.3. Setup and assembly times for all assembly operations.

Assembly Operation	Work Station	Setup Time (min/operation)	Runtime (min/operation)	Fixture/ Setup
Wristwatch Assembly & Inspection (A5)	WS-A5	1.0	3.0	S-A5
Watch Body Assembly & Inspection (A4)	WS-A4	2.0	18.0	S-A4
Crown Sub-assembly (A3)	WS-A3	1.0	3.0	S-A3
Time Sub-assembly (A2)	WS-A2	1.0	7.0	S-A2
Case Sub-assembly (A1)	WS-A1	1.0	8.0	S-A1

3.1.4 System process flow

The production system was configured in such a way that once an in-house fabricated part is completed, it is transported directly to the workstation where it is required for assembly, instead of batch transportation. This placed higher complexity on the scheduling of operations due to the fact that there are more parts moving around in the system throughout the production run. However, this should be outweighed by the saving in holding of the finished parts waiting for assembly at separate locations such as an Automated Storage and Retrieval System (ASRS). This also forces each part to be fabricated on-demand from orders placed by customers in line with the pull methodology [36]. The process flow of fabricated parts from raw material to final assembly can be seen in Figure 3.3.

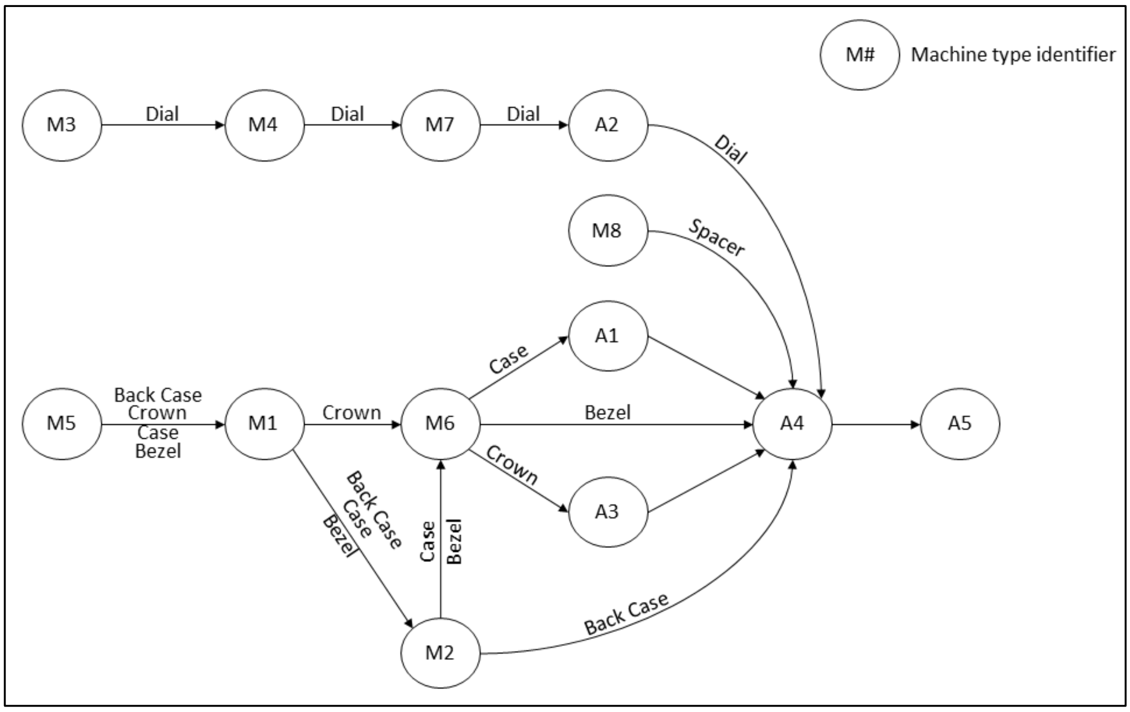


Figure 3.3. Process flow diagram on in-house fabricated parts.

The machine type identifiers in Figure 3.3 refer to the machine types mentioned in Table 3.2, i.e. M3 in Figure 3.3 refers to Machine Type 3 in Table 3.2. Similarly Assembly station identifiers in

Figure 3.3 refer to assembly workstations in Table 3.3. All raw materials enter the system on the left of the figure, either at M3 or M5, depending on the part being manufactured, and move through the system in a general left to right direction, and exits the system as part of a final assembly after assembly station A5.

3.2 Static simulation models

The initial stage of the simulation modelling approach involved the development of two static simulation models of the manufacturing system to characterise the system and to form the foundation for the dynamic simulation model. These were a capacity planning model and an Aggregate Production Planning (APP) model.

3.2.1 Capacity planning model

A capacity planning model was used to determine workstation capacity requirements and represented a snapshot of the system under ideal steady state operating conditions. In this context, capacity planning was aimed at long-term planning and investigating a range of expected average order arrival rates for steady state operation.

3.2.1.1 Capacity planning modelling methodology

The capacity planning model calculated workstation capacity requirements based on processing and setup times as given in Table 3.2 and Table 3.3, and varying order arrival rates. The goal of the capacity planning activity was to maximise the utilisation of all workstations, by addition or subtraction of capacity according to the forecasted order arrival rate. The setup and processing times of each workstation were calculated by taking the processing requirements of each part moving through the workstation and multiplying this by the rate at which the parts were entering the system.

From the processing times and the workstation capacity per shift, the utilisation of each workstation was calculated. The capacities were then adjusted to maximise utilisation of all workstations but still ensure they did not exceed 100 %. Using this procedure, the workstation capacities for any expected order arrival rate could be determined. Equation 3.1 shows this calculation as an expression.

$$Utilisation = \frac{(Setup\ Time + Processing\ Time) \times Order\ Arrival\ Rate}{Available\ Time \times Workstation\ Capacity} \tag{3.1}$$

The setup and processing times for each processing workstation were calculated by adding the times from all the parts requiring processing by the particular machine in question together. The setup and processing times for the assembly workstations were simply taken as the single individual value, as an assembly process necessarily requires all the parts being assembled together all at the same time. For example, all the setup and processing times for the parts being assembled into the watch body were not added together, but the single individual time for the parent part, i.e. the case, was taken as the time required to process all the parts at the same time. The model was created in Microsoft Excel 2013 and a macro was written for traversing the range of order arrival rates automatically. The programming code can be found in Appendix C.

3.2.1.2 Capacity planning model inputs

The capacity planning model took as input an average arrival rate of orders into the system. A range of average order arrival rates was used to investigate the system's response to various arrival rates. The values for the parameters were based on the processing requirements of the wristwatch product range described in Section 3.1. Table 3.4 shows the processing requirements for the 'dial' component of the wristwatch assembly as an example. A table like this one exists for each part that makes up the complete assembly. These tables can be found in Appendix B.

Table 3.4. Setup and processing time requirements per workstation for 'dial' component in minutes.

	Process No.	**1**	**2**	**3**	**4**	**5**	**6**	**Total**	**Arrival Rate (Orders/day)**
Dial	**Station**	WS-M3	WS-M4	WS-M7	WS-A2	WS-A4	WS-A5		10.00
	Setup	2.5	5.0	2.0	1.0	2.0	1.0	13.5	
	Process	1.0	7.5	1.0	7.0	6.0	6.0	28.5	
	Total	3.5	12.5	3.0	8.0	8.0	7.0	42.0	

3.2.1.3 Capacity planning model implementation

The expected utilisations and corresponding workstation capacities were calculated for a range of order arrival rates, from one unit order per day up to 200 unit orders per day. This was done by adjusting the workstation capacities to achieve the maximum utilisation up to, but not exceeding, 100 % for each increment of order arrival rate. The data generated was plotted per workstation for all arrival rate values, as seen in Appendix B. The workstation capacities calculated in this experiment represented the ideal operation of the system across the range of order arrival rates.

The correctness of the capacity model was verified by using system variables from a known dataset published online by the developers of the Simio simulation software, in a worked example [64]. The outputs when compared with those produced by this example matched exactly. The capacity planning model was also used to verify the correctness and validate the behaviour of the dynamic simulation model.

3.2.1.4 Capacity planning model outputs

Table 3.5 shows the calculated capacity requirements of all workstations, including workers operating the workstations, based on an order arrival rate of ten orders per day, as an example of the format of the capacity planning model output. The utilisations shown in Table 3.5 are the actual levels for the calculated capacities as shown and a unit order arrival rate of ten orders per day. The numbers of machine and assembly station operators required were calculated from the setup times for the processing workstations (WS-M#) and the full assembly times required at the assembly stations (WS-A#), respectively.

It was clear from the calculated utilisations in Table 3.5 that ten unit orders per day was not an economically beneficial situation, as most workstation utilisations were well below 100 %. The overall median system utilisation for the range of arrival rates, including capacity adjustments, is presented in Figure 3.4. In Figure 3.4 an early peak appeared at 64 unit orders/day with a median overall utilisation of 90 %, subsequently, a median overall utilisation of 90 % was only reached again at 112 orders/day. Beyond 112 orders/day the median overall utilisation seemed to stabilise around the 80 – 90 % range.

Table 3.5. Workstation capacity requirements based on setup and processing time requirements.

Minutes/Day:	480								
Workstations	**Capacity**	**Dial**	**Case**	**Bezel**	**Back Case**	**Crown**	**Spacer**	**Total (mins)**	**Expected Util**
4-Axis CNC Milling Machine (WS-M1)	2		480.0	60.0	90.0	100.0		730.0	76.0%
Multi-purpose Grinding Machine (WS-M2)	1		55.0	45.0	40.0			140.0	29.2%
Piercing and Blanking Machine (WS-M3)	1	35.0						35.0	7.3%
Pad Printing Machine (WS-M4)	1	125.0						125.0	26.0%
Feed-through CNC Lathe (WS-M5)	2			205.0	200.0	170.0		575.0	59.9%
PVD Machine (WS-M6)	1		50.0	50.0		50.0		150.0	31.3%
Stencil Gluing Machine (WS-M7)	1	30.0						30.0	6.3%
3D Printing Machine (WS-M8)	1						270.0	270.0	56.3%
Case Sub-assembly (WS-A1)	1		75.0					75.0	15.6%
Time Sub-assembly (WS-A2)	1	80.0						80.0	16.7%
Crown Sub-assembly (WS-A3)	1					40.0		40.0	8.3%
Watch Body Assembly & Inspection (WS-A4)	1	80.0	80.0	80.0	80.0	80.0	80.0	80.0	16.7%
Wristwatch Assembly & Inspection (WS-A5)	1	70.0	70.0	70.0	70.0	70.0	70.0	70.0	14.6%
Machine Operators	2	95.0	255.0	195.0	160.0	150.0	20.0	875.0	91.1%
Assembly Station Operators	1							345.0	71.9%

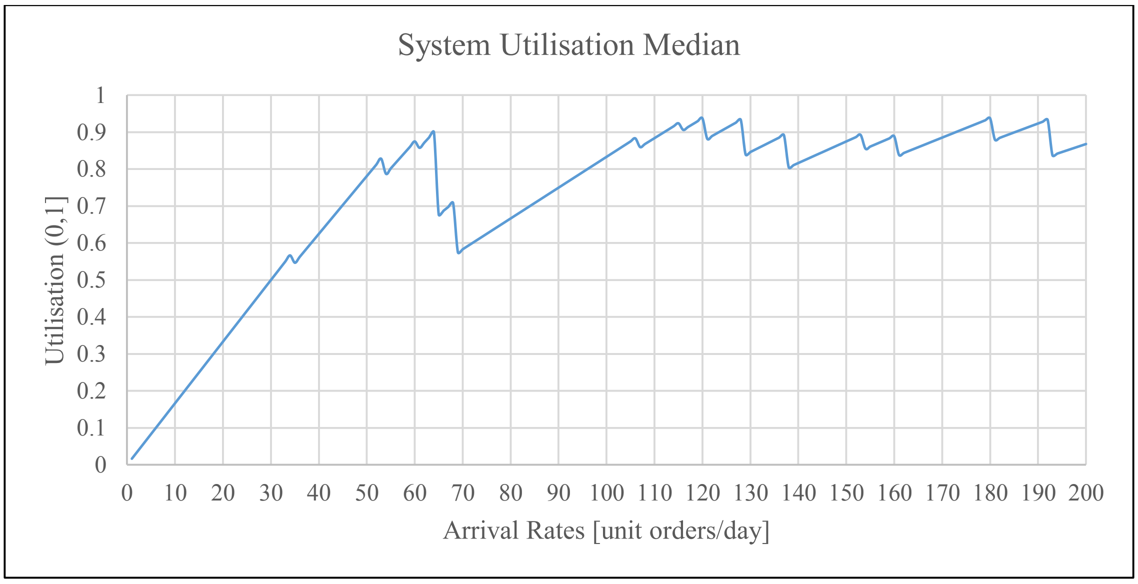


Figure 3.4. System median utilisation over a range of order arrival rates.

3.2.1.5 Capacity planning model discussion

The response of the system utilisation to the range of order arrival rates could be due to coincident drops in individual workstation utilisations in the range between 60 and 110 orders/day. This could be specific to the system under investigation. However, it could be conjectured that this kind of behaviour may appear in other systems, but at different arrival rate ranges.

The graph of median overall utilisation could be used by decision makers to determine whether a certain arrival rate and corresponding production rate is financially viable or not. This is in line with research carried out by Singholi et al. [65] on methods for improvement of FMSs. A threshold of minimum overall utilisation can be set under which it is decided to produce more stock and hold inventory or allow for stockouts to take advantage of higher overall utilisation by under producing for a certain period, and boosting production once a certain level of back orders has built up. The production rate used as this threshold can then be used in APP over a planning horizon to decide on production and workforce level adjustments, whether it be overtime, downtime, or hiring and laying off of workers.

3.2.2 Aggregate Production Planning model

In this context, the APP model represented an intermediate-term planning tool for planning the overall production of units based on the wristwatch product platform. The APP model was used to optimise aggregate production plans such that the cost of production was minimised, by varying monthly production, inventory and stockouts, and workforce by implementing a novel optimisation algorithm. The development of this optimisation algorithm is documented in Chapter 4. This section describes the methodology followed in developing the APP model, its inputs and the structure of its outputs.

The APP model used the demand forecast over a certain planning period to calculate monthly production and workforce requirements, as well as the cost of production [3]. The APP model was more detailed than the capacity planning model in that it looked at the system on a month-to-month basis, whereas the capacity planning model represented a snapshot of the system in steady state.

3.2.2.1 Aggregate Production Planning modelling methodology

The APP model was designed to produce aggregate production plans based on traditional APP strategies. This was done to create a foundation for the APP optimisation algorithms described in Chapter 4. The three main traditional planning strategies, according to Chase et al. [3], are:

1. Chase strategy, in which the production rates are matched to the order arrival rate exactly by the hiring and laying off of workers as the order arrival rates vary.
2. Stable workforce – variable work hours, in which production is varied by varying the number of production hours worked, by implementing flexible shifts or overtime.
3. Level strategy, in which the workforce is kept stable with constant output rates allowing for inventory build-up or shortages.

These strategies can be implemented individually as standard strategies, or they can be mixed in various ways. Traditionally, mixed strategies are found through a manual charting process known as cut-and-try [3]. For example, the order arrival rate could be tracked perfectly for some of the planning period, and for the rest of the period production could be varied to allow for a stable workforce. Emphasis can be placed on a single cost parameter, or the objective can be to minimise the overall cost over the planning period.

The model carried out the necessary calculations to produce a plan based on each of the strategies listed above. It then compared the total cost of these plans to each other and suggested the lowest cost plan for the decision makers. Each plan required slight variations of the base calculations. However, the plans produced all took the same format, shown in Table 3.9 and described in Section 3.2.2.4. This format consisted of the following variables on a per month basis:

- Starting inventory (based on previous month’s ending inventory)
- Production requirements (based on demand forecast, safety stock, and inventory on hand)
- Demand forecast
- Safety stock (based on demand forecast)
- Actual production
- Workforce
- Ending inventory

- Monthly cost
- Total cost up to the planning horizon

Calculations of the variables within the APP model followed a cascading pattern from one month to the next starting with the input starting inventory, demand forecast, and safety stock requirement to calculate the required production. From the production requirement the required workforce and production cost was calculated. However, the actual production requirements did not necessarily need to match the theoretical production requirements as it could be more economical to over or under produce and have surplus or stockout situations in some months. For this reason the exact calculations varied with the planning strategy adopted.

3.2.2.2 Aggregate Production Planning model inputs

This study looked at the overall cost as the objective function, which was calculated as described above, using the inputs to the model as listed in Table 3.6, Table 3.7, and Table 3.8. The arrival rate range produced by the capacity planning model that gave feasible utilisations was used to estimate feasible demand forecasts to use as input to the APP model as shown in Table 3.6. The one-off production parameter and cost parameter values of Table 3.7 and Table 3.8, respectively, are hypothetical.

Table 3.6. Monthly production data parameters for Aggregate Production Planning.

Monthly Production Parameter	**Description**	**Value**					
		Jan	**Feb**	**Mar**	**Apr**	**May**	**Jun**
Working days	Days available per month	22	19	21	21	22	20
Demand forecast	Units required per month up to the planning horizon	1400	1150	1260	1240	1380	1280

Table 3.7. One-off production data parameters for Aggregate Production Planning.

One-off Production Parameter	**Description**	**Value**
Production time	Minutes per unit manufactured	122
Planning horizon	Number of months for which to plan	6
Safety stock	Percentage of demand forecast required to be held as safety stock	25
Starting inventory	Inventory on hand at the start of the planning period	100
Initial workforce	Number of workers in employment at the start of the planning period	16

Table 3.8. Production cost parameters for Aggregate Production Planning.

Cost Parameter	Unit	Value
Cost of holding inventory	Dollars per unit held per month	15.00
Stock-out cost	Dollars per unit short per month	20.00
Worker hiring cost	Dollars per worker hired	500.00
Worker lay-off cost	Dollars per worker laid off	750.00
Manufacturing cost (regular time)	Dollars per hour per worker	20.00
Overtime cost	Dollars per hour per worker	30.00
Downtime cost	Dollars per hour	10.00

3.2.2.3 Aggregate Production Planning model implementation

The APP model was developed to support the development of a novel CI based algorithm for optimising the aggregate production plan. The APP model was programmed in C++ for ease of incorporating the optimisation algorithm which was also programmed in C++. The programming code can be found in Appendix D. Chapter 4 provides further details on the development of the novel optimisation algorithm.

The model read the input data from two separate text files, one containing the production data as per Table 3.6 and Table 3.7 and the other containing the cost parameters as per Table 3.8. The APP model then calculated the production costs associated with the three standard planning strategies discussed in Section 3.2.2.1. The production plan with the lowest associated cost was printed out along with the description of the strategies on which it was based. This plan was used as the benchmark for the optimisation algorithm of Chapter 4.

3.2.2.4 Aggregate Production Planning model outputs

The outputs of the APP model are shown in Table 3.9. These outputs represent the production, workforce, and inventory levels that are required each month to achieve the associated cost. The model determined that the stable workforce – varying production strategy produced the lowest cost plan.

Table 3.9. Best standard strategy based aggregate production plan.

	Jan	Feb	Mar	Apr	May	Jun
Starting Inventory [units]	100	85	131	193	275	280
Demand Forecast [units]	1400	1150	1260	1240	1380	1280
Safety Stock [units]	350	288	315	310	345	320
Production Req.'s [units]	1650	1088	1287	1235	1415	1255
Actual Production [units]	1350	1196	1322	1322	1385	1259
Ending Inventory [units]	85	131	193	275	280	259
Workforce [workers]	16	16	16	16	16	16
Monthly Cost [$]	56 320	48 640	53 760	53 760	56 320	51 200
Total Cost [$]	320 000					

3.2.2.5 Aggregate Production Planning model discussion

Although the APP model is a relatively coarse tool for calculating monthly production and production cost, it is useful for calculating resource requirements on a month-to-month. An APP model can also be programmed to account for known production constraints such as the need for the workforce to remain stable, or the need for a certain number of units left in stock at the end of each planning period.

Aggregate production plans can be developed on a product platform basis for MCM, as has been the case here. However, an APP model is only useful from an MCM perspective if the product range is based on a product platform which inherently lends itself to aggregation, i.e. it contains parts and sub-assemblies that can be grouped together easily in terms of processing requirements. This APP model has been designed for the case study wristwatch product range that consisted of a single product platform for producing custom wristwatches. It formed a good foundation for optimisation of the production plans for this case study. It has also provided a good foundation for development of the dynamic system model.

3.3 Dynamic system operations model

The increased complexity of manufacturing systems and the advancements in desktop computing technology have increased the popularity of simulation modelling in the field of manufacturing systems research [66]. Uses of simulation models in manufacturing systems research can be found in development of new manufacturing systems as well as improvements and modifications to existing manufacturing systems. The main advantage of simulation modelling over prototyping and performing physical pilot studies is the reduction in development cost and capital investments [67]. Verification and validation of a simulation model is critical for it to be accepted as a sufficiently accurate representation of the system under investigation. This section documents the development of the dynamic simulation model of the wristwatch production system case study, in preparation for developing and testing a novel scheduling optimisation strategy.

3.3.1 Methodology

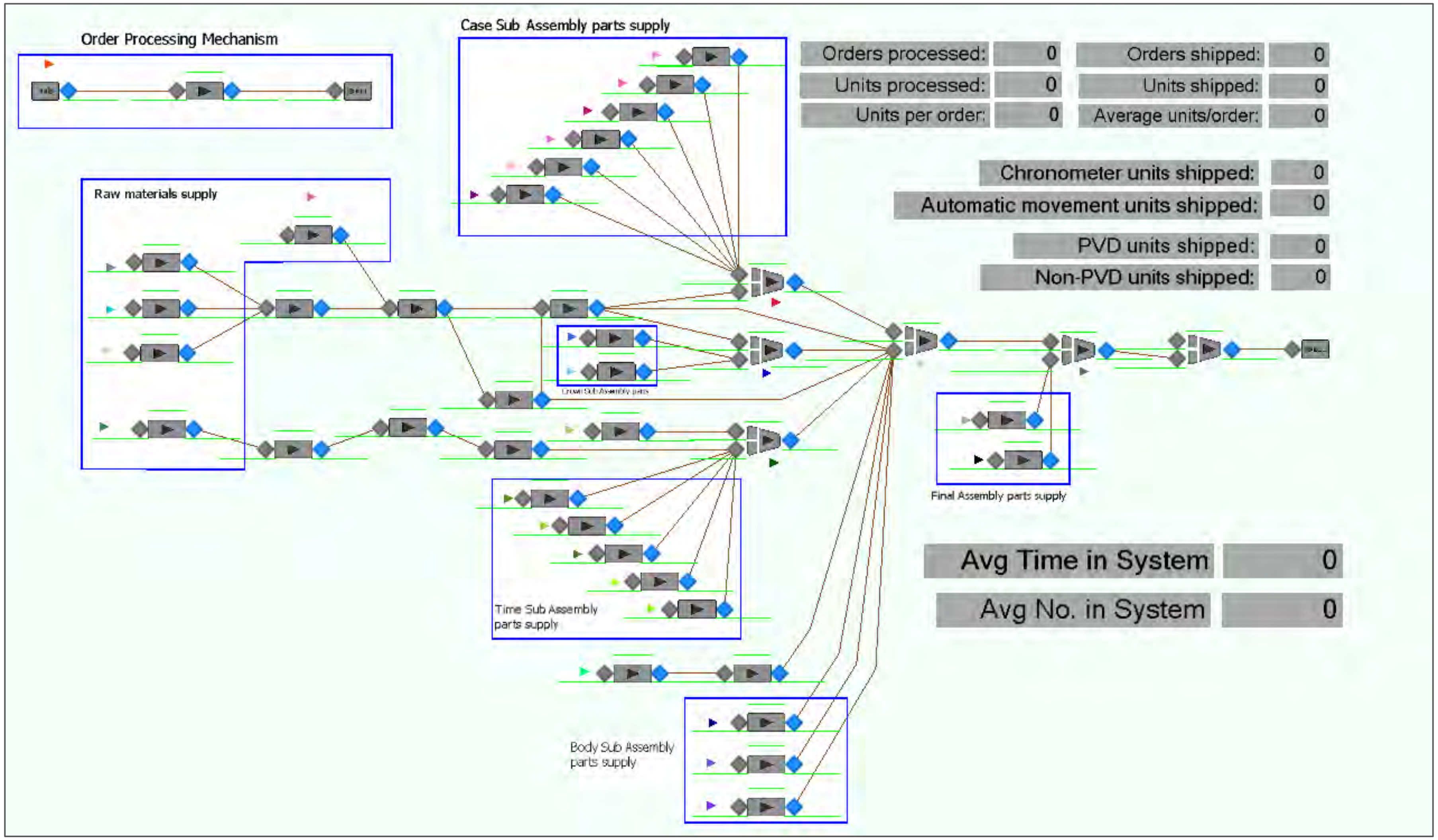
The dynamic simulation model has been developed in Simio Simulation Software (version 7.128.12863) [68]. Simio is a simulation software package which is fundamentally based on queuing theory and Discrete Event Systems simulation (DEVS) [69]. Simio is a general purpose simulation package, but it is very well suited to manufacturing system simulation due to its DEVS foundation.

A dynamic, discrete-time, stochastic simulation model for the wristwatch product range case study was developed. It simulated the lead time associated with the manufacturing activities only, thus no supply chain components were modelled. In other words, a perfect supply of materials and purchased parts was assumed as well as an immediate dispatch of finished goods. The model was made up of objects as listed in Table 3.10. Key functions of these objects and instances in the simulation model are also given in the table.

Table 3.10. Simulation model object types.

Object	Function	Instances
Model Entity	Travel through the simulation model	Parts and sub-assemblies
Source	Create model entities	Raw materials, and purchased part supplies
Server	Delay transfer of model entities	Processing workstations
Combiner	Delay and combine parent and member model entities in a user-defined configuration	Assembly workstations
Sink	Destroy model entities	Order transfer station and dispatching station

Figure 3.5 shows a screenshot of the visual representation of the model in Simio. The release of the order entity into the production system triggered the dispatching of entities representing raw materials to the first stages of fabrication of those parts that are fabricated in-house. These entities follow paths along the processing sequences for each part to be fabricated according to the order specifications. The dispatching of purchased parts into the system was also triggered by the order entity release. These purchased parts may have independently varying order delay times. However, to maintain focus on the production system itself these delays were not modelled. Fabricated and purchased parts converged at assembly stations where intermediate sub-assemblies were produced.


Order Processing Mechanism
Case Sub Assembly parts supply
Raw materials supply
Crown Sub Assembly parts
Final Assembly parts supply
Time Sub Assembly parts supply
Body Sub Assembly parts supply

Orders processed: 0
Units processed: 0
Units per order: 0
Orders shipped: 0
Units shipped: 0
Average units/order: 0
Chronometer units shipped: 0
Automatic movement units shipped: 0
PVD units shipped: 0
Non-PVD units shipped: 0
Avg Time in System 0
Avg No. in System 0

Figure 3.5. Dynamic simulation model visualisation in Simio.

The final stages of the model included the final assembly station and an order completion station where units were dispatched according to their orders. Metadata was linked to each unit and its components when they entered the system, including unit number, order number, and configuration parameters. This information was used throughout the system for scheduling of processing at the individual workstations. By default, parts or sub-assemblies waiting to be processed were processed in order of their unit numbers, lowest to highest.

3.3.2 Input modelling

Customer orders were modelled as batches of single unit size. The inter-arrival time for customer orders was modelled using an exponential distribution as this most closely models actual order arrival processes [70]. It is proposed that these orders may be placed online from any location, including in-store, using a product configurator as proposed by Picario in [71]. The model included an order processing phase which was used to simulate the conversion of product specifications into manufacturing requirements. Subsequently, the order was released into the flexible flow shop manufacturing system as a new job. The order processing time was modelled as a Pert distribution, with a minimum, a mode, and a maximum value specified as required by the function. The Pert distribution is a special case of the Beta distribution where the shape parameters of the Beta distribution are calculated from the minimum, mode, and maximum parameters [72].

All setup times, processing times, and travel times were stored in tables in Simio which were linked to Microsoft Excel spreadsheets for easy modification of these times. Randomness, due to uncertainty in the times required to perform tasks, was modelled using probability distributions that approximate common distributions found in practice. According to Law [66], the log-normal and Weibull distributions are best suited to this type of application. In Simio the LogNormal function takes the normal mean and normal standard deviation as argument, whereas the Weibull function takes a shape and scale parameter as argument. Estimates of log-normal mean and standard deviation were converted to normal values and used in the LogNormal distribution function of Simio.

Hypothetical estimates of the means and standard deviations for the fabrication technologies in use, were transformed for use in the LogNormal distribution function. Let *m* be the mean and *v* the standard deviation of the log-normal. Then Equations 3.2 and 3.3 describe the transformations of the log-normal mean to the normal mean, and log-normal standard deviation to the normal standard deviation, respectively,

$$\mu = ln\left(\frac{m}{\sqrt{1+\frac{v}{m^2}}}\right) \qquad 3.2$$

Where:

μ is the normal mean,

m is the log-normal mean, and

v is the log-normal standard deviation.

$$\sigma = \sqrt{ln\left(1+\frac{v}{m^2}\right)} \qquad 3.3$$

Where σ is the normal standard deviation. The expected value of the log-normal distribution according to [72] is then:

$$E(x) = e^{\left(\mu+\sigma^2/_2\right)} \qquad 3.4$$

All fabrication processing times and assembly processing times were approximated using this distribution function.

Travel times between workstations were modelled using the Pert probability distribution. The travel routes were categorised according to method of transport used between the respective workstations and estimated distances between the respective workstations based on the approximate layout as depicted in Figure 3.6. The supply department supplied raw materials to the fabrication department, and purchased parts to the assembly department. Completed assemblies were transferred from the assembly department to the dispatch department for packaging and shipping.

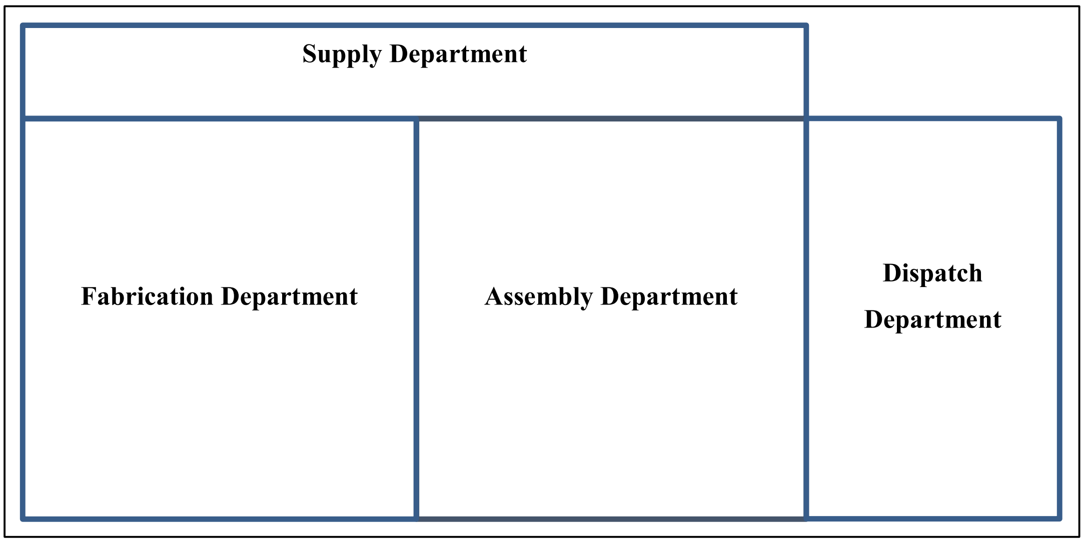


Figure 3.6. Production system layout for travel time estimation.

The following assumptions were made with regard to the method of transport within the production system:

- Transfers within the fabrication department: Automated Guided Vehicles (AGVs).
- Transfer within the assembly department: Conveyors.
- Transfer from supply department to fabrication and assembly departments: AGVs.
- Transfer from fabrication department to assembly department: AGVs.
- Transfer from assembly department to dispatch department: Conveyor to an ASRS.

Relative minimum, mode, and maximum values for travel times were estimated based on these assumptions, which were used in the Pert probability distributions.

3.3.3 Product specification handling

The various product specifications affected the product configuration and production in different ways. For example, the colour of the dial was an important customer-selectable product specification. However, this specification did not affect the processes required for the production of the wristwatch. In contrast, the selection between a chronograph and a standard automatic movement affected the fabrication processes required as well as the assembly of the product. The two most influential product specifications were the selection between a chronograph and a standard automatic movement, and the option of having the watch body coloured using the PVD process or not.

The option of having the watch coloured required that the bezel, case, and crown parts be treated in the PVD process. If this was not selected this process was skipped for these parts. Selecting a chronograph movement over a standard automatic movement required additional holes to be drilled and tapped in the case for the chronograph pushers. It required the inclusion of pusher parts in the assembly of the case, and it also affected the geometry of the spacer used to hold the movement in place in the final assembly. The chronograph selection also required additional hands to be assembled in the time sub-assembly which included the dial, movement, hands, and dial holding pins.

The model addressed product specifications by making use of the intelligence built into the model entities available in Simio. States linked to each entity instance, which were initialised at the time of order arrival, were propagated through the model to retain processing requirements in terms of processing times as well as process inclusion throughout the production process. These configuration states were Boolean and were modelled using discrete probability distributions; configurations thought to be more popular having higher probability values. Each unit ordered could have its own individual configuration and the sequence of configurations was completely unknown. In other words, the production system was required to handle any sequence of consecutive configurations.

3.3.4 Model verification

Verification of computer simulation models is an important step in the model development process to ensure confidence in the correctness of the programming of the simulation model [73]. Verification of the computer simulation model was performed by way of both static and dynamic tests. Product configuration specifications were set to constant known values and the outputs were recorded in order to determine whether the outputs matched the input specifications and to determine the consistency of the outputs. The model itself was also monitored while running in order to confirm that the correct product configurations in terms of part requirements were adhered to for known product configurations.

The two product configuration specifications that had the largest effect on the processing requirements in the production of the wristwatch were whether the customer selected a chronograph movement or a standard movement, and whether the customer selected a coloured finish on the complete watch or not. These two specifications were set to constant values in combination with each other, giving four different product configurations. The simulation model was run under each of these configurations and the outputs were found to be consistent with the product configurations in all four combinations.

The system throughput under the different product configuration specification conditions was also recorded and checked for consistency with regard to the processing times required for the respective product configuration specifications. Table 3.11 shows the results from simulation experiments conducted for the four combinations of product configuration specifications. Fifty replications of the model were run for a simulation time of 1000 hrs each, with a warm-up period of 200 hrs during which no statistics were collected. The results shown are the average total units produced along with the 95 % confidence interval half-widths for each product configuration.

Table 3.11. System throughputs for constant known product configurations.

	Chronograph movement	**Standard movement**
Coloured	1040.78 ± 13.37	1049.54 ± 14.45
Standard colour	1059.16 ± 12.10	1060.72 ± 13.78

Qualitatively, these results were reasonable in relation to the processing times and processing requirements associated with the four different product configurations. Relatively speaking, the coloured unit with a chronograph movement would require the longest total processing time, since it would include the PVD process for all coloured parts as well as the additional parts required for the chronograph movement. In contrast, the standard colour unit with a standard automatic movement would require the shortest total processing time, since the PVD processing times reduced to zero and less parts compared to the chronograph variant meant shorter assembly times.

Dynamically, the animation of the simulation model was monitored while running, to determine whether the correct parts were generated for the different product configurations as aforementioned. It was observed that the parts associated with the chronograph movement were indeed generated when the chronograph specification was set, and vice versa. It was also observed that when the colour specification was set, the PVD time delay was evident. In the opposite case it was evident that the processing time through the PVD process was zero. The results from the experiments discussed here all reinforced the notion that the computer simulation model was programmed correctly and was verified.

3.3.5 Model validation

Validation of the simulation model is the final step in the model development process, and is defined as the confirmation that a model possesses an acceptable range of accuracy, consistent with its proposed application [73]. Model validation is a crucial step in the model development process, since this step provides credibility to the model in the opinions of the intended users of the model. However, it is usually impossible to prove the correctness of a model completely due to the fact that some or all of the system under investigation usually does not yet exist. For this reason certain techniques are used to prove the correctness of a model to an acceptable level [69].

Two independent techniques were employed to prove the validity of the model developed here. The output of the simulation model was compared to the output of the static model of the same system, under various conditions. Sensitivity analysis was also performed to identify the most important variables for the purpose of the model, and to ensure that they were modelled as accurately as possible. These techniques were selected due to the fact that the system under investigation does not exist, and thus no real-world data were available with which to compare the simulation output data.

3.3.5.1 Model comparison methodology and results

The dynamic model was compared to the static capacity planning model, as described in Section 3.2.1. The same order arrival rate and workstation capacities were used in the comparison. Table 3.12 shows the output of the static model compared to that of the dynamic model. Fifty replications were run for a total of 1000 hrs, with a warm-up period of 200 hrs during which no statistics were collected. In the table above "*h*" denotes the 95 % confidence interval half-widths of the simulation outputs.

Table 3.12. Comparison between workstation utilisations for static and dynamic models.

Workstation utilisation (%)			
Workstations	**Static Model**	**Simulation Model**	
		Average	***h***
4-Axis CNC Milling Machine (WS-M1)	76.04	76.15	0.68
Multi-purpose Grinding Machine (WS-M2)	29.17	29.21	0.26
Piercing and Blanking Machine (WS-M3)	7.29	7.31	0.07
Pad Printing Machine (WS-M4)	26.04	26.09	0.23
Feed-through CNC Lathe (WS-M5)	59.90	60.00	0.54
PVD Machine (WS-M6)	31.25	31.29	0.28
Stencil Gluing Machine (WS-M7)	6.25	6.26	0.06
3D Printing Machine (WS-M8)	56.25	56.35	0.51
Case Sub-assembly (WS-A1)	15.63	15.65	0.14
Time Sub-assembly (WS-A2)	16.67	16.71	0.16
Crown Sub-assembly (WS-A3)	8.33	8.34	0.08
Body Sub-assembly (WS-A4)	16.67	16.66	0.15
Final Assembly (WS-A5)	14.58	14.60	0.13

3.3.5.2 Model comparison discussion

The expected utilisation of the workstations predicted by the static model compared well with the scheduled workstation utilisations obtained from the simulation model. For all workstations, the expected utilisations fell within the 95 % confidence intervals. These results suggested that the simulation model was a good representation of the system under investigation at steady state. These results also confirmed that the static model was a useful tool to calculate the workstation capacities according to the order arrival rate.

3.3.5.3 Sensitivity analysis methodology

When performing a sensitivity analysis, the first step is to identify the most relevant input parameters, referred to as control variables in the context of sensitivity analysis [66]. The objective was to determine the maximum range of variation of the performance metrics in response to the maximum expected range of the input variables. The control variables identified were the order arrival rate, average order size, and the two product configuration parameters namely, colour and movement type which determined the wristwatch style. Multiple unit orders were also included here to investigate the effect on the system. The performance metrics, or responses, used in the sensitivity analysis included throughput in units per hour, average time in system in hours, and average number of units in the system.

To investigate the interaction between these input parameters a *k*-factorial experiment design was adopted, in which the extreme values of each input parameters was tested with the extreme values of every other input parameter [66]. Since there were four input variables there were $2^4 = 16$ scenarios to investigate. The ranges for the input parameters are shown in Table 3.13.

Table 3.13. Descriptions of maxima and minima of input parameters used in validation experiment.

Variable	Maximum (H)	Minimum (L)
Body colour	All units ordered coloured	No units ordered coloured
Movement	All units ordered with chronograph movement	All units ordered with standard movement
Units per order	Discrete probabilistic distribution with the following values: 1,0.85; 2,0.95; 3,0.9575; 4,0.965; 5,0.9728; 6,0.98; 7,0.985; 8,0.99; 9,0.995; 10,1.0	One unit per order for all orders
Order inter-arrival time	Exponential probabilistic distribution with mean value 0.4 hrs	Exponential probabilistic distribution with mean value 0.75 hrs

For the maximum value of units per order, the first value in each comma-separated pair was the number of units per order and the second was the probability of this value occurring, which was cumulative. In other words, the probability of the order size being one was 85 %, the probability of the order size being two was (0.95 – 0.85) x 100 % = 10 %, etc. In Table 3.13, the mean order inter-arrival times may seem to be reversed. However, the minimum and maximum values represented expected, and unfavourable states of the parameters, which corresponded to moderate mean inter-arrival times and low inter-arrival times, respectively.

3.3.5.4 Sensitivity analysis results

Each scenario was simulated using the experiment function in Simio. This function allows multiple scenarios of the same model to be run concurrently, without the graphical component. Control variables can also be initialised in the model, which can then be modified between scenarios. The average number of units in the system, the average time a unit spends in the system, and the throughput rate were recorded for each scenario. Fifty replications were run for each scenario and each replication ran for a simulation time of 1000 hrs in order to achieve steady state conditions. The outputs can be seen in the Table 3.14.

Table 3.14. Summarised simulation model validation experiment results.

Scenario	Colour	Chrono	Units/ Order	Order Inter-arrival Time	Avg NumberIn System [units]	Avg TimeIn System [hrs]	Through-put rate [units/hr]
1	H	H	H	H	1121.40	330.02	1.16
2	H	H	H	L	91.90	48.84	1.62
3	H	H	L	H	530.00	210.93	1.45
4	H	H	L	L	4.41	2.95	1.33
5	H	L	H	H	1033.18	307.73	1.28
6	H	L	H	L	51.55	26.86	1.72
7	H	L	L	H	486.67	192.45	1.55
8	H	L	L	L	3.77	2.50	1.33
9	L	H	H	H	1105.33	325.93	1.16
10	L	H	H	L	108.73	57.70	1.62
11	L	H	L	H	530.05	210.48	1.45
12	L	H	L	L	4.42	2.92	1.34
13	L	L	H	H	1073.34	313.96	1.26
14	L	L	H	L	52.30	27.18	1.72
15	L	L	L	H	470.00	186.42	1.56
16	L	L	L	L	3.59	2.37	1.33

Figure 3.7 - Figure 3.9 show graphical representations of the output variables from Table 3.14.

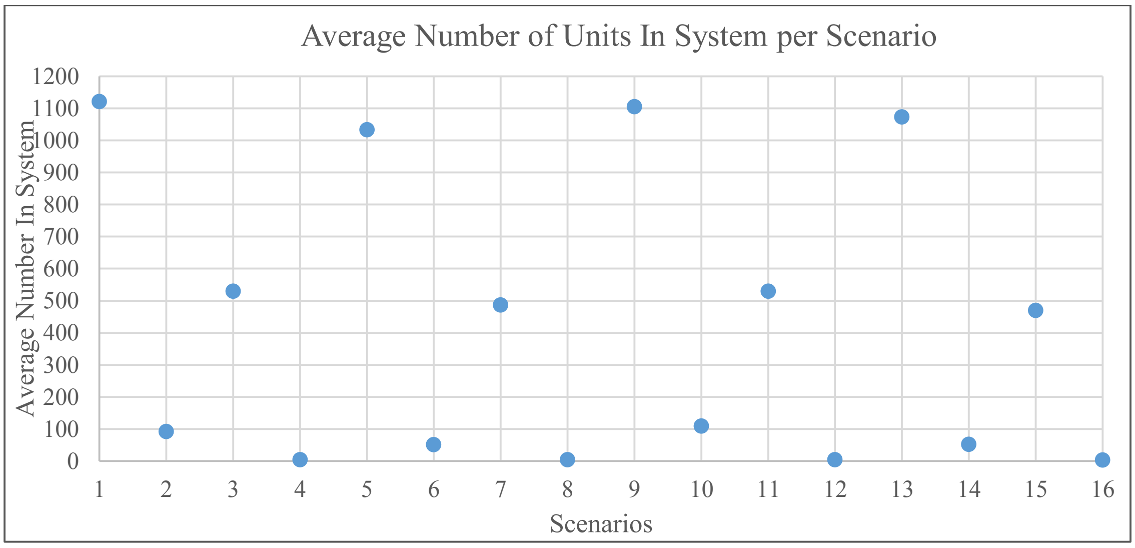


Figure 3.7. Average number of units in the system for all scenarios.

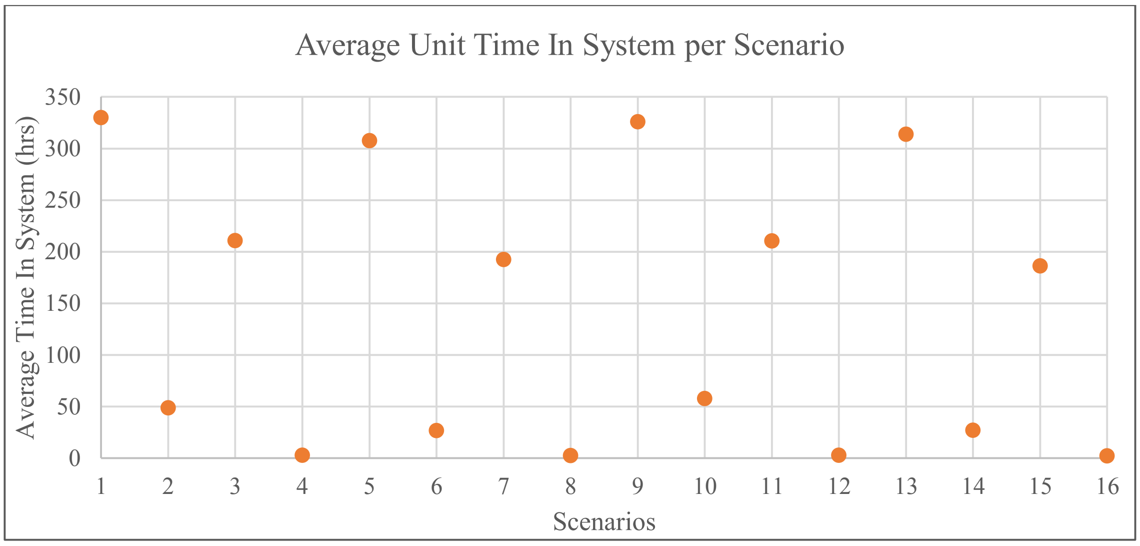


Figure 3.8. Average time a unit spends in the system for all scenarios.

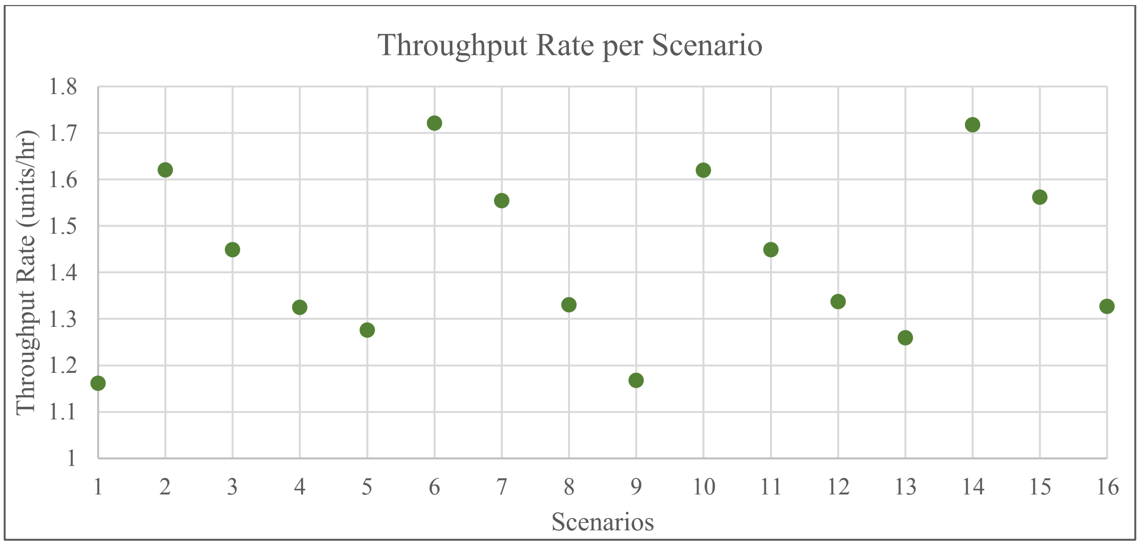


Figure 3.9. Throughput rates for all scenarios.

From Table 3.14 it can be seen that the maximum throughput rate occurred in Scenario 6, where the input variable combination was closest to the expected combination. The minimum throughput rate occurred in Scenario 1, where all variables were at their maximum values. Scenario 1 also produced the highest average number of units in the system, and the lowest occurred in Scenario 16. The longest average time a unit spent in the system occurred in Scenario 1, and the shortest occurred in Scenario 16. It could also be noted from Table 3.14 that the ranges of the number of units in the system and the time spent in the system per unit were much larger than the range of the throughput rate variable.

From the patterns observed in Figure 3.7 - Figure 3.9 the following conclusions can be made:

1. Order inter-arrival time had the most significant effect on the average number of units in the system as well as the average time in system.
2. The second most significant effect on the average number of units in system and average time in system was from the units per order variable.
3. The most significant effect on the throughput rate of the system was from an inverse combination of number of units per order and the order inter-arrival time.
4. The second most significant effect on the throughput rate was from the chronograph selection variable.
5. Irrespective of the levels of the two selection variables, the throughput rate was at its lowest when the units per order variable was high and the order inter-arrival time was low.

3.3.5.5 Model validation discussion

The fact that the highest number of units in the system as well as the longest time in the system both occurred in Scenario 1, when all input variables are at their maximum values, and the lowest number of units in the system and the shortest time in the system occurred in Scenario 16 when all input variables were at their minimum values provided good validation for the operation of the system as a whole.

These results will be used in the design of the algorithm for optimising scheduling of the manufacturing system. The average number of units in the system and the average time spent in the system were much more sensitive to variations in the input variables than the throughput rate. Therefore, these two variables should be the primary performance metrics in the design of the optimisation algorithm.

In the planning stage, the system would be designed from a resource point of view to produce a target throughput rate based on the expected arrival rate of orders. So, in theory, as the arrival rate increases above the expected value, the throughput should remain relatively stable. However, the orders will be blocked more and more causing longer times in-system as well as higher numbers of units in the system. This causes the order lead time to increase dramatically.

From the results achieved in the sensitivity analysis it was decided that the order size variable should be maintained at one in the case study to remove the effect of multiple unit orders on the system performance. This focussed attention on single unit order processing in terms of setup and processing times, which is more in line with the MCM, and simplified the analysis of system performance.

3.4 Chapter summary

This chapter described a product platform for a custom wristwatch product range from an MCM perspective. It documented the development of a static capacity planning model and a static APP model as well as a dynamic system model based on a production facility for producing custom wristwatches. The static and dynamic simulation models were successfully verified and validated using sound methods to ensure their credibility and successful use for system optimisation.

Chapter 4 Aggregate Production Planning Optimisation

Planning for production to meet demand forecasts allows a manufacturer to anticipate and plan for variations and is integral to the success of a manufacturing enterprise. This chapter presents the development of a novel algorithm for determining an optimal aggregate production plan for a wristwatch product range based on a common product platform. The performance of the new algorithm is compared with traditional planning strategies as well as with an optimisation algorithm based on an established Artificial Intelligence principle to address the research question posed at the start of the study.

4.1 Algorithm selection

A Biogeography-Based Optimisation (BBO) algorithm was selected for the optimisation of aggregate production plans, as explained in Chapter 2. According to Chase et al. [3] the main objective of an aggregate production plan is to identify the optimal combination of production rate, workforce level, and inventory on hand. The aggregate production planning problem is a time consuming exercise which involves a trial-and-error search of possible optimal solutions. A manual search method described by Chase et al. [3], known as the cut-and-try method, is based on costing various planning alternatives and selecting the best one. The BBO algorithm replicates and accelerates this process.

4.2 Biogeography-Based Optimisation principle

Biogeography-Based Optimisation is founded on the principle of biogeography, which is the study of species, their migration between habitats, and their extinction [33]. Habitats, also referred to as islands, are rated for their fitness for supporting life using a term known as the habitat suitability index (HSI). A high HSI is associated with a habitat that is fit to support a large number of species, whereas a habitat with a low HSI is only fit to support a small number of species.

Migration is driven by the numbers of species within the habitats of the system, which determines the immigration and emigration rates of the species in each habitat from a graph similar to that shown in Figure 4.1 [33]. In other words, a habitat with a high HSI will contain a large number of species which means it will necessarily have a high emigration rate, μ. In contrast, a habitat with a low HSI will contain a small number of species and so will have a high immigration rate, λ.

As migration occurs, the increasing species diversity of the low HSI habitats will cause their HSIs to increase and the reduction in species diversity in the high HSI habitats will cause their HSIs to reduce. This will continue until the numbers of species reach an equilibrium, S_0. In reality, only a small group of individuals migrate between habitats leaving a population behind in their original habitat. However, with BBO entire populations are assumed to migrate. This is necessary because with BBO the species represent independent variables of the objective function, which replace each other between the set of candidate solutions, or habitats.

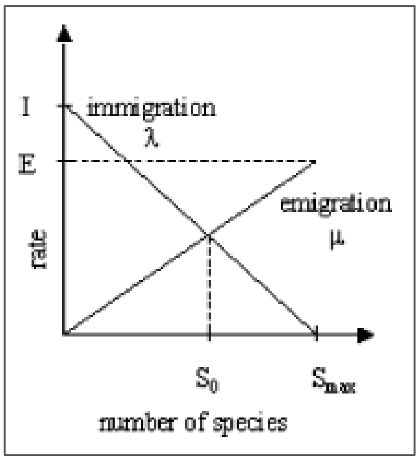


Figure 4.1. Single habitat species model. Adapted from [2].

With BBO, λ_i represents the probability that an independent variable, or species, in the i-th habitat will be replaced [33]. And the probability, P, of a given species, in habitat x_j, emigrating from x_j to replace the emigrating species in habitat x_i is calculated using Equation 4.1.

$$P(x_j) = \frac{\mu_j}{\sum_1^N \mu_k} \quad 4.1$$

Where k = 1, 2, 3, . . . N, and N is the number of habitats in the system. This is based on the principle of fitness proportionate selection in which selection pressure is proportional to the fitness of the candidates [74].

Biogeography-Based Optimisation can also incorporate mutation, which represents the introduction of random disturbances to the HSIs of habitats [33]. The method of deciding whether a given species, or independent variable, in a certain habitat should be mutated is to compare a user-defined mutation probability parameter with a randomly generated number in the same range and mutating the variable by randomly adjusting its value within its range.

4.3 Biogeography-Based Optimisation algorithm development

A BBO algorithm was developed for optimisation of an aggregate production plan for a case study of a wristwatch product range based on a common product platform. This work has been based on the static system models documented in Section 3.2 and their associated production data. This section documents the development of the BBO algorithm to address the APP problem. It describes the implementation of the algorithm as well as its inputs and initialisation.

4.3.1 Implementation

The BBO algorithm developed here consisted of an inner and an outer loop. The outer loop repeated for a user-defined number of iterations, and the inner loop stepped through the user-defined number of habitats, or candidate solutions. The population of possible solutions, i.e. aggregate production plans, was generated at the initialisation of the algorithm, based on production plan parameters. The parameters were calculated based on randomly generated variables. These were an inventory-to-production ratio, and a workforce level parameter.

The inventory-to-production ratio and workforce level variables were used to vary the inventory to production ratio and workforce level from month to month, respectively. Equations 4.2 and 4.3 show the expressions used for these two variables. The expression for inventory-to-production ratio increased or decreased the ratio of inventory to production for each month by up to 17 %, while the workforce level variables parameter added or subtracted up to two workers to or from the workforce from one month to the next.

$$\textit{Inventory-to-Production-Ratio} = ((\textit{float})(\textit{rand}()\ \%\ 50 - 25))\ /\ 150 \qquad 4.2$$

$$\textit{Workforce-Variable} = \textit{round}(((\textit{rand}()\ \%\ 2) * 2 - 1)*((\textit{float})(\textit{rand}()\ \%\ 2))) \qquad 4.3$$

From these two variables all the plan parameters were calculated. The plan parameters included production levels, workforce levels, and inventory levels for each month in the planning period. In the context of the BBO principle, each plan represented a habitat, and each plan parameter represented a species. The algorithm calculated the cost of each plan, then ranked and sorted them according to their cost. Immigration and emigration rates for each plan were calculated based on the rank of the plan, i.e. the plan with the lowest cost had the highest emigration rate and the plan with the highest cost had the lowest emigration rate. The immigration rates were calculated by subtracting the emigration rates from one, as shown in Equation 4.4.

$$\mu = 1 - \lambda \qquad 4.4$$

Iterations began by placing the two lowest cost plans into an elitist matrix for replacing the two highest cost plans in the next iteration. The inner loop then stepped through the habitats, or plans. For each iteration, production plan parameters of each plan were migrated based on a probability calculated from the HSI and mutated based on a predefined probability. The production plan cost was used as the HSI here. The newly migrated and mutated plan were then sorted from lowest cost to highest.

The validity of the lowest cost plan was checked and if the plan was not valid, that iteration was discarded and repeated with new migrations and mutations. If the lowest cost plan was valid the algorithm proceeded with storing the lowest cost plan of that iteration in a matrix of low cost plans. The validity requirement set in this instance was for the ending inventory level in the final month of the planning period to be greater than zero.

The algorithm then replaced the two highest cost plans with the two elite plans stored in the previous iteration, and placed the lowest cost plan for that iteration into the low cost plans matrix. The low cost plans matrix held the lowest cost valid plans from each iteration. Once the maximum number of iterations had been completed, the absolute lowest cost valid plan was extracted from the low cost plan matrix for outputting. Algorithm 4.1 shows the structure and pseudocode of the proposed BBO algorithm.

Algorithm 4.1. Biogeography-Based Production Planning Optimisation Algorithm.

```
     Input: $ProblemSpace$; $iterations_{max}$; $habitats_{max}$; $elites_{max}$; $mutation_{Prob}$
     Output: $Plan_{best}$
1    Create $habitats_{max}$ x $plans$;
2    Calculate cost of $plans$;
3    Sort plans based on cost: Low – High;
4    Calculate migration rates based on $habitats_{max}$;
5      for $i$ = 1 to $iterations_{max}$ do
6      $elitePlans$ <-- $plans[elites_{max}]$;
7        for j = 1 to $habitats_{max}$ do
8        Migrate $plan$ parameters based on migration rates;
9        Mutate $plan$ parameters based on $mutation_{Prob}$;
10       Calculate cost of new $plans$;
11       Sort plans based on cost: Low –> High;
12       Check validity of $plans[1]$;
13       if $plans[1]$ is invalid then
14         Goto Step 8;
15       else
16         $plans_{best}$ <-- $plans[1]$;
17       end
18     Sort $plans_{best}$ based on cost: Low – High;
19     Highest cost $plans$ <-- $elitePlans$;
20     $elitePlans$ <-- $plans[elites_{max}]$;
21     $Plan_{best}$ <-- $plans_{best}[1]$;
22   End
23   return $Plan_{best}$
```

4.3.2 Inputs

As listed in Algorithm 4.1 the BBO algorithm inputs included the problem space, maximum number of iteration, maximum number of habitats, maximum number of elite habitats, and the mutation probability. The term ‘problem space’ refers to the structure and parameters of the aggregate production plan, including parameters as discussed in Section 3.2.2. Before the BBO algorithm was initialised, all plan parameters were initialised, and the values were read in from two text files. Table 4.1, Table 4.2, and Table 4.3 show the plan parameters and the values for which the optimal aggregate production plan was developed.

Table 4.1. Monthly aggregate production plan parameter values.

Monthly Production Parameter	Value					
	Jan	Feb	Mar	Apr	May	Jun
Working days	22	19	21	21	22	20
Demand forecast [units]	1400	1150	1260	1240	1380	1280

Table 4.2. One-off aggregate production planning parameter values.

One-off Production Parameter	Value
Production time	122 minutes/unit
Planning horizon	6 months
Safety stock	25 % of monthly demand
Starting inventory	100 units
Initial workforce	16 workers

Table 4.3. Aggregate production planning cost parameter values.

Cost Input Parameter	Value
Cost of holding inventory	\$15.00/unit
Stock-out cost	\$20.00/unit
Worker hiring cost	\$500.00/worker
Worker lay-off cost	\$750.00/worker
Manufacturing cost (regular time)	\$20.00/hr/worker
Overtime cost	\$30.00/hr/worker
Downtime cost	\$10.00/hr

Table 4.4 gives the descriptions and values used for the BBO algorithm input parameters.

Table 4.4. Biogeography-based optimisation algorithm input parameters.

Input	Description	Value
iterations$_{max}$	Maximum number of iterations to complete	150
habitats$_{max}$	Number of aggregate production plans to create	100
elites$_{max}$	Number of elite aggregate production plans to keep after each iteration	2
mutation$_{Prob}$	Probability to use for mutating aggregate production plan parameters	0.05

The values of the parameters shown in Table 4.4 were found through trial-and-error by comparing the results of the BBO algorithm from subsequent runs of the algorithm. The values given were found to produce the best results, within a reasonable search space and timeframe. However, because they were found by trial-and-error it cannot be categorically stated that this combination of parameters produces the optimal plan.

4.3.3 Initialisation

Before initialisation of the BBO algorithm, three plans were developed based on the standard planning strategies discussed in Section 3.2.2.1. The lowest cost plan out of these was used as the starting point for the BBO algorithm to compare its results. For the problem at hand the lowest cost plan based on a standard planning strategy was found to be one based on the level workforce – varied production strategy, that is, stable workforce with varying levels of inventory. Table 4.5 shows the calculated plan parameters for this plan, including the total cost of production.

Table 4.5. Best production plan based on standard planning strategy.

	Jan	Feb	Mar	Apr	May	Jun
Starting Inventory [units]	100	85	131	193	275	280
Demand Forecast [units]	1400	1150	1260	1240	1380	1280
Safety Stock [units]	350	288	315	310	345	320
Production Req.'s [units]	1650	1088	1287	1235	1415	1255
Actual Production [units]	1350	1196	1322	1322	1385	1259
Ending Inventory [units]	85	131	193	275	280	259
Workforce [workers]	16	16	16	16	16	16
Monthly Cost [$]	56 320	48 640	53 760	53 760	56 320	51 200
Total Cost [$]	320 000					

The BBO algorithm programme was written in C++, using Microsoft Visual Studio Express 2013, as a Win32 Console Application. The programming code for the BBO algorithm can be found in Appendix D. Solution time for the BBO algorithm was approximately 10 s.

4.4 Aggregate Production Planning optimisation benchmark algorithm

To set a benchmark for assessing the performance of the new BBO algorithm, a Simulated Annealing (SA) algorithm was developed and applied to the same case study production system. Simulated Annealing was used as a benchmark because it is a well-established Artificial Intelligence method and has been widely used in manufacturing systems optimisation research [20], [75]. The SA algorithm is based on the principle of annealing in metallurgy, where a metal is heated to a high temperature and then cooled, in a slow and controlled manner, in order to produce a particular molecular structure in the material [17]. Simulated Annealing is a search scheme that incorporates an exploration component and an exploitation component.

The initial state of the system consists of an initial candidate solution, which is randomly generated, the system temperature variable, *T*, and the temperature schedule parameter. From the initial state, the neighbourhood of the candidate solution is explored, by generating new candidate solutions through varying the solution parameters based on the current system temperature. This is performed for a certain number of iterations before lowering the system temperature based on the temperature schedule. At each iteration the new candidate solution is compared to the best solution found so far, and if the new solution is better, it gets stored as the new best solution.

The temperature schedule is a fraction multiplied by the current temperature. The best candidate solution at the current temperature is kept as the starting point for the next round of iterations at the next system temperature. As the system temperature decreases, the search area around the current best candidate solution decreases, which enhances the exploitation component of the search. However, to avoid the search becoming trapped at a local optimum, an acceptance probability is calculated and compared to a randomly generated fraction. Equation 4.5 shows the expression for calculating the acceptance probability, *P*.

$$P = e^{(S_c - S_n)/_T} \qquad 4.5$$

Where S_c is the current best solution objective function value, S_n is the newly calculated solution objective function value, and T is the current system temperature [76]. If the acceptance probability is greater than the randomly generated fraction, the new candidate solution replaces the current best candidate solution. This step only takes place if the new candidate solution is not better than the current best candidate solution. In other words, the acceptance probability represents the probability of a worse solution being accepted as a possible optimal solution. However, it can be seen from Equation 4.5 that the probability tends to zero as the temperature decreases, since the numerator of the exponent will always be negative.

4.5 Simulated Annealing algorithm development

This section documents the development of the SA algorithm to set a benchmark for testing the performance of the BBO algorithm. It describes the implementation of the algorithm as well as its inputs and initialisation.

4.5.1 Implementation

The SA algorithm used in this instance consisted of two nested loops. The outer loop, known as the temperature loop, represented the slow cooling of the system under consideration. The inner loop, known as the iteration loop, explored the search space at each temperature setting. The algorithm started at a user-defined maximum temperature and carried out a set number of iterations. The number of iterations per temperature step was user-defined. The amount that the temperature was reduced by was determined by the temperature schedule, which was also user-defined.

At each iteration, a new plan was generated using randomly generated variables along with the parameters of the plan generated in the previous iteration for calculating the parameters of the new plan. The variables used here were the same as those discussed in Section 4.3.1, namely, inventory-to-production ratio, and workforce level variable.

The new plan was evaluated for its validity. If the ending inventory of the last month in the planning period was positive the plan was valid and could be used. If the plan was valid then the cost of the plan was calculated and compared to that of the lowest cost plan for the current temperature setting. If the new plan cost was lower than the current lowest cost plan, then it replaced the current lowest cost plan. The new plan cost was also compared to the overall lowest cost plan, and if it was lower, it replaced the overall lowest cost plan.

The acceptance probability for accepting a plan that had a higher cost than the lowest cost plan for the current temperature setting was calculated using the cost of the new plan, the cost of the lowest cost plan for the current temperature setting and the temperature schedule. The acceptance probability was then compared to a randomly generated number in the same range as the acceptance probability. If the acceptance probability was greater than the random number then the new plan replaced the lowest cost plan for the current temperature setting.

Once the maximum number of iterations were completed, the algorithm stepped down to the next temperature setting, according to the temperature schedule, to carry out the next round of iterations. Algorithm 4.2 shows the structure of the SA algorithm designed for this specific application.

Algorithm 4.2. Simulated Annealing Production Planning Optimisation Algorithm

```
    Input: $ProblemSpace; iterations_{max}; temp_{max}; temp_{min}; tempSched$
    Output: $Plan_{best}$
1   $Plan_{current}$ <-- Best Standard Strategy Plan;
2   $Plan_{best}$ <-- $Plan_{current}$;
3   while $temp > temp_{max}$ do
4       for $i = 1$ to $iterations_{max}$ do
5           Create $Plan_{new}$ based on $Plan_{current}$
6           Check validity of $Plan_{new}$
7           Calculate cost($Plan_{new}$)
8           if cost($Plan_{new}$) < cost($Plan_{current}$) then
9               $Plan_{current}$ <-- $Plan_{new}$
10              if cost($Plan_{new}$) < cost($Plan_{best}$) then
11                  $Plan_{best}$ <-- $Plan_{new}$
12              End
13          else if AcceptanceProbability(temp, cost($Plan_{new}$), cost($Plan_{current}$)) > random() then
14              $Plan_{current}$ <-- $Plan_{new}$
15          End
16      temp <-- temp x tempSched
17  End
18  return $Plan_{best}$
```

4.5.2 Inputs

As listed in Algorithm 4.2, the algorithm inputs included the problem space, maximum number of iteration, maximum temperature, minimum temperature, and temperature schedule. As with the BBO algorithm, the term 'problem space' refers to the structure and parameters of the aggregate production plan, including parameters as discussed in Section 3.2.2. Before the algorithm was initialised, all plan parameters were initialised, and the values were read in from two text files. The same plan parameter values as shown in Table 4.1, Table 4.2, and Table 4.3 were used in the development of the SA algorithm.

The SA algorithm was run for 5 000 iterations at each temperature step from 800 000 down to 50 000 at a temperature schedule of 0.99. In other words, at each temperature step the system temperature was equal to 99 % of the system temperature in the previous temperature step. Table 4.6 gives the descriptions and values used for the SA algorithm input parameters.

Table 4.6. Simulated Annealing optimisation algorithm input parameters.

Input	Description	Value
$Iterations_{max}$	Maximum number of iterations to complete per temperature loop	5000
$temp_{max}$	Starting temperature for the temperature loop	800 000
$temp_{min}$	Ending temperature for the temperature loop	500 000
tempSched	Multiplier for calculating temperature for next iteration of temperature loop	0.99

The values of the parameters shown in Table 4.6 were also found through trial-and-error by comparing the SA results of subsequent runs of the algorithm. These values were found to produce the best results, within a reasonable search space and timeframe. However, because they were found by trial-and-error it cannot be categorically stated that this combination of parameters produces the optimal plan.

4.5.3 Initialisation

The SA optimisation algorithm also used the best aggregate production plan based on a standard planning strategy as a starting point, as discussed in Section 3.2.2.1. This plan is shown in Table 4.5 along with the total production cost for that particular plan. The SA algorithm was also coded in C++, using Microsoft Visual Studio Express 2013, as a Win32 Console Application. The programming code for the SA algorithm can be found in Appendix D. Solution time for the SA algorithm was approximately 15 s.

4.6 Results

The results from the BBO and SA algorithms were compared to measure the performance of the new BBO algorithm. The results were compared from a cost convergence perspective as well as the final aggregate production plans produced. All programming was carried out and run on a PC with 8 GB of RAM and an Intel Core i7-4700QM CPU running at 2.4 GHz.

4.6.1 Optimal aggregate production plan cost convergence

Figure 4.2 shows the convergence of the overall best plan cost by the SA algorithm, within the first 5 % of the temperature schedule to near optimal plan, and a final drop to the output value at approximately 50 % through the temperature schedule.

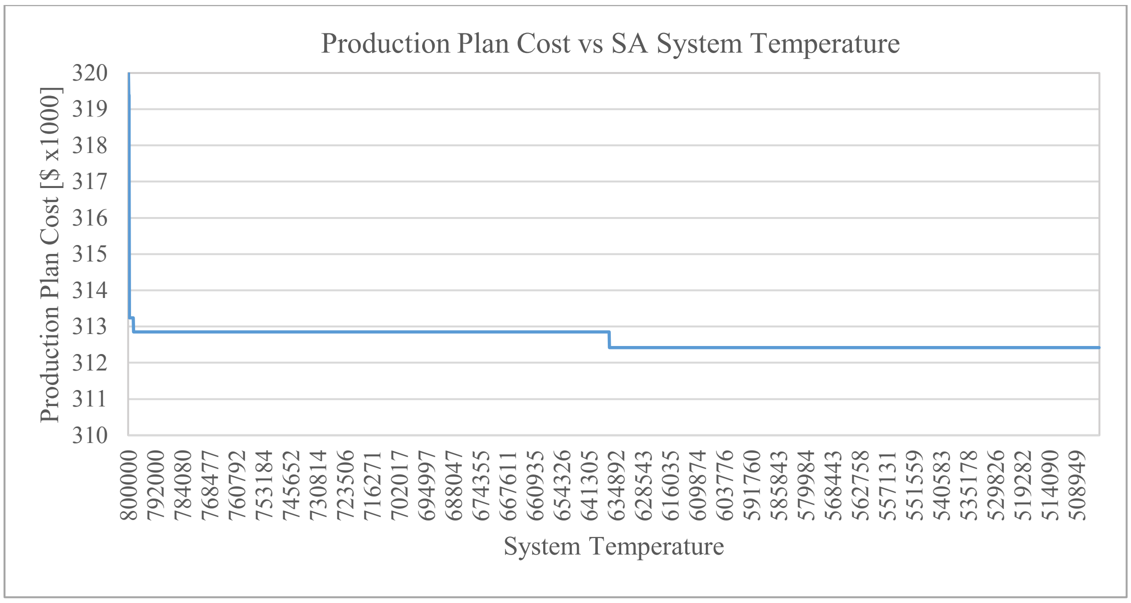


Figure 4.2. Best plan cost convergence using the SA algorithm.

Figure 4.3 shows the convergence of the lowest plan cost for the BBO algorithm. From this figure it can be seen that the algorithm converged to the final value within the first ten iterations, or approximately 7 % of the total run length.

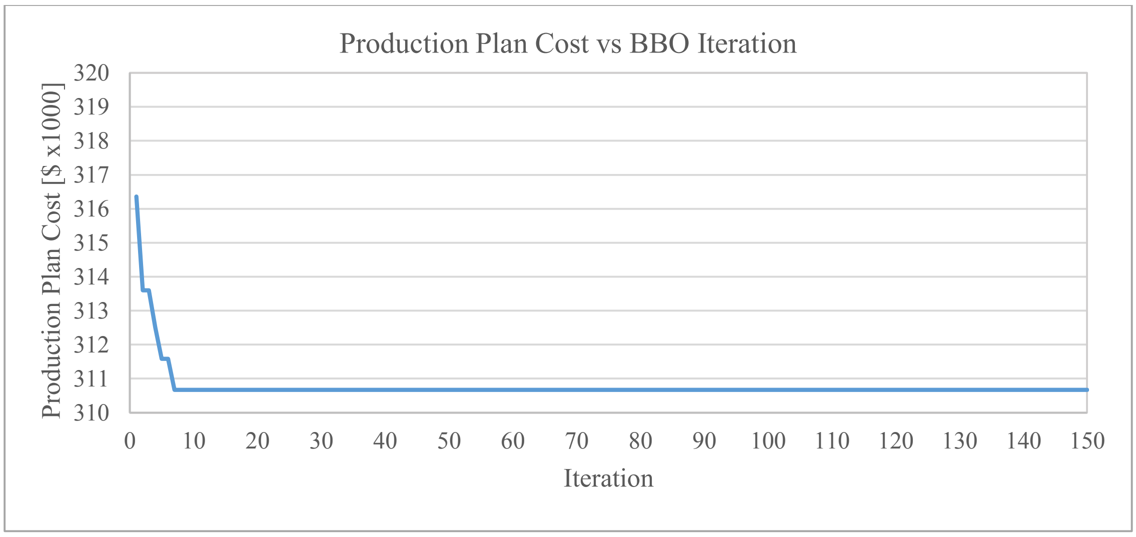


Figure 4.3. Best plan cost convergence using BBO algorithm.

4.6.2 Optimal aggregate production plan comparison

The SA algorithm produced the production plan as shown in Table 4.7. This production plan showed an improvement of 2.37 % of total cost compared to the best standard strategy plan. This translated to an average saving of $ 1 263.33 per month, or $ 15 160.00 annually. The SA algorithm produced a plan with lower inventory levels and a relatively small workforce turnover, while tracking the demand forecasts well compared to the best standard strategy plan.

Table 4.7. Optimal production plan generated by way of SA algorithm.

	Jan	Feb	Mar	Apr	May	Jun
Starting Inventory [units]	100	85	131	28	27	32
Demand Forecast [units]	1400	1150	1260	1240	1380	1280
Safety Stock [units]	350	288	315	310	345	320
Production Req.'s [units]	1650	1353	1444	1522	1698	1568
Actual Production [units]	1385	1196	1157	1239	1385	1259
Ending Inventory [units]	85	131	28	27	32	11
Workforce [workers]	16	16	14	15	16	16
Monthly Cost [$]	56 320	48 640	48 540	50 900	56 820	51 200
Total Cost [$]	312 420					

The BBO algorithm produced the production plan as shown in Table 4.8. This production plan showed an improvement of 2.92 % over the best standard strategy based plan. This translated to an average saving of $ 1 555.00 per month, or $ 18 660.00 annually. The BBO algorithm was able to produce a plan with less disruption to the workforce, with only a single change from one month to the next over the entire planning period. It was also able to produce a plan that tracked the production requirements more closely than the SA algorithm.

Table 4.8. Optimal production plan generated by way of BBO algorithm.

	Jan	Feb	Mar	Apr	May	Jun
Starting Inventory [units]	100	85	131	193	192	110
Demand Forecast [units]	1400	1150	1260	1240	1380	1280
Safety Stock [units]	350	288	315	310	345	320
Production Req.'s [units]	1650	1353	1444	1357	1533	1490
Actual Production [units]	1385	1196	1322	1239	1298	1180
Ending Inventory [units]	85	131	193	192	110	10
Workforce [workers]	16	16	16	15	15	15
Monthly Cost [$]	56 320	48 640	53 760	51 150	52 800	48 000
Total Cost [$]	310 670					

In comparing the total costs of the production plans produced by the SA and BBO algorithms, it was found that the BBO algorithm was able to produce a lower cost plan, at $ 310 670.00, than the plan produced by the SA algorithm, at $ 312 420.00. The total saving in production cost incurred by the BBO algorithm over the SA algorithm was 0.5 %.

4.7 Discussion

Both the SA and BBO algorithms were able to produce production plans that were superior to the best standard strategy, which was the stable workforce – varying production strategy as defined by Chase et al. [3]. When comparing the plans produced by the two optimisation algorithms, the BBO algorithm produced plan had a lower overall cost than the SA algorithm produced plan. The differences in cost savings between the stable workforce – varying production plan, SA produced plan and BBO produced plan fell well within the results of Baykasoglu [25] in their comparison between goal programming and heuristic-based methods.

Both the SA algorithm and the BBO algorithm produced plans that turned out to be mixed plans, i.e. a combination of two or more of the standard strategies as defined by Chase et al. [3]. This was expected as the probability of stochastic algorithms such as these producing standard strategies would be very low. This also aligned well with the prediction by Chase et al. [3] that mixed strategies were usually better than standard strategies.

It is interesting to note that the best standard strategy plan was based on a constant workforce, whereas the SA generated plan has a non-constant workforce. The algorithm was designed to alternate between constant and non-constant workforce plans based on parameters, built into the algorithm, that vary as the algorithm progressed and thus converged to the optimal selection between constant and non-constant. The lower cost plan calculated by the SA algorithm showed that there was room for improvement over the standard planning strategies however small this may have been.

When comparing the workforce levels of the plans produced by the SA algorithm and the BBO algorithm it was found that the BBO algorithm also produced a production plan that involved a non-constant workforce. However, the BBO algorithm was able to produce a plan with less disruption to the workforce, with only a single change from one month to the next over the entire planning period. This was a good result from a human resources perspective, as disruptions to the workforce are not taken lightly from employees' perspectives. However, it is sometimes the only way for a manufacturer to be economically competitive.

The stable workforce – varying production strategy produced a plan with a fairly high ending inventory, whereas both the SA and BBO algorithms produced plans with very low ending inventories. One of the main differences between the optimised plans was that the monthly ending inventories of the SA algorithm plan dropped drastically in the third month. This seems to indicate that the cost of holding inventory played a significant role. However, the BBO plan held the monthly ending inventories high until the very last month. The result of the more stable monthly ending inventories of the BBO produced plan was more stable production requirements which led to more stable actual production.

From a programming perspective, the BBO algorithm contained more steps than the SA algorithm. However, the BBO algorithm was less computationally intensive than the SA algorithm. The SA algorithm performed 5 000 iterations at each step in the temperature schedule, which required approximately 1000 loops. In contrast to this the BBO algorithm only performed 150 iterations, which involved stepping through 100 habitats at each iteration. Furthermore, even though the BBO algorithm was less computationally intensive, it converged to its final output much quicker than the SA algorithm.

4.8 Chapter summary

This chapter presented the development of a novel BBO algorithm for determining an optimal aggregate production plan and compared its performance with that of standard APP strategies as well as an SA algorithm. The BBO and SA principles were explained. The development processes for both algorithms were documented including critical algorithm parameters and structures. Results were presented, discussed, and interpreted with reference to relevant literature.

Chapter 5 Production System Operation Optimisation

Based on research into production system control and scheduling policies, presented in Section 2.4, a new distributed dynamic scheduling strategy is proposed here, incorporating existing shop floor-level scheduling policies and system-level information for optimising the system performance. This chapter documents the development of the strategy. This includes application of the strategy to a hypothetical custom wristwatch production system as a case study to validate its performance and to address the research question posed at the start of the study.

5.1 Selection rule approach

The new scheduling strategy was based on traditional selection rules as defined in Section 2.4, due to their wide spread use and relatively low computational demand [49]. Research into traditional selection rules showed that these rules are typically based on order due dates, local processing times, and overall processing times [49], [77]. Based on these findings, commonly used rules were selected and used in the development of the scheduling strategy:

1. Earliest Due Date (EDD)
2. Shortest Processing Time (SPT)
3. Most Processing Time Remaining (MPTR)
4. Least Processing Time Remaining (LPTR)

Each of these rules addresses a certain general performance target. Earliest Due Date is aimed at minimising order lateness. Shortest Processing Time aims to maximise local throughput. Most Processing Time Remaining is aimed at maximising throughput by prioritising entities that are early in their processing sequences, while LPTR is aimed at maximising throughput by prioritising entities that are later in their processing sequences. In this context, the EDD rule is implemented based on the time that the orders were released to the production system. This assumption removes uncertainty imposed on delivery by handling and shipping and bases the due date on the date on which the order was placed. This also assumes that no priority can be assigned to orders placed after others.

These selection rules formed the basis of the development of the scheduling optimisation strategy as documented in the rest of this chapter. Commonly used selection rules were chosen to focus attention on the implementation of the strategy itself and to effectively test its performance, in conjunction with traditional base rules. In addition to the new strategy, each selection rule was implemented individually in Simio [68] and a separate experiment was run for each in order to set a benchmark for the performance of the new scheduling strategy. Further details on the testing methodology can be found in Section 5.2.1.5.

5.2 Distributed Dynamic Selection Rule Strategy development

From the review of recent literature on production system scheduling it was found that dynamic scheduling using Artificial Intelligence (AI) and other heuristics perform better than single selection rules applied globally during production [52], [53], [55]. All literature were found to be focussed on system-wide application of dynamic scheduling methods. Based on this finding, it was hypothesised that the implementation of a scheduling strategy which dynamically switched selection of individual workstation rules based on a novel principle may outperform the implementation of a single selection rule across the system as a whole. Such a strategy has been developed and was modelled on the operation of the Harmony Search (HS) algorithm.

The HS algorithm is typically used for optimisation of problems with static search spaces by modifying decision variables that make up the objective function through consulting what is known as the "Harmony Memory" at a specific rate [57]. In addition to the harmony consideration operation, it involves a pitch adjustment operation, which is typically designed to act as a randomisation operation to avoid local optima in the search space [57]. The structure of the scheduling strategy under discussion was inspired by that of HS, except that the strategy was designed to automatically adapt to a dynamic search space.

5.2.1 Methodology

To test the hypothesis, an environment for testing and analysing various selection rule implementations was developed. Chapter 3 described the development of a dynamic simulation model for this purpose. The simulation model represented a flexible flow shop manufacturing system and was developed based on a hypothetical product range which consisted of a men's wristwatch product platform with a range of customisable parts and features. These simulation models were created in Simio [68]. Five instances of the model were created, four of which each implemented an individual traditional selection rule, as discussed in Section 5.1, across the entire system. The fifth instance implemented the Distributed Dynamic Selection Rule Strategy (DDSRS) developed here. The results from these five models were compared, to evaluate the performance of the DDSRS.

5.2.1.1 Selection rule decision tree

The DDSRS adopted a decision tree approach to determine when to switch between selection rules at each processing workstation. A decision tree approach was implemented because this approach best suited the nature of the implementation space. In order for the strategy to be effective, it needed to align with the operation of the selection rules for which it was designed.

The discrete nature of the decision tree suited the discrete time decisions required by a selection rule.

Selection rules function on an event basis and query nominated variables when the next item needs to be selected for processing. Implementing the DDSRS in the form of a decision tree allows it to be implemented as part of the selection rules, rather than in addition to these rules. The decision tree queries certain variables at specific times during the production run, as necessary, at workstation level as well as at system level. These variables are discussed in the following subsection.

The strategy was implemented for the processing workstations and not the assembly workstations. This was because the assembly workstations needed to combine parts according to their order numbers in order to produce units with the correct features and specifications as specified by the customer. Simio does not allow for implementation of selection rules at assembly workstation objects. The order of processing in the assembly workstations was governed by the order of arrival of the parent parts to the assembly workstations.

5.2.1.2 Scheduling strategy decision variables

The DDSRS was designed to evaluate the state of the production system and the local buffer at discrete intervals by querying system-level performance metrics as well as the behaviour of the buffer level of the workstation where it is implemented. The three decision variables that were identified for monitoring and querying through the production run were:

1. Order Arrival Period
2. Order Work In Process (WIP)
3. Local Buffer Level

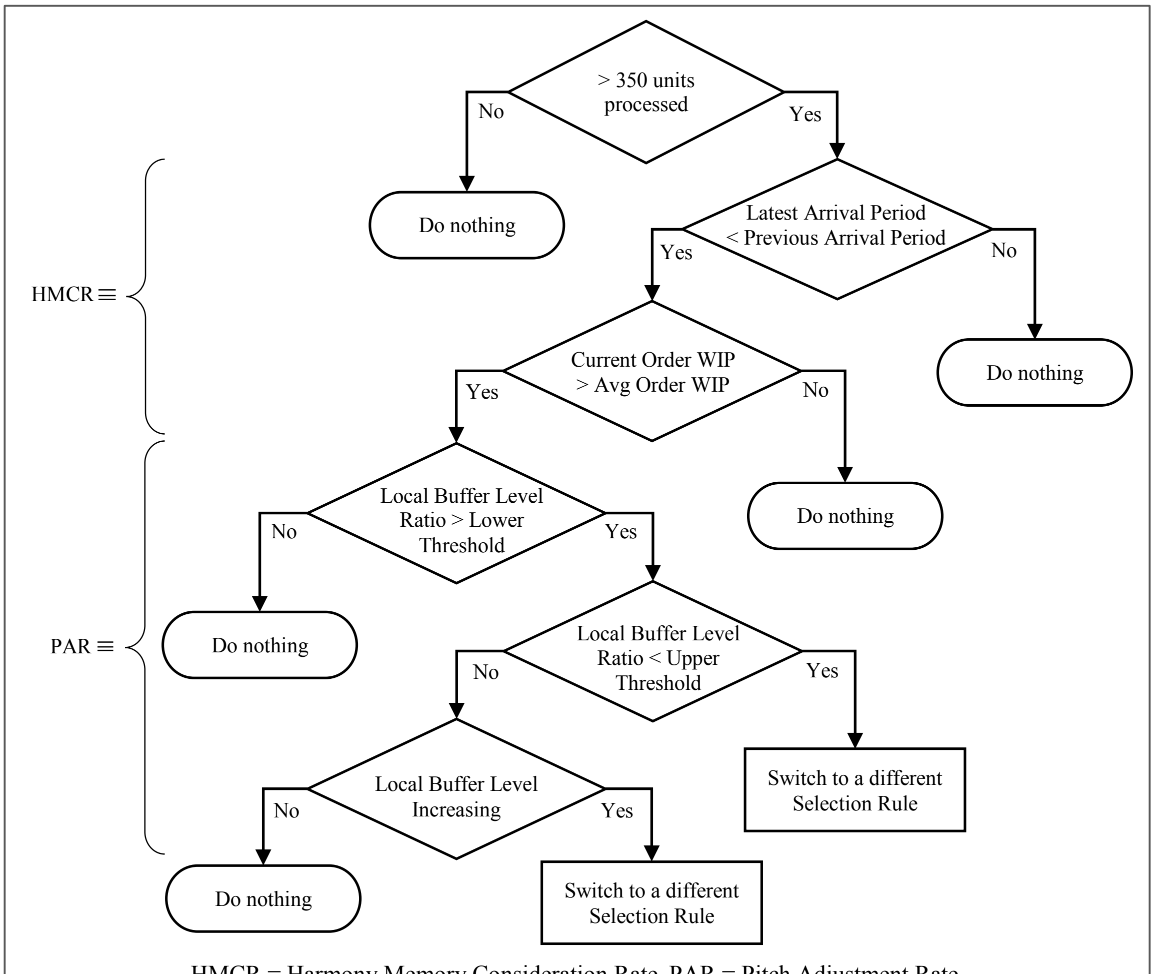
The Order Arrival Period variable was at the top level of the decision tree, and was used to determine the relative trend of the order arrivals as a discretised system variable. The decision tree compared the latest arrival period with the most recently stored value of the same variable and made a decision to activate the rest of the decision tree or not. The Order Arrival Period was defined as the time elapsed between the arrivals of a certain number of orders, which was adjusted depending on the desired response time and nature of the order arrival process.

At the mid-level of the decision tree was the Order WIP variable, which was designed to evaluate the state of the queue of orders waiting to be filled at the order completion station at the end of the production system. The current order WIP was compared to the moving average order WIP at the particular time in the production run. This was used as a further check on the state of the production system performance, by determining whether the orders waiting was in excess of the average up to that point.

At the lowest level of the decision tree, the local buffer level was evaluated. At this level the aim was to determine the most effective course of action for the state of the local buffer level given the path followed to that point through the decision tree. The local buffer was evaluated for its level relative to its moving average as well as the trajectory of its level. It was proposed that, in addition to the buffer level ratio, the trajectory of the local buffer level gave a further indication of the effectiveness of the current selection rule in place.

5.2.1.3 Distributed Dynamic Selection Rule Strategy structure

Figure 5.1 shows the decision tree implemented for determining when to switch selection rules at the individual workstation level of the production system. The upper most decision in the tree represents a warm-up period to allow the system to build data history for the decision tree to base its decisions on. Before 350 orders have been processed the default selection rule remains in place at all processing stations.


> 350 units processed
No
Yes
Do nothing
Latest Arrival Period < Previous Arrival Period
Yes
No
HMCR ≡
Current Order WIP > Avg Order WIP
Yes
No
Do nothing
Local Buffer Level Ratio > Lower Threshold
No
Yes
Do nothing
PAR ≡
Do nothing
Local Buffer Level Ratio < Upper Threshold
No
Yes
Local Buffer Level Increasing
No
Yes
Switch to a different Selection Rule
Do nothing
Switch to a different Selection Rule

HMCR = Harmony Memory Consideration Rate, PAR = Pitch Adjustment Rate

Figure 5.1. Distributed Dynamic Selection Rule Strategy decision tree structure.

The length of the warm-up period was determined experimentally, by monitoring the instantaneous order lead time compared to the moving average order lead time. With no selection rule in place, i.e. a First In First Out (FIFO) queue ranking rule in place, the first time the instantaneous order lead time went below the moving average was at approximately 350 orders.

The decision based on the Order Arrival Period variable, i.e. the time elapsed between the arrivals of a certain number of orders, was designed to respond to the volatility of the order arrival process. If the latest recorded arrival period was smaller than or equal to the previously recorded arrival period then the decision tree moved to the Order WIP decision point. If this was not the case then it held the current selection rule in place.

The Order WIP decision point compared the current number of orders waiting to be filled with the moving average of this variable up to that point. This was designed to respond to the state of the order waiting queue. This level of the decision tree was aimed at responding to the state of the system as a function of its ability to fill orders in a timely manner. If the current Order WIP was greater than its moving average then the decision tree moved to the next level in the tree. If this was not the case then it held the current selection rule in place.

At the next level down in the decision tree the local buffer was evaluated for its level as well as its trajectory. The current local buffer level was compared to its moving average up to that point in the production run, and the direction of the trajectory of the buffer level was evaluated. If the current buffer level was above a certain upper threshold as a multiple of the moving average then the decision was made to change selection rules. If the buffer level was between another, lower, threshold and the upper threshold, then the trajectory was queried before the decision was made. If the trajectory was positive then the decision to switch selection rules was made. If this was not the case then it held the current selection rule in place.

5.2.1.4 The Harmony Search metaphor

In line with the HS algorithm structure, the Order Arrival Period and Order WIP decision levels were viewed as equivalent to the Harmony Memory Consideration Rate (HMCR), as indicated in Figure 5.1. In the traditional HS algorithm, the HMCR governed the rate at which new harmonies were generated from the harmony memory [78]. The HMCR represented the tempo at which the overall tune was updated. Similarly, the Order Arrival Period and Order WIP decision points were governed by the tempo at which the system was running at any given time.

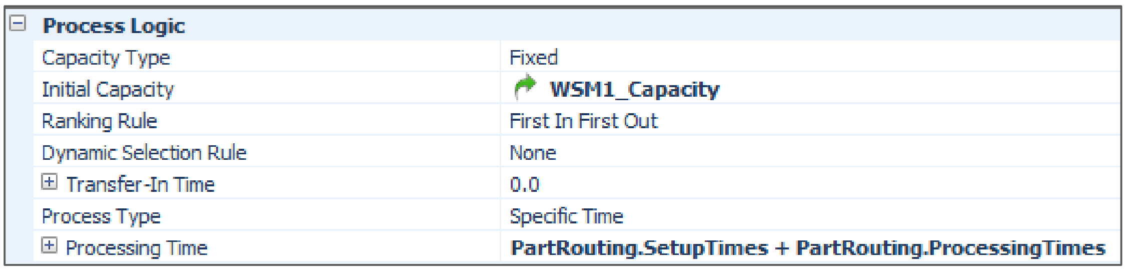
The rate at which the buffer level variables were queried were equivalent to the Pitch Adjustment Rate (PAR) of the HS algorithm, as indicated in Figure 5.1. In the traditional HS algorithm, the PAR determined the rate at which pitches within the selected harmony, based on the HMCR, were adjusted [78]. Similarly, the local buffer variable decision points were reached subsequent to the

Order Arrival Period and Order WIP decision points in the DDSRS decision tree. Although the traditional HS algorithm was designed for static problems, the metaphor on which it was based has a dynamic foundation which lends itself well to the problem at hand.

5.2.1.5 Implementation

To set a benchmark for the performance of the DDSRS, experiments were run in Simio with the four traditional selection rules implemented separately. Each experiment consisted of 50 replications, each of 1000 hrs run time. Testing of the DDSRS was carried out in simulation subsequent to testing the traditional selection rules. All testing was carried out on the dynamic simulation model which was created in Simio, and represented a hypothetical wristwatch production system case study, as described in Chapter 3.

For buffer queue handling, ranking rules and dynamic selection rules in Simio were implemented as part of the process logic of each object instance that represented a processing workstation. By default, no selection rule was active and the queue was simply ranked according to the order of part arrivals, i.e. FIFO, as shown in Figure 5.2.



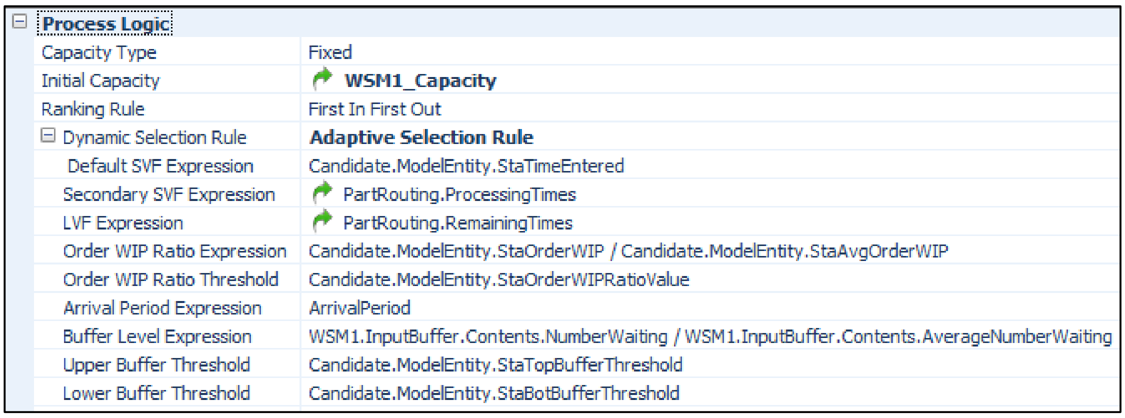
Process Logic	
Capacity Type	Fixed
Initial Capacity	WSM1_Capacity
Ranking Rule	First In First Out
Dynamic Selection Rule	None
Transfer-In Time	0.0
Process Type	Specific Time
Processing Time	PartRouting.SetupTimes + PartRouting.ProcessingTimes

Figure 5.2. Default process logic for Workstation M1 of wristwatch production system [68]**.**

Bases for common traditional selection rules were built into the process logic. These included smallest value first, largest value first, and three campaign-based rules. For each of these, the value expression for the rule to use in the selection of the next entity to process was nominated. Any dynamic selection rule that was implemented overrode the ranking rule. The standard and extension Application Program Interfaces (APIs) incorporated in Simio was used to implement the DDSRS.

Extension API interfaces exist for a number of components of Simio, including interfaces for the definition and implementation of selection rules. The Simio APIs were written in the C# programming language. The DDSRS was implemented by programming the decision tree in the form of a series of cascading if-then statements, which was followed by a switch statement that actioned the outcome of the decision tree. The code can be found in Appendix F.

The selection rules that were implemented as part of the DDSRS, as discussed in Section 5.1, were written as maximum or minimum value searching for-loops that searched through the candidate parts for the part that possessed the highest or lowest value of the expression as nominated in the simulation model. Default expressions for these for-loops were written into the extension API code for all input variables to the DDSRS. Figure 5.3 shows the input variables for the DDSRS as implemented in Simio.



Process Logic	
Capacity Type	Fixed
Initial Capacity	WSM1_Capacity
Ranking Rule	First In First Out
Dynamic Selection Rule	Adaptive Selection Rule
Default SVF Expression	Candidate.ModelEntity.StaTimeEntered
Secondary SVF Expression	PartRouting.ProcessingTimes
LVF Expression	PartRouting.RemainingTimes
Order WIP Ratio Expression	Candidate.ModelEntity.StaOrderWIP / Candidate.ModelEntity.StaAvgOrderWIP
Order WIP Ratio Threshold	Candidate.ModelEntity.StaOrderWIPRatioValue
Arrival Period Expression	ArrivalPeriod
Buffer Level Expression	WSM1.InputBuffer.Contents.NumberWaiting / WSM1.InputBuffer.Contents.AverageNumberWaiting
Upper Buffer Threshold	Candidate.ModelEntity.StaTopBufferThreshold
Lower Buffer Threshold	Candidate.ModelEntity.StaBotBufferThreshold

Figure 5.3. Dynamic selection rule input variables for the DDSRS for Workstation M1 of the wristwatch case study.

The following applies to Figure 5.3:

a) *Default SVF Expression* is the expression used for the EDD selection rule
b) *Secondary SVF Expression* is the expression used for the SPT selection rule
c) *LVF Expression* is the expression used for the MPTR and LPTR selection rules
d) *Order WIP Ratio Expression* takes the order WIP ratio variable as argument
e) *Order WIP Ratio Threshold* takes the order WIP ratio threshold property as argument
f) *Arrival Period Expression* takes the arrival period property as argument
g) *Buffer Level Expression* takes the local buffer level ratio variable as argument
h) *Upper Buffer Threshold* takes the upper threshold property for the local buffer level ratio as argument
i) *Lower Buffer Threshold* takes the lower threshold property for the local buffer level ratio as argument

Expressions d) through i) were used as variables and parameters by the decision tree of the DDSRS. Experiments were designed and carried out in Simio to determine the optimal ranges of values for these variables. The *k*-factorial experiment design method was used to investigate the effects of the values of these variables on the performance of the production system. The primary performance indicator was the average order lead time. The average number of order waiting to be filled was also monitored. Table 5.1 shows the values used in the *k*-factorial experimentation.

Table 5.1. *K*-Factorial experiment decision variable value ranges.

Order WIP Ratio	Units per Arrival Period	Buffer Level Ratio Lower Threshold	Buffer Level Ratio Upper Threshold
0.5	1	0.5	1.0
1.0	5	1.0	1.75
1.5	10	1.25	2.5

The *k*-factorial method produced $3^4 = 81$ scenarios. However, some scenarios were not feasible. These were where the Lower Buffer Level Ratio Threshold was equal to or greater than the Upper Buffer Level Ratio Threshold. These made up 18 scenarios, which left 63 scenarios to investigate. All scenarios were run for 50 replications, concurrently with each other. These ranges were determined through trial-and-error by adjusting each variable individually.

From the results of the *k*-factorial experiment, the ranges of the decision variables could be refined for a subsequent optimisation experiment. The optimisation experiment was run using OptQuest, a third-party Simio add-in for optimisation purposes developed by OptTek Systems, Inc. [79]. The optimisation experiment was set up with parameter settings as shown in Table 5.2. The OptQuest add-in automatically generated scenarios with different values for the control variables. The control variables used here were Order WIP ratio, Units per Arrival Period, Upper Buffer Level Ratio Threshold, and Lower Buffer Level Ratio Threshold.

Table 5.2. OptQuest parameter settings for final stage in DDSRS development.

Parameter	Setting
Minimum Replications	20
Maximum Replications	50
Maximum Scenarios	100
Confidence Level	95 %
Relative Error	0.1
Objective	Single Objective

With reference to Table 5.2, the Confidence Level parameter represented the level of accuracy to be used in the statistical comparison of one objective value to another. The Relative Error parameter represented the relative error of the confidence level expressed as a percentage of the mean. The objective used in the optimisation experiment was to minimise average order lead time. These parameters were used in the selection of the optimal scenario. The reason for running 50 replications of each scenario in all experiments was to ensure that statistically sound results could be achieved. Running a single random scenario for an increasing number of replications showed that this was a sufficient number of replications to generate statistically reliable results.

5.2.1.6 Production system case study

The DDSRS was developed in application to a hypothetical wristwatch case study production system. This production system was described in Chapter 3 along with the dynamic simulation model representing the system. The production system was designed to produce custom wristwatches as single unit orders. Based on the production requirements and probability distributions used in the setup of the dynamic simulation model in Chapter 3, Table 5.3 shows the expected theoretical maximum and minimum order lead times and number of orders waiting.

Table 5.3. Theoretical order lead time and number of orders waiting ranges (from Chapter 3).

	Minimum	**Maximum**
Order Lead Time (hrs)	2.20	4.33
Number of Orders Waiting	16.50	32.48

The values for the number of orders waiting were calculated based on Little’s Law [80] and the order arrival rate. Little’s Law can be stated in expression for as shown in Equation 5.1:

$$L = \lambda W \tag{5.1}$$

Where L is the average number of orders waiting, λ is the average order arrival rate, and W is the average order lead time. The average order arrival rate used was 7.5 orders/hr. The minimum values represent the time required for a unit to be produced assuming all workstations are already set up correctly, and processing times are two standard deviations below their mean values. The maximum values represent set up required at all workstations and processing times are two standard deviations above their mean values. All the values in Table 5.3 represent ideal steady state operation of the system.

All experiments were run in Simio (version 7.128.12863) installed on a PC with 8 GB of RAM and an Intel Core i7-4700MQ CPU running at 2.4 GHz. The *k*-factorial experiment took approximately 12 hrs to complete all 63 scenarios of 50 replications each. The optimisation experiment took approximately 20 hrs to complete all 100 scenarios of 50 replications each.

5.2.1.7 Data analysis

Analysis of the *k*-factorial experiment results involved identifying the scenarios that produced the lowest average order lead times to determine the ranges of the decision variables to use in the optimisation experiment. The results from the optimisation experiment were used to determine the final output in the development of the DDSRS. This produced a set of DDSRS decision variable values for use in the production system scheduling to minimise average order lead times.

To evaluate the final DDSRS results, the statistical distributions of the average order lead times and average number of orders waiting to be filled, generated by the individual selection rules and DDSRS were compared using a one-way ANOVA. Post-hoc pairwise comparison was carried out using the Tukey test, to determine which methods were statistically significantly different from one another.

The relationships between the average order lead times and average number of orders waiting were also analysed as a final rationality check for all selection rules and the DDSRS, to ensure the system behaviour followed Little's Law [80], within statistical reason. Statistical analysis was carried out using RStudio (version 0.99.489) [81].

5.2.2 Results

This section presents results achieved from the individual implementation of traditional selection rules as well as the DDSRS on the custom wristwatch production system case study simulation model. These results follow the development methodology as described in Section 5.2.1.

Table 5.4 shows a summary of the results for the average order lead time distribution of the individual selection rule implementations. Table 5.5 shows a summary of the results for the average number of orders waiting distribution for the individual selection rule implementations. When looking at the medians of Table 5.4 and Table 5.5 it can be seen that the EDD selection rule seems to outperform all other rules.

Table 5.4. Individual selection rule implementation average order lead time distribution summaries.

	Average Order Lead Time (hr)			
	EDD	SPT	MPTR	LPTR
Minimum	3.48	4.62	4.29	3.22
First Quartile	4.47	5.62	6.41	5.17
Median	6.20	7.12	8.63	6.95
Third Quartile	7.55	10.29	11.39	10.41
Maximum	13.33	21.44	20.29	21.63

Table 5.5. Individual selection rule implementation average number of orders waiting distribution summaries.

	Average Number of Orders Waiting			
	EDD	**SPT**	**MPTR**	**LPTR**
Minimum	25.56	33.91	31.52	23.65
First Quartile	33.15	41.50	47.44	38.52
Median	46.83	53.25	64.73	52.11
Third Quartile	56.87	78.00	85.37	79.90
Maximum	101.15	166.70	153.50	166.76

The DDSRS decision variables from the four scenarios in the *k*-factorial experiment that produced the lowest median average order lead times are contained in Table 5.6, along with their median average order lead time results. From these results the optimisation experiment was prepared as part of the development of the DDSRS. It can be seen from Table 5.6 that the ranges for the DDSRS decision variables could be reduced by a third from the *k*-factorial experiment ranges as per Table 5.1.

Table 5.6. Top four *k*-factorial experiment scenarios decision variable values.

Order WIP Ratio	Units per Arrival Period	Lower Buffer Level Ratio Threshold	Upper Buffer Level Ratio Threshold	Average Order Lead Time Median (hr)
0.5	5	0.5	2.5	5.63
1	5	1	1.75	6.05
1.5	1	0.5	1.75	5.82
0.5	5	1	2.5	5.97

The ranges used in the optimisation experiment were selected as given in Table 5.7. The intervals were selected such that they provide a fine enough resolution to be able to determine the optimal solution.

Table 5.7. OptQuest experiment decision variable value ranges.

Order WIP Ratio	Units per Arrival Period	Lower Buffer Level Ratio Threshold	Upper Buffer Level Ratio Threshold
0.5 to 1 in intervals of 0.025	1 to 5 in intervals of 1	0.5 to 1 in intervals of 0.0125	1.5 to 2.5 in intervals of 0.025

The scenario produced by the optimisation experiment that achieved the lowest median average order lead time and number of orders waiting was based on DDSRS decision variable values as shown in Table 5.8. Appendix G contains the average order lead times and numbers of orders waiting for all scenarios produced by the optimisation experiment.

Table 5.8. Optimisation experiment decision variable value results.

Order WIP Ratio	Units per Arrival Period	Lower Buffer Level Ratio Threshold	Upper Buffer Level Ratio Threshold
0.5	5	0.5	2.5

The scenario based on the decision variables in Table 5.8 produced an average order lead time distribution as summarised in Table 5.9, presented in comparison with the results achieved by the EDD selection rule. This scenario represented the optimal DDSRS decision variable combination. The median average order lead time produced by the DDSRS was 9.2 % lower than that of the EDD rule.

Table 5.9. Earliest Due Date and DDSRS order lead time distribution summary.

	Average Order Lead Time (hr)	
	EDD	**DDSRS**
Minimum	3.48	3.57
First Quartile	4.47	4.84
Median	6.20	5.63
Third Quartile	7.55	7.29
Maximum	13.33	15.25

Table 5.10 shows the distribution summary of the average number of orders waiting for the optimal DDSRS scenario based on the decision variable values of Table 5.8, in comparison with that of the EDD selection rule implementation.

Table 5.10. Earliest Due Date and DDSRS average number of orders waiting distribution summary.

	Average Number of Orders Waiting	
	EDD	**DDSRS**
Minimum	25.56	26.23
First Quartile	33.15	36.02
Median	46.83	42.07
Third Quartile	56.87	54.16
Maximum	101.15	117.51

Average order lead times generated using the individual selection rules and DDSRS are shown graphically in Figure 5.4. Corresponding average number of orders waiting are shown in Figure 5.5. Figure 5.4, Table 5.4, Table 5.5, Table 5.9, and Table 5.10 show distribution summaries of these results. From Figure 5.4 and Figure 5.5 it can be seen that the best permutation found by the optimisation experiment procedure appears to produce an overall lower average lead time range than two of the four individual selection rules.

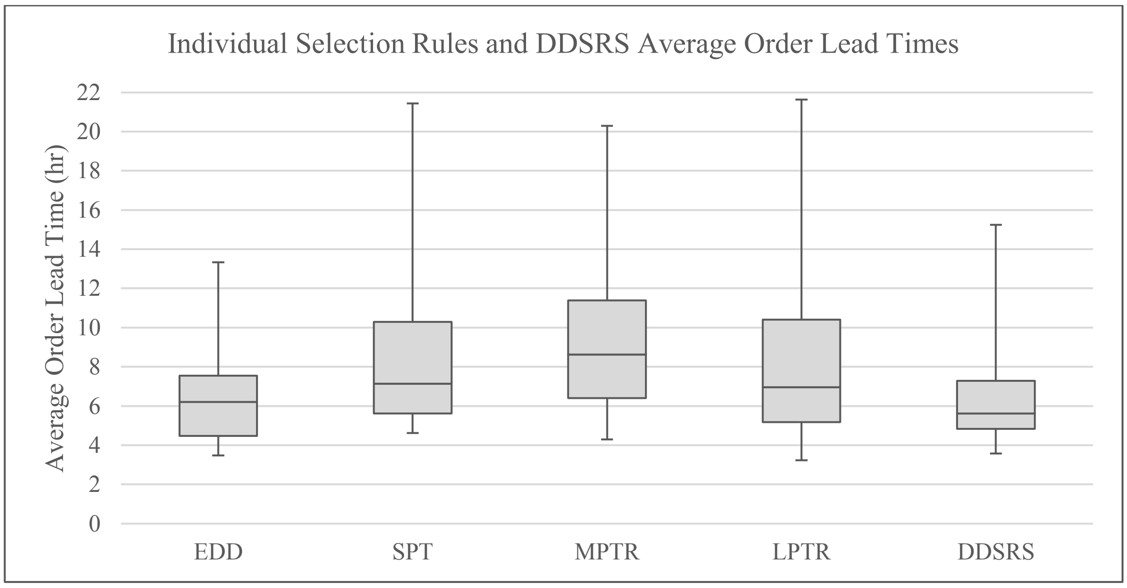


Figure 5.4. Box and whisker plot of individual selection rules and DDSRS average order lead times.

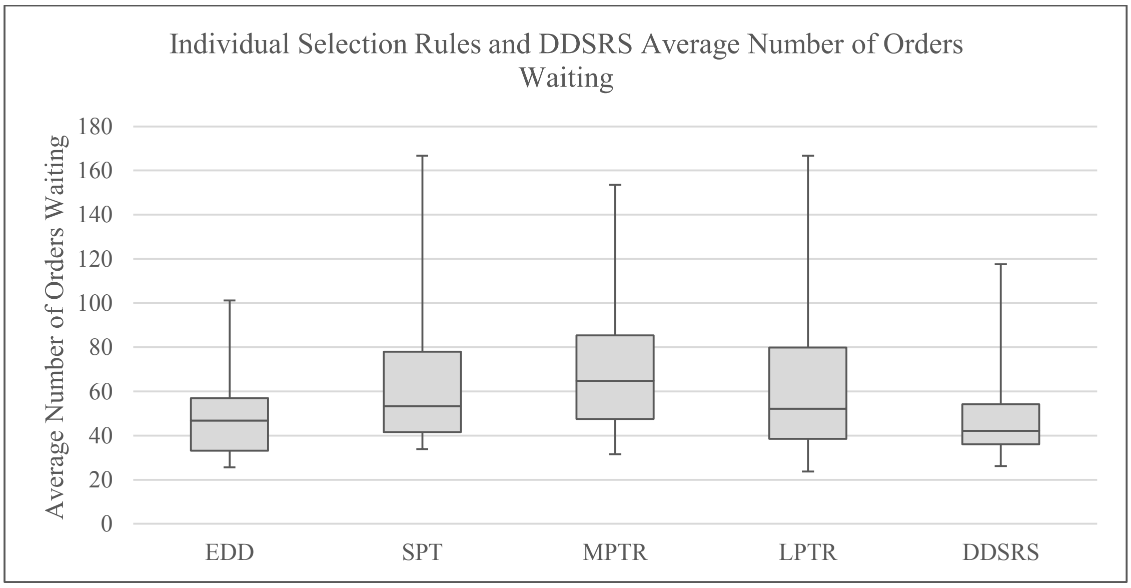


Figure 5.5. Box and whisker plot of individual selection rules and DDSRS average number of orders waiting.

The effective average order arrival rates were calculated (Table 5.11) from the median values of Table 5.4, Table 5.5, Table 5.9, and Table 5.10 using Little's Law as described in Section 5.2.1.6. All results were within 1 % of the actual average order arrival rate of 7.5 orders/hr. These results validated the correct functioning of the simulation models in all instances. The consistency of these results also meant that a one-way ANOVA and Tukey test were only required to be carried out on the average order lead time variable to effectively analyse the performance of the DDSRS.

Table 5.11. Effective average order arrival rates.

Selection Rule/ Strategy	Effective Average Order Arrival Rate (Orders/hr)
EDD	7.55
SPT	7.48
MPTR	7.50
LPTR	7.50
DDSRS	7.47

The one-way ANOVA, used to compare the average order lead times of all four selection rule implementations and the DDSRS with each other, produced results as shown in Table 5.12.

Table 5.12. Selection rule comparison one-way ANOVA results for average order lead times.

	Degrees of Freedom	Sum of Squares	Mean of Squares	F	p
Individual	4	371.4	92.85	7.51	1.01e-05
Residuals	245	3029.7	12.37		

The null hypothesis here was that all mean values of average lead times achieved by the different individual selection rules and the DDSRS were equal. The results of the one-way ANOVA, shown in Table 5.12, indicated that this hypothesis could be rejected (one-way ANOVA, $F > 1$, $p < 0.05$). The Tukey test used to determine whether statistically significant differences existed between pairs of methods produced results as shown in Table 5.13.

Table 5.13. Results of Tukey test pairwise comparison of selection rule implementations.

Pairs:	**Average Order Lead Times (hrs)**		**p adjusted**
(1)-(2)	**Median of (1)**	**Median of (2)**	
EDD-DDSRS	6.20	5.63	0.99998
LPTR-DDSRS	6.95	5.63	0.07497
MPTR-DDSRS	8.63	5.63	**0.00018**
SPT-DDSRS	7.13	5.63	**0.02750**
LPTR-EDD	6.95	6.20	0.06943
MPTR-EDD	8.63	6.20	**0.00016**
SPT-EDD	7.13	6.20	**0.02516**
MPTR-LPTR	8.63	6.95	0.38834
SPT-LPTR	7.13	6.95	0.99598
SPT-MPTR	7.13	8.63	0.99998

The Tukey test showed that there were statistically significant differences between the following pairs of scheduling rules/strategy:

- Most Processing Time Remaining and Distributed Dynamic Selection Rule Strategy
- Shortest Processing Time and Distributed Dynamic Selection Rule Strategy
- Most Processing Time Remaining and Earliest Due Date
- Shortest Processing Time and Earliest Due Date

From Figure 5.4 and Table 5.13 it can be established that EDD and DDSRS are most similar to each other in terms of their ranges and statistical distributions. Shortest Processing Time, LPTR and MPTR are most similar to each other, but not to EDD or DDSRS. The EDD and DDSRS results are overall lower than all other results. Statistical similarities exist between LPTR and EDD, and LPTR and DDSRS. This is due to the exceptionally wide distribution of the LPTR results. This is represented by the fact that the Tukey test was not able to detect a statistically significant difference between LPTR and any of the other methods, as seen in Table 5.13.

5.2.3 Discussion

According to Subramaniam et al [54] and Shaw et al. [82], it is impossible for a single selection rule to be optimally effective at every instance of its implementation over the entire length of time the production system is in operation. The DDSRS addressed this shortcoming by starting with a set of simple selection rules and enabling each instance of the strategy implementation to independently switch between rules. This way the system could adjust dynamically to its local and global state. The performance of the DDSRS applied to the hypothetical wristwatch production system case study has been discussed here with reference to relevant literature. The methodology of developing the DDSRS has also been discussed here as a general approach to solving the dynamic manufacturing scheduling problem.

The average order lead time and number of orders waiting achieved by the individual selection rule implementations showed the range of variation in these metrics possible, simply by implementing a different rule to a manufacturing system globally. The EDD selection rule produced lower average order lead time and number of orders waiting than all other selection rules tested. The Author believed that a contributing factor to these results is the fact that the EDD rule managed to keep the flow through the processing workstations relatively close to the flow order required by the assembly workstations. This was due to the fact that order due dates were directly linked to their arrival dates to the system. However, this did not imply that the most effective means of minimising the average order lead time and number of orders waiting was by matching the flow order required by the assembly workstations.

Statistical analysis carried out during this study showed that there was no statistically significant difference between the average order lead time produced by the DDSRS and the individual implementation of the EDD selection rule. However, the results of the Tukey test did show that the DDSRS performed just as well as the best performing globally implemented selection rule of those tested, that is the EDD selection rule. These results aligned well with previous work by Shaw et al. [82]. Shaw et al. predicted that their globally dynamic scheduling system would perform at least as well as the best among the candidate selection rules. Furthermore, the order of the improvement in average order lead time over traditional selection rules in this study were within the ranges found by other researchers when applying machine learning techniques to the dynamic scheduling problem for FMS and job shop manufacturing systems [52], [53], [55].

From the perspective of the DDSRS development methodology, identifying the most applicable traditional selection rules, as defined by Panwalkar and Iskander in [49], was the first step. In this study the most applicable traditional selection rules were aimed at the production time as measured by the average order lead time. If the main objective was uncertain or indistinguishable from others, testing of individual selection rules would need to be carried out in simulation to determine which subset performed well. This set could then be used as the base selection rules to switch between by the DDSRS. The goal in this step was to keep to tried-and-trusted selection rule implementations.

In the next step of the DDSRS development methodology, the effects of the decision variables were evaluated by way of a *k*-factorial experimental design method. This step was aimed at narrowing down the ranges of the decision variables leading on to the final step in the development methodology. The number of scenarios to investigate was calculated by the typical method of k^n, where k was the number of levels of each variable and n was the number of variables. However, some scenarios were excluded from the set due to the relationship between the two buffer level threshold variables. Although these variables were independent from each other in the sense that the value of one is not necessary for determining the value of the other, the value of one does exclude a certain range of the other. Specifically, the upper threshold logically needed to be greater than the lower threshold for the decision tree to function correctly.

In the final step of the DDSRS development methodology an OptQuest optimisation experiment was prepared based on the results of the *k*-factorial experiment discussed above. This experiment did not produce a scenario that showed improved average order lead time over those explored in the *k*-factorial experiment. Thus, it would be possible to forego the *k*-factorial experiment all together and go straight to the optimisation experiment to find the best combination of decision variable values. However, this is not advisable as it would necessitate increasing the maximum number of scenarios to be created by the optimisation experiment, which would lengthen the solution time of the optimisation experiment exponentially. This would be necessary to effectively explore the search space represented by the ranges of the decision variables. In this instance the ranges of the DDSRS decision variables could effectively be reduced by a third. This reduced the number of decision variable value combinations from 1 440 000 to 160 000.

The DDSRS development methodology has been designed to be followed for any flow shop-type production system. As part of the development of new production systems it is recommended to incorporate a simulation phase for cost-saving reasons [83]. The implementation of a methodology such as this one can assist the manufacturing enterprise in further improving their production capacity by reducing average order lead times which can increase throughput. It can also be used in a what-if study for order arrival process permutations, by preparing the threshold values for the different behaviour patterns expected from the order arrival process. The fundamental functionality of the DDSRS makes it adjustable to a range of system configurations and product platforms.

The DDSRS development methodology relies on the development of a simulation model of the system under investigation incorporating enough detail for modelling the scheduling of the system. In the development of a new production system, the development of a simulation model holds many advantages. These include time and cost saving by avoiding physical testing and prototyping [83]. Although proprietary simulation software was used in the development of the DDSRS, it is believed that the methodology is generic enough that it can be followed using any good simulation software if the practitioner was familiar enough with the software they were using.

5.3 Chapter summary

This chapter documented the development of a new distributed dynamic scheduling strategy, incorporating existing shop floor-level scheduling policies and system-level information for optimising the system performance. The strategy was applied to a hypothetical custom wristwatch production system as a case study to validate its performance and to answer the research question posed at the start of the study. Experimental results were presented, discussed and interpreted in relation to relevant literature and research objectives.

Chapter 6 Conclusion

This chapter restates the aim and objectives of the study as given in Chapter 1 and gives a brief summary of the research study with relation to the research question. Conclusions drawn from the work of each of the development chapters are discussed, and future research topics are identified.

6.1 Aim and objectives

The aim of this study was to research, develop, and test novel Computational Intelligence-based optimisation methods built on simulation models for Advanced Manufacturing Systems, at the planning and scheduling levels. The objectives identified at the outset of this study were:

1. Research the state of manufacturing strategies and manufacturing system types for compatibility with the research approach and the state of Computational Intelligence as a technology.

2. Develop manufacturing system models for implementation and testing of planning and scheduling optimisation techniques.

3. Research and develop a Computational Intelligence based optimisation technique for optimising production planning activities within the manufacturing system type identified in research.

4. Research and develop a Computational Intelligence based optimisation technique for optimising production scheduling activities within the manufacturing system type identified in research.

5. Deploy Computational Intelligence optimisation techniques in computer simulations of a case study production system to evaluate the performance of the techniques against traditional planning and scheduling methods, by analysing and interpreting computer simulation results, and draw conclusions.

6.2 Research summary

This research study investigated the application of novel approaches to the production planning and scheduling problems in order to determine whether Advanced Manufacturing Systems (AMSs) striving towards Mass Customisation Manufacturing (MCM) can be optimised more effectively using Computational Intelligence (CI) principles than by traditional methods. One common thread throughout the study was the simulation modelling methodology used in the investigation. Another common thread throughout the study was the concept of exploiting tried-and-trusted traditional methods using new principles, rather than fitting entirely new principles to the problem, as has been the trend in past research.

To address the question of whether AMSs could be optimised more effectively using new CI principles than traditional methods, from a production planning perspective, the previously unused Biogeography-Based Optimisation (BBO) principle was applied to the Aggregate Production Planning (APP) problem. Static simulation models were developed from the capacity planning activity through to Aggregate Production Planning. The BBO algorithm was applied as a search method to automate and accelerate the traditional cut-and-try method used for APP. To test the effectiveness of the BBO algorithm, results were analysed by comparison with results achieved by a Simulated Annealing (SA) algorithm as well as traditional planning strategies. The BBO algorithm was able to produce lower cost production plans than three traditional planning strategies. It also proved to outperform the SA algorithm by producing a lower cost plan in a smaller number of iterations with a smaller impact on workforce levels and closer tracking of production requirements.

To address the question of whether AMSs could be optimised more effectively than traditional methods using new CI principles, from a production scheduling perspective, a new Distributed Dynamic Selection Rule Strategy (DDSRS) for production scheduling was developed. This strategy was inspired by the metaphor on which the Harmony Search (HS) algorithm was based. This principle has not been used in this context before. The DDSRS was founded on simple traditional selection rules, and the development methodology was presented as a generic methodology for application to a range of production scheduling problems. Statistical analysis of the results achieved by the DDSRS compared to the traditional selection rules showed no statistically significant improvement over the best performing rule. However, some improvement was observable in terms of resistance to variation in order arrival rates and marginally lower average order lead times and average number of orders waiting to be filled.

In summary, the research question posed at the start of this study was approached from two different perspectives, one a static planning perspective, and the other a dynamic scheduling perspective. In doing so, two new approaches were found to be able to optimise an AMS more effectively than traditional methods. These were a BBO algorithm for APP and an HS inspired distributed dynamic scheduling strategy. This study also made a contribution toward the promotion of simulation modelling for production system optimisation.

6.3 Simulation modelling

Simulation modelling has become more than just a what-if scenario testing tool. It can be a valuable asset to a manufacturing enterprise if developed with the correct objectives. Different models of a manufacturing system can be developed for different purposes. This is advisable to avoid attempting to develop a single simulation model that is completely accurate, but too complex and computationally demanding to be useful. The simulation model developed for production scheduling was useful for investigating scheduling optimisation, but in hindsight could have been simpler in its construction and inclusion of components such as order handling and processing. This would have reduced run time of the experiments designed into the methodology, which would improve the profitability of using simulation modelling for this purpose.

6.4 Production planning

Aggregate Production Planning calculates estimates of production requirements based on demand forecasts and production costs. Due to the nature of the problem there will always be an element of uncertainty. By taking into account worst-case conditions of setup times required for every processing and assembly operation in such a production system the BBO algorithm was able to calculate a lower cost production plan than traditional strategies as well as a Simulated Annealing algorithm.

The goal in this part of the study was to develop a new algorithm founded on tried-and-trusted knowledge of APP as well as newly developed technology. This has produced a planning optimisation technique that achieved more cost effective plans than traditional optimisation techniques. The results achieved by the BBO algorithm were within ranges achieved by similar algorithms found in literature, which reinforced the validity of the approach. A conclusion that can be drawn from this is that it is not always necessary to reinvent the wheel, so to speak. Extension of existing technologies designed for high volume low variety production is a feasible option for optimising systems designed for MCM through single order unit production.

It was found that both the optimisation algorithms developed for the purpose of optimising the aggregate production plan possessed control parameters which greatly affected the performance of the algorithms. However, due to the nature of the problem space the task of adjusting these parameters was not a time consuming one. It is believed that the time saved by applying algorithms such as these would still outweigh the time spent adjusting the algorithm control parameters. These algorithms would also allow production planners to explore more scenarios in a shorter amount of time once the algorithm parameters were adjusted adequately.

6.5 Distributed dynamic production scheduling

Although no statistically significant improvement was found between the average order lead time produced by the DDSRS and that produced by the Earliest Due Date (EDD) selection rule, there was an observable improvement in the results achieved by the DDSRS in terms of the median of the average order lead time as well as the distribution between the first and third quartiles. This supports the conclusion that it can be more beneficial for each processing workstation in such a production system to individually switch its selection rule rather than implementing a global selection rule across the system.

The DDSRS allowed each workstation to dynamically and individually respond to its local situation as well as the system state, which improved the responsiveness of the system as a whole. Furthermore, it can be stated that the extension of tried-and-trusted methods, in this case traditional selection rules, was a worthwhile endeavour for improving scheduling of production systems processing single unit orders. Especially using principles that were analogous to the structure and operation of the fundamental scheduling system.

Although production systems are typically based on a finite set of system philosophies and management systems, every manufacturing system is subtly different in many respects. For this reason a generic methodology is proposed for the development of the DDSRS on a case-specific basis. It can be stated that such a methodology is much more useful to production scheduling practitioners because of the fact that every production system is unique. Despite this fact, the instruments for solving the production scheduling problem have converged to computer based software such as Simio simulation software. This convergence enhances the usefulness of a generic methodology because these instruments tend to possess the same capabilities and functionality in terms of experimentation and optimisation.

6.6 Future research

The fact that the BBO algorithm parameters require adjustment is its biggest drawback. This is the case for most algorithms such as this one, which depend on parameters to determine its exact operation under specific circumstances. Although it is never advisable for decision makers to use models as "black boxes", it does put more of a burden on the decision makers to learn how these models work, which is usually a deterrent for adopting new technology. Effort into developing a simpler more intuitive algorithm would do well for promoting its adoption by production planning practitioners.

The dynamic simulation model and the extension to the simulation software provide a good foundation for further work in simulation-based optimisation research. This would be especially useful for continued research into distributed dynamic selection rule switching based on other CI paradigms. The addition of limited machine learning or memory capability, without straining the computational system too much may also improve results.

In general, it is believed that it is a worthwhile undertaking to spend further research effort on the extension of existing widely used and widely understood planning and scheduling methods. This aligns well with the evolution of production systems from high volume low variety to high volume high variety production systems for MCM. This strategy will also improve chances of wider acceptance of new approaches by industry decision makers and practitioners, if they know that the new approaches are based on those that they know and trust.

References

[1] F. S. Fogliatto, G. J. C. da Silveira, and D. Borenstein, “The mass customization decade: An updated review of the literature,” *Int. J. Prod. Econ.*, vol. 138, no. 1, pp. 14–25, Jul. 2012.

[2] B. J. Pine, *Mass customization: The frontier in business competition*. Boston, MA: Harvard Business School Press, 1993.

[3] R. B. Chase, N. J. Aquilano, and F. R. Jacobs, *Operations management for competitive advantage*, 11th Editi. McGraw-Hill, 2006.

[4] B. MacCarthy, P. G. Brabazon, and J. Bramham, “Fundamental modes of operation for mass customization,” *Int. J. Prod. Econ.*, vol. 85, no. 3, pp. 289–304, Sep. 2003.

[5] S. Rajagopalan and N. Xia, “Product variety, pricing and differentiation in a supply chain,” *Eur. J. Oper. Res.*, vol. 217, no. 1, pp. 84–93, Feb. 2012.

[6] H. ElMaraghy, G. Schuh, W. ElMaraghy, F. Piller, P. Schönsleben, M. Tseng, and A. Bernard, “Product variety management,” *CIRP Ann. - Manuf. Technol.*, vol. 62, no. 2, pp. 629–652, Jan. 2013.

[7] C. Fornell, “A national customer satisfaction barometer: The Swedish experience,” *J. Mark.*, vol. 56, no. January, pp. 6–21, 1992.

[8] H. A. ElMaraghy, “Flexible and reconfigurable manufacturing systems paradigms,” *Int. J. Flex. Manuf. Syst.*, vol. 17, no. 4, pp. 261–276, Oct. 2006.

[9] J. (Roger) Jiao, T. W. Simpson, and Z. Siddique, “Product family design and platform-based product development: a state-of-the-art review,” *J. Intell. Manuf.*, vol. 18, no. 1, pp. 5–29, Jul. 2007.

[10] J. (Roger) Jiao, L. Zhang, and S. Pokharel, “Process platform planning for variety coordination from design to production in mass customization manufacturing,” *IEEE Trans. Eng. Manag.*, vol. 54, no. 1, pp. 112–129, Feb. 2007.

[11] A. I. Shabaka and H. A. ElMaraghy, “Generation of machine configurations based on product features,” *Int. J. Comput. Integr. Manuf.*, vol. 20, no. 4, pp. 355–369, Jun. 2007.

[12] J. A. Collins, “Numerical control and flexible manufacturing systems,” *Fact. Autom. Invit. Pap. INFORTECH State Art Report, Ser. 8, No. 6*, 1980.

[13] A. Osareh, B. Shadgar, and R. Markham, “A computational-intelligence-based approach for detection of exudates in diabetic retinopathy images.,” *IEEE Trans. Inf. Technol. Biomed.*, vol. 13, no. 4, pp. 535–545, 2009.

[14] J. J. Valdes, A. Pou, and A. P. Julio J Valdes, “Climatic variation of the structure of maximum daily temperatures in Spain: A combined statistical and computational intelligence approach (2009),” in *International Joint Conference on Neural Networks*, 2009, pp. p. 3172–3179.

[15] P. P. Wang, *Computational Intelligence in Economics and Finance*, 1st ed. Springer, 2010.

[16] A. K. Kordon, “Issues in applying computational intelligence,” in *Applying*

Computational Intelligence, vol. 4, no. 3, Springer, 2010, pp. 257–276.

[17] A. P. Engelbrecht, *Computational intelligence: an introduction*, Second Edi. Chichester, England: John Wiley & Sons, Inc., 2007.

[18] B. Xing and W.-J. Gao, *Innovative computational intelligence : A rough guide to 134 clever algorithms*. London: Springer, 2013.

[19] A. K. Kordon, *Applying computational intelligence: how to create value*, vol. 12, no. 1. Berlin, Heidelberg: Springer-Verlag, 2009.

[20] C. Renzi, F. Leali, M. Cavazzuti, and A. O. Andrisano, “A review on artificial intelligence applications to the optimal design of dedicated and reconfigurable manufacturing systems,” *Int. J. Adv. Manuf. Technol.*, vol. 72, no. 1–4, pp. 403–418, Feb. 2014.

[21] J. D. Ferreira, “Bio-Inspired Self-Organisation in Evolvable Production Systems,” The Royal Institute of Technology, 2013.

[22] G. Lanza and S. Peters, “Integrated capacity planning over highly volatile horizons,” *CIRP Ann. - Manuf. Technol.*, vol. 61, no. 1, pp. 395–398, Jan. 2012.

[23] O. Ceryan and Y. Koren, “Manufacturing capacity planning strategies,” *CIRP Ann. - Manuf. Technol.*, vol. 58, no. 1, pp. 403–406, Jan. 2009.

[24] K. E. Stecke, “Design, planning, scheduling, and control problems of flexible manufacturing systems,” *Ann. Oper. Res.*, vol. 3, no. 1, pp. 3–12, Jan. 1985.

[25] A. Baykasoglu, “MOAPPS 1.0: Aggregate production planning using the multiple-objective tabu search,” *Int. J. Prod. Res.*, vol. 39, no. 16, pp. 3685–3702, 2001.

[26] V. Jääskeläinen, “A goal programming model of aggregate production planning,” *Swedish J. Econ.*, vol. 71, no. 1, pp. 14–29, Mar. 1969.

[27] D. A. Goodman, “A goal programming approach to aggregate planning of production and work force,” *Manage. Sci.*, pp. 1569–1575, 1974.

[28] J. Mula, R. Poler, J. P. García-Sabater, and F. C. Lario, “Models for production planning under uncertainty: A review,” *Int. J. Prod. Econ.*, vol. 103, no. 1, pp. 271–285, Sep. 2006.

[29] R.-C. Wang and H.-H. Fang, “Aggregate production planning with multiple objectives in a fuzzy environment,” *Eur. J. Oper. Res.*, vol. 133, pp. 521–536, 2001.

[30] A. Baykasoglu and T. Gocken, “Multi-objective aggregate production planning with fuzzy parameters,” *Adv. Eng. Softw.*, vol. 41, no. 9, pp. 1124–1131, 2010.

[31] S. C. H. Leung and Y. Wu, “A robust optimization model for stochastic aggregate production planning,” vol. 15, no. 5, pp. 502–514, 2004.

[32] M. Mezghani, T. Loukil, and B. Aouni, “Aggregate planning through the imprecise goal programming model: Integration of the manager’s preferences,” *Int. Trans. Oper. Res.*, vol. 19, no. 4, pp. 581–597, 2012.

[33] D. Simon, “Biogeography-based optimization,” *IEEE Trans. Evol. Comput.*, vol. 12, no. 6, pp. 702–713, 2008.

[34] A. M. Bonvik, C. E. Couch, and S. B. Gershwin, “A comparison of production-line control

mechanisms," *Int. J. Prod. Res.*, vol. 35, no. 3, pp. 789–804, 1997.

[35] C. Chatzopoulos, A. Tsigkas, and R. Freund, "Making approaches between flow manufacturing and mass customization industries," in *Proceedings of the 15th International Scientific Conference on Industrial Systems*, 2011.

[36] B. Stump and F. Badurdeen, "Integrating lean and other strategies for mass customization manufacturing: A case study," *J. Intell. Manuf.*, vol. 23, no. 1, pp. 109–124, Jul. 2012.

[37] R. L. Shell and E. L. Hall, Eds., *Handbook of industrial automation*. New York: Marcel Dekker, Inc., 2000.

[38] H. Sharifi and Z. Zhang, "Agile manufacturing in practice. Application of a methodology," *Int. J. Oper. Prod. Manag.*, vol. 21, no. 5/6, pp. 772–794, 2001.

[39] Y. Y. Yusuf, M. Sarhadi, and A. Gunasekaran, "Agile manufacturing: The drivers, concepts and attributes," *Int. J. Prod. Econ.*, vol. 62, pp. 33–43, 1999.

[40] S. L. Goldman and R. N. Nagel, "Management, technology and agility: the emergence of a new era in manufacturing," *Int. J. Technol. Manag.*, vol. 8, pp. 18–38, 1993.

[41] E. M. Goldratt, *What is this thing called theory of constraints and how should it be implemented?* North River Press, 1990.

[42] P. Priore, D. De La Fuente, A. Gomez, and J. Puente, "A review of machine learning in dynamic scheduling of flexible manufacturing systems," *Artif. Intell. Eng. Des. Anal. Manuf.*, vol. 15, no. July 2001, pp. 251–263, 2001.

[43] A. Sharifnia, M. Caramanis, and S. B. Gershwin, "Dynamic setup scheduling and flow control in manufacturing systems," *Discret. Event Dyn. Syst. Theory Appl.*, vol. 1, no. 2, pp. 149–175, 1991.

[44] W.-M. Lan and T. L. Olsen, "Multiproduct systems with both setup times and costs: Fluid bounds and schedules," *Oper. Res.*, vol. 54, no. 3, pp. 505–522, 2006.

[45] K. E. Stecke, "Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems," *Manage. Sci.*, vol. 29, no. 3, pp. 273–288, 1983.

[46] C. Chiu and Y. Yih, "A learning-based methodology for dynamic scheduling in distributed manufacturing systems," *Int. J. Prod. Res.*, vol. 33, no. 11, pp. 3217–3232, 1995.

[47] S. Bergmann, S. Stelzer, and S. Strassburger, "On the use of artificial neural networks in simulation-based manufacturing control," *J. Simul.*, vol. 8, no. 1, pp. 76–90, Apr. 2013.

[48] A. Negahban and J. S. Smith, "Simulation for manufacturing system design and operation: Literature review and analysis," *J. Manuf. Syst.*, vol. 33, no. 2, pp. 241–261, Apr. 2014.

[49] S. S. Panwalkar and W. Iskander, "A Survey of scheduling rules," *Oper. Res.*, vol. 25, no. 1, pp. 45–61, 1977.

[50] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov, "A survey of scheduling problems with setup times or costs," *Eur. J. Oper. Res.*, vol. 187, no. 3, pp. 985–1032, Jun. 2008.

[51] S. C. Park, N. Raman, and M. J. Shaw, "Adaptive scheduling in dynamic flexible

manufacturing systems : A dynamic rule selection approach," *IEEE Trans. Robot.*, vol. 13, no. 4, pp. 486–502, 1997.

[52] K. R. B. Reddy, X. Na, and V. Subramaniam, "Dynamic scheduling of flexible manufacturing systems", *ser. Innovation in Manuf, Syst. and Techno., Singapore-MITAlliance,* 2004.

[53] M. Kapanoglu and M. Alikalfa, "Learning IF–THEN priority rules for dynamic job shops using genetic algorithms," *Robot. Comput. Integr. Manuf.*, vol. 27, no. 1, pp. 47–55, 2011.

[54] V. Subramaniam, T. Ramesh, G. K. Lee, Y. S. Wong, and G. S. Hong, "Job shop scheduling with dynamic fuzzy selection of dispatching rules," *Int. J. Adv. Manuf. Technol.*, vol. 16, no. 10, pp. 759–764, 2000.

[55] P. Priore, D. de la Fuente, J. Puente, and J. Parreno, "A comparison of machine-learning algorithms for dynamic scheduling of flexible manufacturing systems," *Eng. Appl. Artif. Intell.*, vol. 19, no. 3, pp. 247–255, 2006.

[56] S. Nakasuka and T. Yoshida, "Dynamic scheduling system utilizing machine learning as a knowledge acquisition tool," *Int. J. Prod. Res.*, vol. 30, no. 2, pp. 411–431, 1992.

[57] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: Harmony search," *Simulation*, vol. 76, no. 2, pp. 60–68, 2001.

[58] L. Wang, Q. Pan, and M. F. Tasgetiren, "A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem," *Comput. Ind. Eng.*, vol. 61, no. 1, pp. 76–83, 2011.

[59] Y. Yuan, H. Xu, and J. Yang, "A hybrid harmony search algorithm for the flexible job shop scheduling problem," *Appl. Soft Comput.*, vol. 13, no. 7, pp. 3259–3272, 2013.

[60] D. Weyland, "A rigorous analysis of the harmony search algorithm: How the research community can be misled by a 'novel' methodology," *Int. J. Appl. Metaheuristic Comput.*, vol. 1, no. 2, pp. 50–60, 2010.

[61] Z. W. Geem, "Research Commentary: Survival of the fittest algorithm or the novelest algorithm?," *Int. J. Appl. Metaheuristic Comput.*, vol. 1, no. 4, 2010.

[62] F. Penati, "P77: Filipo Penati blogger & photographer, always in love with new technologies," 2012. [Online]. Available: http://filippopenati.com/2012/05/18/esercizi-di-design-p77-watches-on-fewsome-com/. [Accessed: 27-Aug-2015].

[63] Watchshop.com, "Men's Accurist chronograph watch," 2015. [Online]. Available: http://www.watchshop.com/mens-accurist-chronograph-watch-mb936bb-p99940811.html. [Accessed: 25-Nov-2015].

[64] Simio LLC, "Flexible manufacturing example," 2015. [Online]. Available: http://www.simio.com/resources/videos/Flexible-Manufacturing-Cell-Video-Series/index.php. [Accessed: 26-Nov-2015].

[65] A. Singholi, D. Chhabra, and M. Ali, "Towards improving the performance of flexible manufacturing system: A case study," *J. Ind. Eng. Manag.*, vol. 3, no. 1, pp. 87–115, 2010.

[66] A. M. Law, *Simulation modeling & analysis*, 4th ed. New York, NY: McGraw-Hill, 2007.

[67] J. Banks, J. Carson, B. L. Nelson, and D. Nicol, *Discrete-event system simulation*. Prentice

Hall, 2004.

[68] Simio LLC, “Simio simulation software.” Simio LLC, Sewickley, PA, 2015.

[69] D. W. Kelton, J. S. Smith, and D. T. Sturrock, *Simio and simulation: Modeling, analysis, application*, Third. Simio LLC, 2014.

[70] C. A. Chung, *Simulation modeling handbook: A practical approach*. Boca Raton, FL: CRC Press LLC, 2004.

[71] Picario, “Picario - Ecommerce visualization solutions,” 2015. [Online]. Available: http://www.picario.com. [Accessed: 27-Aug-2015].

[72] Simio LLC, *Simio reference guide*. Simio LLC, Sewickley, PA, 2013.

[73] R. G. Sargent, “Verification and validation of simulation models,” *J. Simul.*, vol. 7, no. 1, pp. 12–24, Dec. 2012.

[74] J. R. Koza, *Genetic programming: On the programming of computers by means of natural selection*. Bradford, 1992.

[75] M. Souier, Z. Sari, and A. Hassam, “Real-time rescheduling metaheuristic algorithms applied to FMS with routing flexibility,” *Int. J. Adv. Manuf. Technol.*, vol. 64, no. 1–4, pp. 145–164, 2013.

[76] F. Musharavati and A. M. S. Hamouda, “Simulated annealing with auxiliary knowledge for process planning optimization in reconfigurable manufacturing,” *Robot. Comput. Integr. Manuf.*, vol. 28, no. 2, pp. 113–131, Sep. 2011.

[77] B. K. Choi and N. K. You, “Dispatching rules for dynamic scheduling of one-of-a-kind production.,” *Int. J. Comput. Integr. Manuf.*, vol. 19, no. 4, pp. 383–392, 2006.

[78] J. Brownlee, *Clever Algorithms*. LuLu, 2011.

[79] OpTek Systems, “OptQuest | OptTek Systems, Inc. - World Leader in Simulation Optimization,” 2011. [Online]. Available: http://www.opttek.com/OptQuest. [Accessed: 11-Nov-2015].

[80] J. D. C. Little, “A Proof for the Queuing Formula: L= λW,” *Oper. Res.*, vol. 9, no. 3, pp. 383–387, 1961.

[81] RStudio Team, “RStudio: Integrated development environment for R.” RStudio, Inc., Boston, MA, 2015.

[82] M. J. Shaw, S. Park, and N. Raman, “Intelligent scheduling with machine learning capabilities: The induction of scheduling knowledge,” *IIE Trans.*, vol. 24, no. 2, pp. 156–168, 1992.

[83] A. M. Law, “Simulation of manufacturing systems,” in *Modeling Simulation and Analysis*, 4th ed., vol. 1, McGraw-Hill, 2014, pp. 14–1 – 14–47.

Appendix A   Wristwatch Product Range Production Data

Table A.1. Wristwatch product range Generic Bill of Materials.

Item No.	Part Family ID	Part Family Name	Parent Item	No. Required	Material
1	CHM-BY-BL	Bezel	CHM-BY	1	Stainless Steel
2	CHM-BY-GS	Glass	CHM-BY	1	Mineral crystal
3	CHM-BY-GG	Glass gasket	CHM-BY	1	Nylon
4	CHM-CE-PR	Pusher	CHM-CE	2/0	Stainless Steel
5	CHM-CE-PS	Pusher Screw	CHM-CE	2/0	Stainless Steel
6	CHM-CE-PG	Pusher spring	CHM-CE	2/0	Spring Steel
7	CHM-CE-PT	Pusher case tube	CHM-CE	2/0	Stainless Steel
8	CHM-CE-PL	Pusher seal	CHM-CE	2/0	Rubber
9	CHM-CE-CT	Crown case tube	CHM-CE	1	Stainless Steel
10	CHM-CE-CE	Case	CHM-CE	1	Stainless Steel
11	CHM-CE	Case sub-assembly	CHM-BY	1	Multiple
12	CHM-TS-DL	Dial	CHM-TS	1	Stainless Steel
13	CHM-TS-HP	Dial holding pin	CHM-TS	2	Stainless Steel
14	CHM-TS-HH	Hour hand	CHM-TS	1	Stainless Steel
15	CHM-TS-MH	Minute hand	CHM-TS	1	Stainless Steel
16	CHM-TS-SH	Second hand	CHM-TS	1	Stainless Steel
17	CHM-TS-SM	Small hand	CHM-TS	3/0	Stainless Steel
18	CHM-TS-MT	Movement	CHM-TS	1	Multiple
19	CHM-TS	Time sub-assembly	CHM-BY	1	Multiple
20	CHM-CS-CS	Crown seal	CHM-CS	1	Rubber
21	CHM-CS-SM	Stem	CHM-CS	1	Stainless Steel
22	CHM-CS-CN	Crown	CHM-CS	1	Stainless Steel
23	CHM-CS	Crown sub-assembly	CHM-BY	1	Multiple
24	CHM-BY-SR	Spacer	CHM-BY	1	ABS Plastic
25	CHM-BY-BG	Back case gasket	CHM-BY	1	Rubber
26	CHM-BY-BC	Back case	CHM-BY	1	Stainless Steel
27	CHM-BY	Body sub-assembly	CHM	1	Multiple
28	CHM-BT	Bracelet	CHM	1	Polyurethane
29	CHM-BP	Bracelet pin	CHM	2	Stainless Steel
30	CHM	Wrist watch	NONE	1	Multiple

Table A.2. Wristwatch product range processing Bill of Operations.

Seq. No.*	Processing Operation	Work station	Setup Time (min/part)	Runtime (min/part)	Fixture/ Setup
35	Wrist Watch Assembly & Inspection (A5)	WS-A5	1.0	6.0	S-A5
30	Watch Body Assembly & Inspection (A4)	WS-A4	2.0	6.0	S-A4
25	Crown Sub-assembly (A3)	WS-A3	1.0	3.0	S-A3
20	Crown Fabrication Operation 3 (P13)	WS-M6	2.0	3.0	S-M6/C
10	Crown Fabrication Operation 2 (P12)	WS-M1	6.0	4.0	S-M1/C
5	Crown Fabrication Operation 1 (P11)	WS-M5	7.0	10.0	S-M5/C
25	Time Sub-assembly (A2)	WS-A2	1.0	7.0	S-A2
15	Dial Fabrication Operation 3 (P10)	WS-M7	2.0	1.0	S-M7
10	Dial Fabrication Operation 2 (P9)	WS-M4	5.0	7.5	S-M4
5	Dial Fabrication Operation 1 (P8)	WS-M3	2.5	1.0	S-M3
25	Case Sub-assembly (A1)	WS-A1	1.0	6.5	S-A1
20	Case Fabrication Operation 3 (P7)	WS-M6	2.0	3.0	S-M6/A
15	Case Fabrication Operation 2 (P6)	WS-M2	2.5	3.0	S-M2/C
10	Case Fabrication Operation 1 (P5)	WS-M1	21.0	27.0	S-M1/B
15	Back Case Fabrication Operation 3 (P17)	WS-M2	2.0	2.0	S-M2/B
10	Back Case Fabrication Operation 2 (P16)	WS-M1	4.0	5.0	S-M1/A
5	Back Case Fabrication Operation 1 (P15)	WS-M5	10.0	10.0	S-M5/B
20	Bezel Fabrication Operation 4 (P4)	WS-M6	2.0	3.0	S-M6/B
15	Bezel Fabrication Operation 3 (P3)	WS-M2	2.5	2.0	S-M2/A
10	Bezel Fabrication Operation 2 (P2)	WS-M1	4.0	2.0	S-M1/D
5	Bezel Fabrication Operation 1 (P1)	WS-M5	11.0	9.5	S-M5/A
25	Spacer Fabrication Operation 1 (P14)	WS-M8	2.0	25.0	S-M8

* Same sequence numbers indicate processing able to be completed in parallel

Table A.3. Wristwatch product range assembly Bill of Operations.

Seq. No.*	Assembly Operation	Work station	Setup Time (min/ operation)	Runtime (min/ operation)	Fixture/ Setup
35	Wrist Watch Assembly & Inspection (A5)	WS-A5	1.0	6.0	S-A5
30	Watch Body Assembly & Inspection (A4)	WS-A4	2.0	6.0	S-A4
25	Crown Sub-assembly (A3)	WS-A3	1.0	3.0	S-A3
25	Time Sub-assembly (A2)	WS-A2	1.0	7.0	S-A2
25	Case Sub-assembly (A1)	WS-A1	1.0	6.5	S-A1

* Same sequence numbers indicate processing able to be completed in parallel.

Table A.4. Wristwatch product range detailed processing requirements.

Case		*Proc. Times (min)*	*Setup Time /machine*	*Proc. Time /machine*
Raw Material:	60 x 60 x 20 mm Stainless Steel Billet	58.5	25.5	33.0
Processing Machine Type 1:	4-Axis CNC Milling Machine	48.0	21.0	27.0
Setup		5.0		
Process 1:	Rough shape milling - Top	3.0		
Tool change		1.0		
Process 2:	Finish milling - Top	5.0		
Refixture & tool change		2.0		
Process 3:	Rough shape milling - Bottom	3.0		
Tool change		1.0		
Process 4:	Finish milling - Bottom	5.0		
Refixture & tool change		2.0		
Process 5:	Drill pin holes	2.0		
Tool change		1.0		
Process 6:	Drill crown hole	1.0		
Tool change		1.0		
Process 7:	Face crown hole	1.0		
Tool change		1.0		
Process 8:	Thread crown hole	1.0		
Refixture & tool change		2.0		
Process 9:	Drill upper pusher hole	1.0		
Tool change		1.0		
Process 10:	Face upper pusher hole	1.0		
Tool change		1.0		
Process 11:	Thread upper pusher hole	1.0		
Tool change		1.0		
Process 12:	Drill lower pusher hole	1.0		
Tool change		1.0		
Process 13:	Face lower pusher hole	1.0		
Tool change		1.0		
Process 14:	Thread lower pusher hole	1.0		
Processing Machine Type 2:	Multi-purpose Grinding Machine	5.5	2.5	3.0
Setup		2.5		
Process 15:	Surface finishing	3.0		
Processing Machine Type 6:	Physical Vapour Deposition (PVD) Machine	5.0	2.0	3.0
Setup		2.0		
Process 16:	PVD case	3.0		
Dial		*Proc. Times (min)*	*Setup Time /machine*	*Proc. Time /machine*
Raw Material:	0.5 x 50 mm Stainless Steel Sheet Metal Roll	19.0	9.5	9.5
Processing Machine Type 3:	Piercing and Blanking Machine	3.5	2.5	1.0

Setup		2.0		
Process 1:	Pierce out dial holes	0.5		
Refixture		0.5		
Process 2:	Blank out Dial	0.5		
Processing Machine Type 4:	Pad Printing Machine	12.5	5.0	7.5
Setup		3.0		
Process 3:	Pad printing base colour	4.5		
Tool change		2.0		
Process 4:	Pad print accents/small dials	3.0		
Processing Machine Type 7:	Stencil Gluing Machine	3.0	2.0	1.0
Setup		2.0		
Process 5:	Glue hour/luminous markers	1.0		
Crown		*Proc. Times (min)*	*Setup Time /machine*	*Proc. Time /machine*
Raw Material:	Dia. 10 mm Stainless Steel Bar Stock	32.0	15.0	17.0
Processing Machine Type 5:	Feed-through CNC Lathe	17.0	7.0	10.0
Setup		3.0		
Process 1:	Face bar stock	0.5		
Process 2:	Rough turn stock down to max dia.	0.5		
Process 3:	Finish turn outside dia.	1.0		
Process 4:	Turn down step for case tube cavity	2.0		
Tool change		1.0		
Process 5:	Drill out centre for stem cavity	1.0		
Tool change		1.0		
Process 6:	Bore centre for case tube cavity	1.0		
Tool change		1.0		
Process 7:	Thread stem cavity	2.0		
Tool change		1.0		
Process 8:	Turn down groove(s) for crown seal	1.0		
Process 9:	Part off	1.0		
Processing Machine Type 1:	4-Axis CNC Milling Machine	10.0	6.0	4.0
Setup		5.0		
Process 11:	Shape end face and machine pattern into end face	2.0		
Tool change		1.0		
Process 12:	Machine pattern into outer dia.	2.0		
Processing Machine Type 6:	PVD Machine	5.0	2.0	3.0
Setup		2.0		
Process 12:	PVD Crown	3.0		
Back Case		*Proc. Times (min)*	*Setup Time /machine*	*Proc. Time /machine*
Raw Material:	Dia. 60 mm Stainless Steel Bar Stock	33.0	16.0	17.0

Processing Machine Type 5:	Feed-through CNC Lathe	20.0	10.0	10.0
Setup		3.0		
Process 1:	Face stock	1.0		
Tool change		1.0		
Process 2:	Turn down max dia.	1.0		
Tool change		1.0		
Process 3:	Turn down minor outside dia.	1.0		
Tool change		1.0		
Process 4:	Thread minor outside diameter	2.0		
Tool change		1.0		
Process 5:	Bore out inside dia.	2.0		
Tool change		1.0		
Process 6:	Part off	1.0		
Refixture & Tool change		2.0		
Process 7:	Shape end face	2.0		
Processing Machine Type 1:	4-Axis CNC Milling Machine	9.0	4.0	5.0
Setup		3.0		
Process 8:	Mill out tool gripping cavities	2.0		
Tool change		1.0		
Process 9:	Engraving	3.0		
Processing Machine Type 2:	Multi-purpose Grinding Machine	4.0	2.0	2.0
Setup		2.0		
Process 10:	Surface finishing	2.0		

Bezel		*Proc. Times (min)*	*Setup Time /machine*	*Proc. Time /machine*
Raw Material:	OD 50 mm x ID 20 mm Stainless Steel Tube Stock	36.0	19.5	16.5
Processing Machine Type 5:	Feed-through CNC Lathe	20.5	11.0	9.5
Setup		5.0		
Process 1:	Face stock	1.0		
Tool change		1.0		
Process 2:	Turn down maximum diameter	1.0		
Tool change		1.0		
Process 3:	Drill minimum inner diameter	2.0		
Tool change		1.0		
Process 4:	Bore out inner step with lip	2.0		
Tool change		1.0		
Process 5:	Part off	1.0		
Refixture & Tool change		2.0		
Process 6:	Shape end face	2.5		
Processing Machine Type 1:	4-Axis CNC Milling Machine	6.0	4.0	2.0
Setup		4.0		
Process 7:	Shape outer diameter profile	2.0		

Processing Machine Type 2:	Multi-purpose Grinding Machine	4.5	2.5	2.0
Setup		2.5		
Process 8:	Surface finishing	2.0		
Processing Machine Type 6:	PVD Machine	5.0	2.0	3.0
Setup		2.0		
Process 9:	PVD Outer Bezel	3.0		
Spacer		*Proc. Times (min)*	*Setup Time /machine*	*Proc. Time /machine*
Raw Material:	ABS plastic filament roll	27.0	2.0	25.0
Processing Machine Type 8:	3D Printing Machine	27.0	2.0	25.0
Setup		2.0		
Process 1:	Print spacer in ABS plastic	25.0		

Appendix B Capacity Planning Model Data and Results

B.1. Capacity planning model data

Table B.1. Wristwatch production processing requirement totals.

		1	2	3	4	5	6	Total	Arrival Rate (Orders/day)
Dial	**Station**	WS-M3	WS-M4	WS-M7	WS-A2	WS-A4	WS-A5		60.00
	Setup	2.5	5.0	2.0	1.0	2.0	1.0	13.5	
	Process	1.0	7.5	1.0	7.0	6.0	6.0	28.5	
	Total	3.5	12.5	3.0	8.0	8.0	7.0	42.0	
Case	**Station**	WS-M1	WS-M2	WS-M6	WS-A1	WS-A4	WS-A5		60.00
	Setup	21.0	2.5	2.0	1.0	2.0	1.0	29.5	
	Process	27.0	3.0	3.0	6.5	6.0	6.0	51.5	
	Total	48.0	5.5	5.0	7.5	8.0	7.0	81.0	
Bezel	**Station**	WS-M5	WS-M1	WS-M2	WS-M6	WS-A4	WS-A5		60.00
	Setup	11.0	4.0	2.5	2.0	2.0	1.0	22.5	
	Process	9.5	2.0	2.0	3.0	6.0	6.0	28.5	
	Total	20.5	6.0	4.5	5.0	8.0	7.0	51.0	
Back Case	**Station**	WS-M5	WS-M1	WS-M2	WS-A4	WS-A5			60.00
	Setup	10.0	4.0	2.0	2.0	1.0		19.0	
	Process	10.0	5.0	2.0	6.0	6.0		29.0	
	Total	20.0	9.0	4.0	8.0	7.0		48.0	
Crown	**Station**	WS-M5	WS-M1	WS-M6	WS-A3	WS-A4	WS-A5		60.00
	Setup	7.0	6.0	2.0	1.0	2.0	1.0	19.0	
	Process	10.0	4.0	3.0	3.0	6.0	6.0	32.0	
	Total	17.0	10.0	5.0	4.0	8.0	7.0	51.0	
Spacer	**Station**	WS-M8	WS-A4	WS-A5					60.00
	Setup	2.0	2.0	1.0				5.0	
	Process	25.0	6.0	6.0				37.0	
	Total	27.0	8.0	7.0				42.0	

Table B.2. Wristwatch production capacity requirements.

Minutes/Day:	480								
Workstations	**Capacity**	**Dial**	**Case**	**Bezel**	**Back Case**	**Crown**	**Spacer**	**Total (mins)**	**Expected Util**
4-Axis CNC Milling Machine (WS-M1)	10		2880.0	360.0	540.0	600.0		4380.0	91.3%
Multi-purpose Grinding Machine (WS-M2)	2		330.0	270.0	240.0			840.0	87.5%
Piercing and Blanking Machine (WS-M3)	1	210.0						210.0	43.8%
Pad Printing Machine (WS-M4)	2	750.0						750.0	78.1%
Feed-through CNC Lathe (WS-M5)	8			1230.0	1200.0	1020.0		3450.0	89.8%
PVD Machine (WS-M6)	2		300.0	300.0		300.0		900.0	93.8%
Stencil Gluing Machine (WS-M7)	1	180.0						180.0	37.5%
3D Printing Machine (WS-M8)	4						1620.0	1620.0	84.4%
Case Sub-assembly (WS-A1)	1		450.0					450.0	93.8%
Time Sub-assembly (WS-A2)	1	480.0						480.0	100.0%
Crown Sub-assembly (WS-A3)	1					240.0		240.0	50.0%
Watch Body Assembly & Inspection (WS-A4)	1	480.0	480.0	480.0	480.0	480.0	480.0	480.0	100.0%
Wrist Watch Assembly & Inspection (WS-A5)	1	420.0	420.0	420.0	420.0	420.0	420.0	420.0	87.5%
Machine Operators	11	570.0	1530.0	1170.0	960.0	900.0	120.0	5250.0	99.4%
Assembly Station Operators	5							2070.0	86.3%

B.2. Capacity planning model results

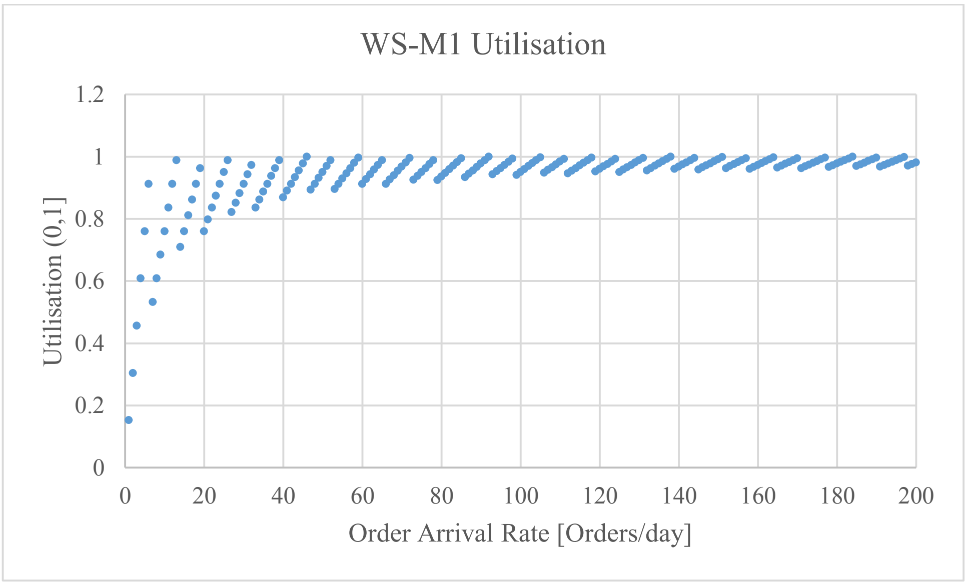


Figure B.1. WS-M1 utilisation versus order arrival rate.

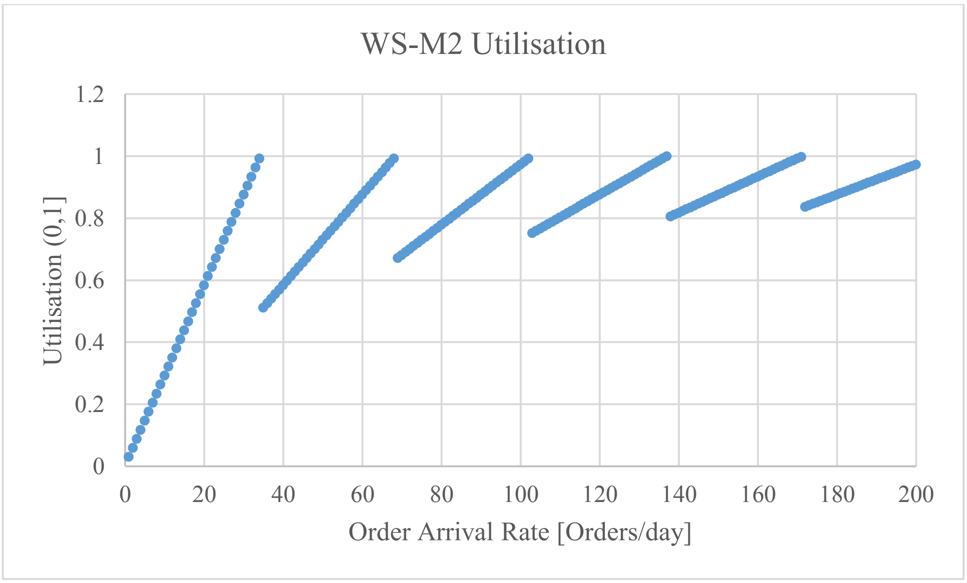


Figure B.2. WS-M2 utilisation versus order arrival rate.

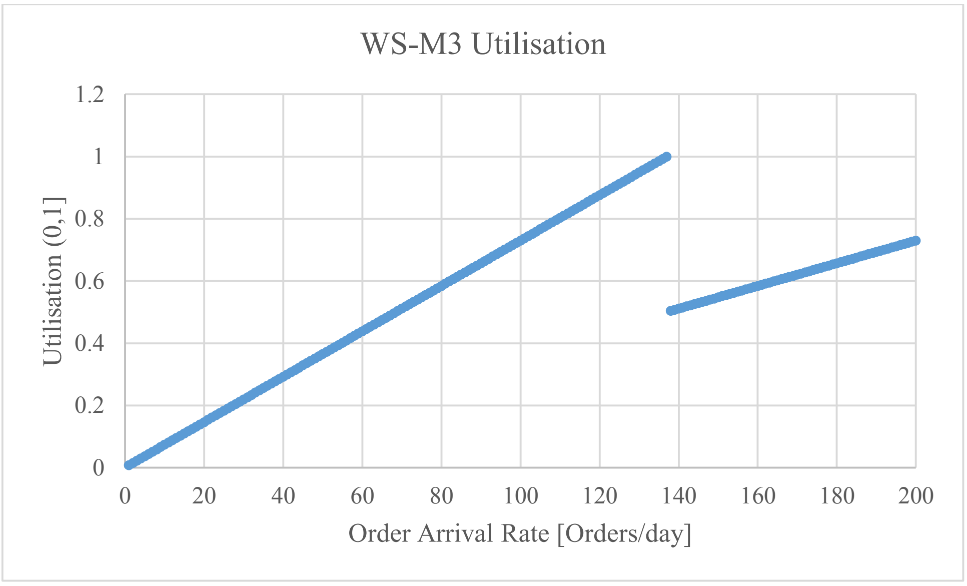


Figure B.3. WS-M3 utilisation versus order arrival rate.

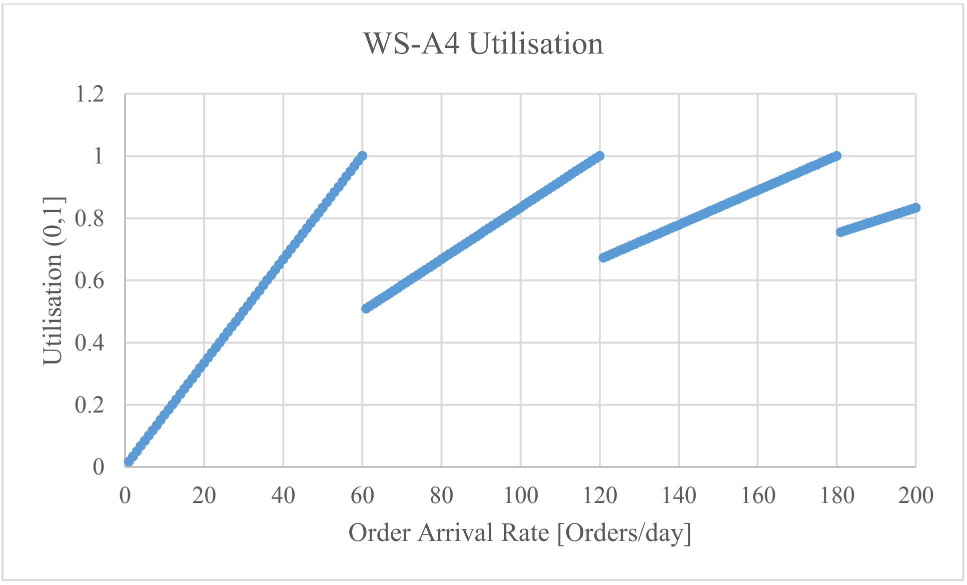


Figure B.4. WS-M4 utilisation versus order arrival rate.

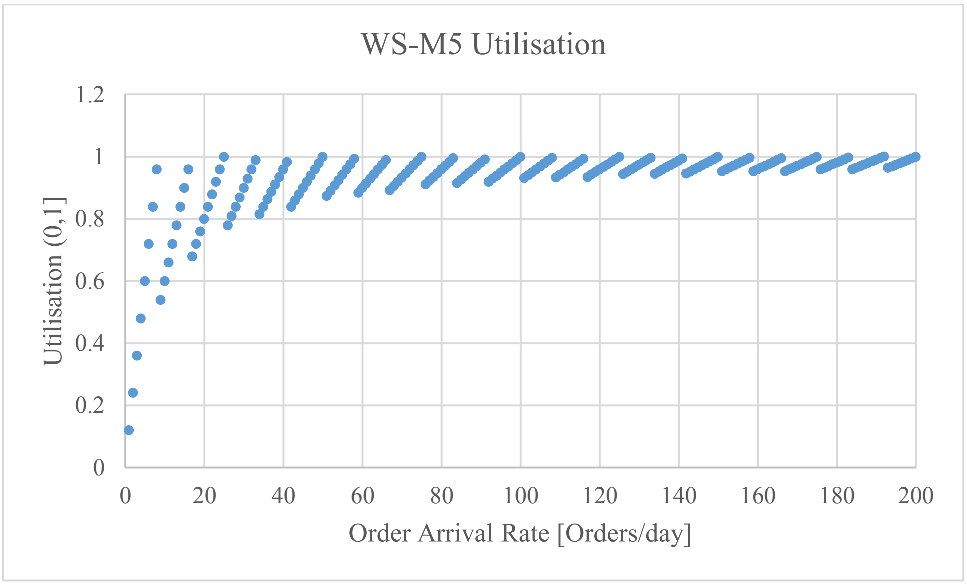


Figure B.5. WS-M5 utilisation versus order arrival rate.

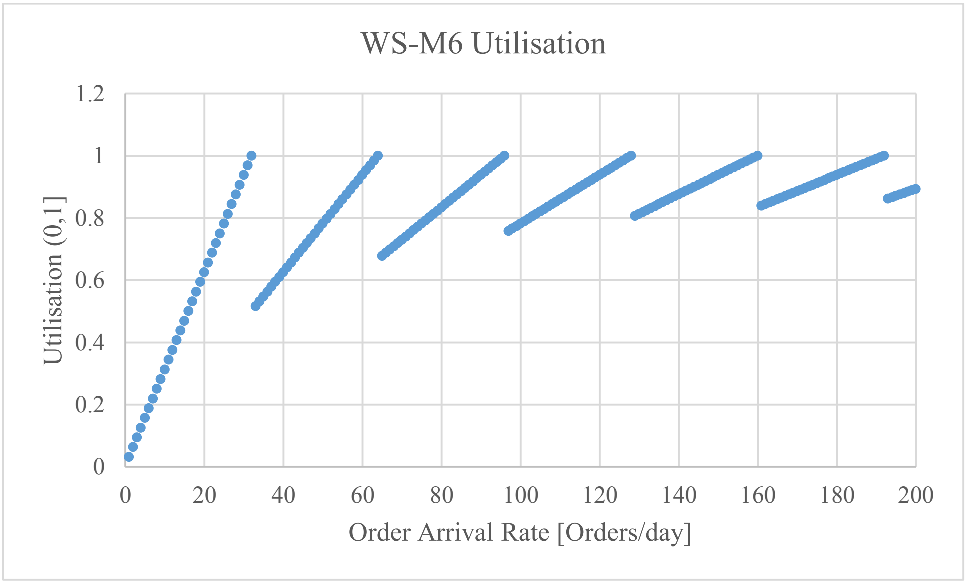


Figure B.6. WS-M6 utilisation versus order arrival rate.

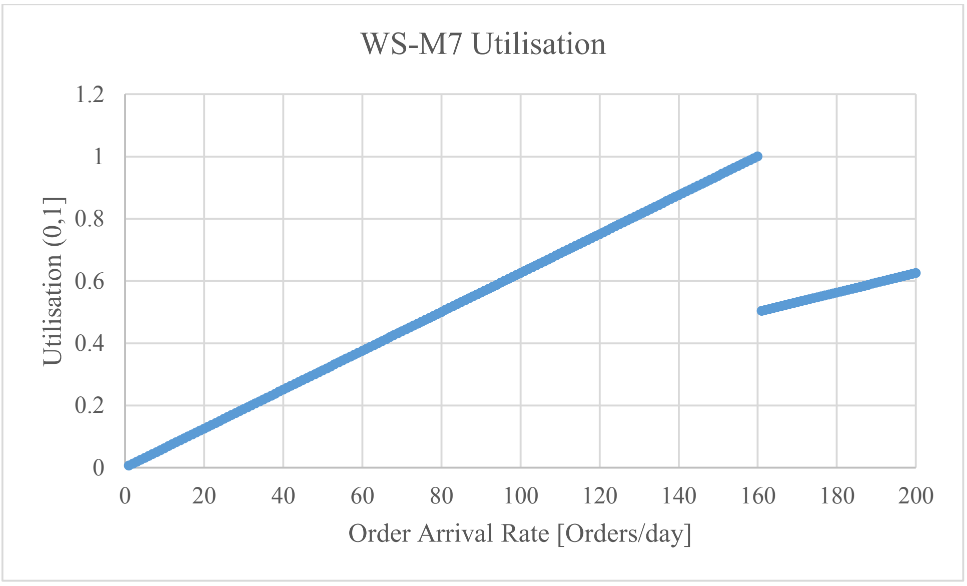


Figure B.7. WS-M7 utilisation versus order arrival rate.

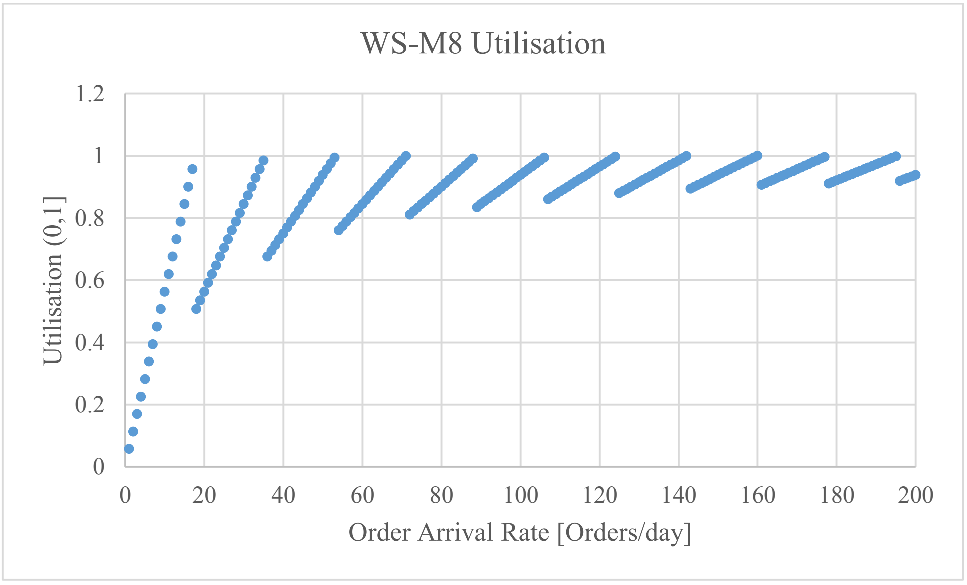


Figure B.8. WS-M8 utilisation versus order arrival rate.

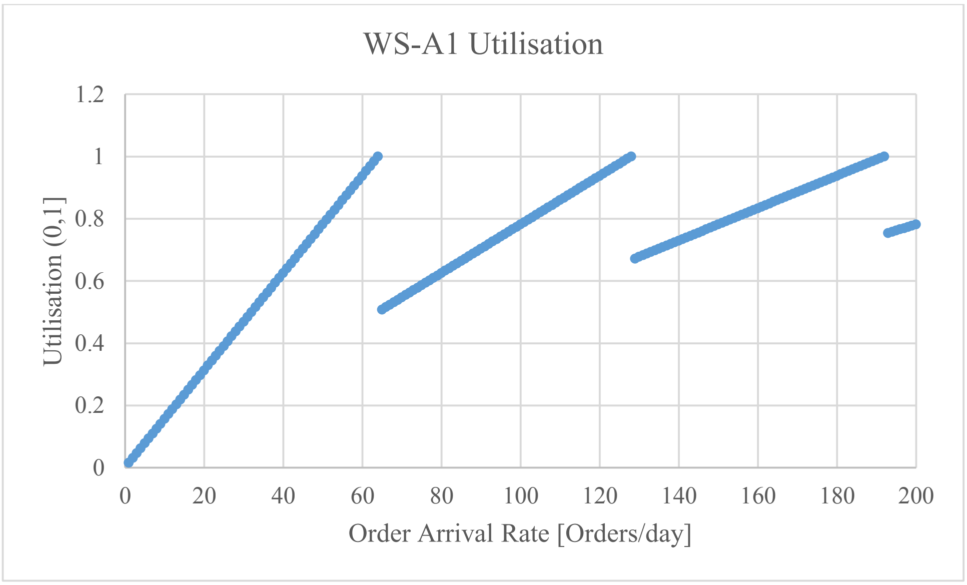


Figure B.9. WS-A1 utilisation versus order arrival rate.

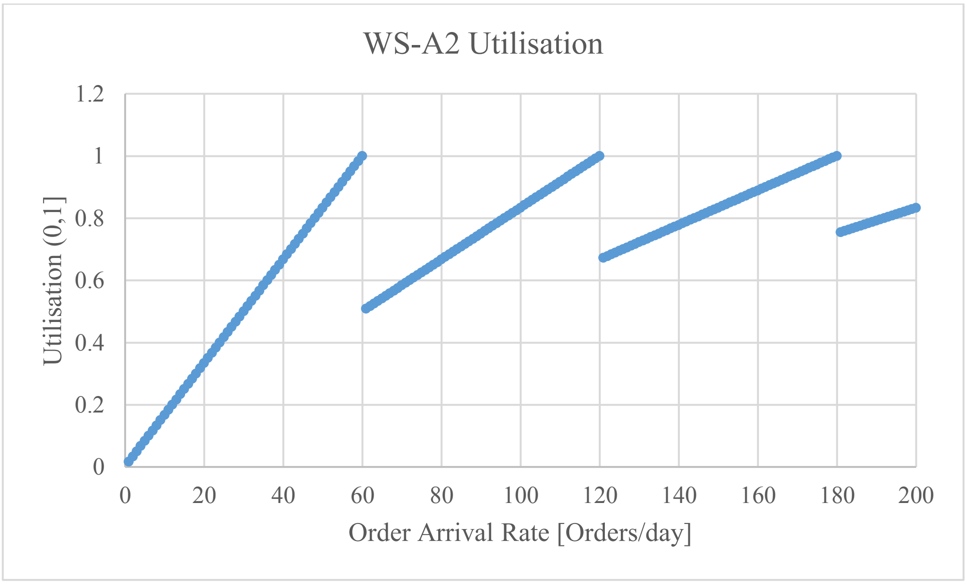


Figure B.10. WS-A2 utilisation versus order arrival rate.

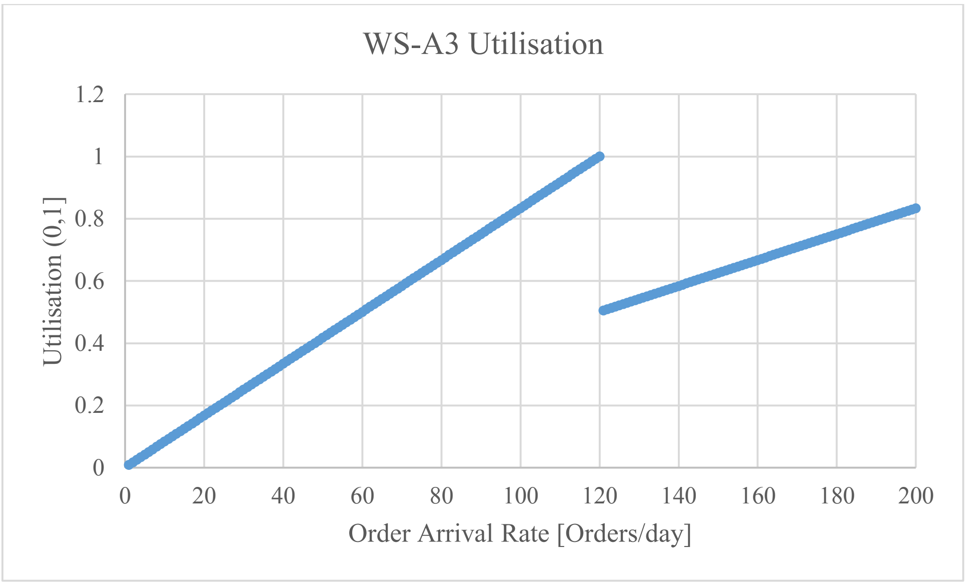


Figure B.11. WS-A3 utilisation versus order arrival rate.

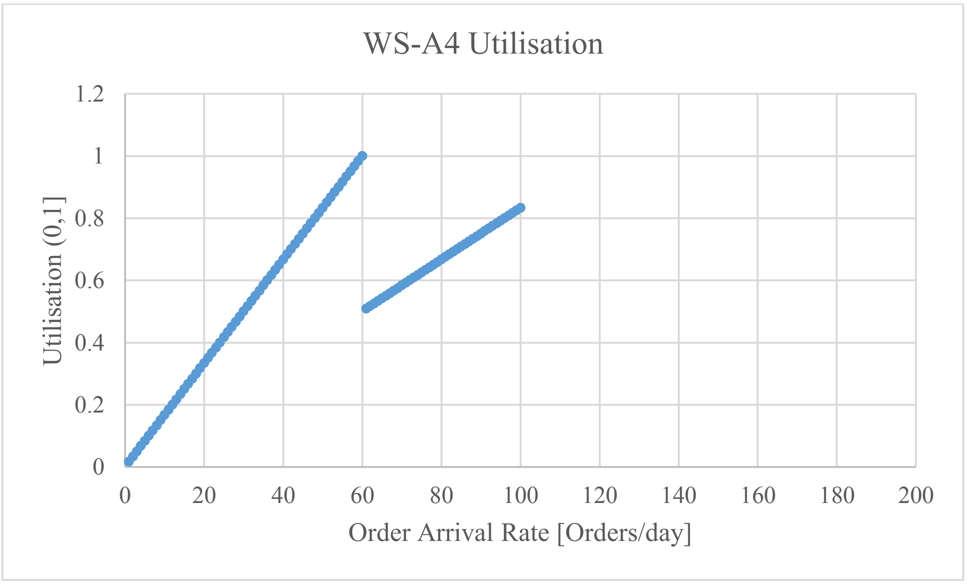


Figure B.12. WS-A4 utilisation versus order arrival rate.

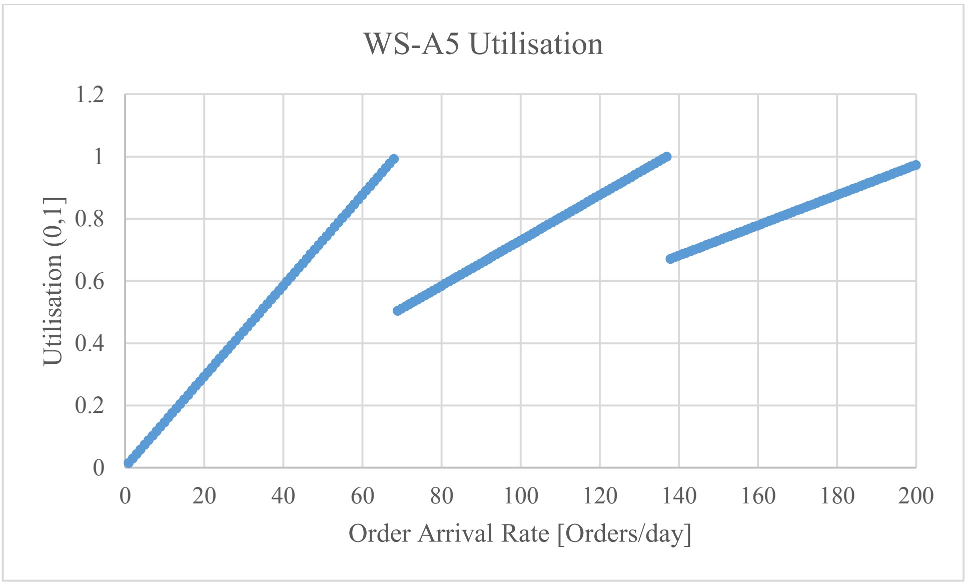


Figure B.13. WS-A5 utilisation versus order arrival rate.

Appendix C Capacity Planning Macro Code

C.1. Capacity plan generation code

```
Sub run()
MaxArrRate = 200
temp = setup(1, 1, MaxArrRate)
temp1 = MedianRange(MaxArrRate)
Worksheets("Capacity Optimisation").Activate
End Sub

' ------------------ FUNCTION FOR CALCULATING THE UTILISATION MEDIAN FOR ALL
WORKSTATIONS ------------------ '
Function MedianRange(MaxArrRate As Variant)
Dim Mdn As Variant
Dim Avg As Variant
Worksheets("Capacity Optimisation").Activate
For i = 0 To MaxArrRate - 1
Mdn = Application.WorksheetFunction.Median(Cells(2, 3 + 2 * i), Cells(3, 3 +
2 * i), Cells(4, 3 + 2 * i), Cells(5, 3 + 2 * i), Cells(6, 3 + 2 * i),
Cells(7, 3 + 2 * i), Cells(8, 3 + 2 * i), Cells(9, 3 + 2 * i), Cells(10, 3 +
2 * i), Cells(11, 3 + 2 * i), Cells(12, 3 + 2 * i), Cells(13, 3 + 2 * i),
Cells(14, 3 + 2 * i))
Avg = Application.WorksheetFunction.Average(Cells(2, 3 + 2 * i), Cells(3, 3 +
2 * i), Cells(4, 3 + 2 * i), Cells(5, 3 + 2 * i), Cells(6, 3 + 2 * i),
Cells(7, 3 + 2 * i), Cells(8, 3 + 2 * i), Cells(9, 3 + 2 * i), Cells(10, 3 +
2 * i), Cells(11, 3 + 2 * i), Cells(12, 3 + 2 * i), Cells(13, 3 + 2 * i),
Cells(14, 3 + 2 * i))
Cells(21, 2 + 2 * i).Activate
ActiveCell.Value = Mdn
Worksheets("Plotting").Cells(3 + i, 28).Value = Mdn
Worksheets("Plotting").Cells(3 + i, 29).Value = Mdn
Next i
End Function

' ---------------------- FUNCTION FOR SETTING UP THE DATA FOR OPTIMISATION
ALGORITHM ---------------------- '
Function setup(StartArrRate As Variant, ArrRateInterval As Variant,
EndArrRate As Variant)
'StartArrRate    : Order arrival rate at start of iterations [orders/day]
'ArrRateInterval : Interval between order arrival rate iterations
'EndArrRate      : Order arrival rate at end of iterations [orders/day]

' PREPARING OPTIMISATION WORKSHEET
Worksheets("Capacity Optimisation").Activate
Range("B2:ZZ14").Clear
Range("B16:ZZ17").Clear
Range("B19:ZZ30").Clear
Worksheets("Plotting").Range("A3:ZZ9999").Clear

' Activate the display worksheet
```

```
Worksheets("Initial Capacity Model").Activate

' TEST THE ARRIVAL RATE RANGE FOR DIVISION BY THE INTERVAL
remainder = (EndArrRate - StartArrRate) Mod ArrRateInterval
If remainder = 0 Then
' IF THE REMAINDER IS ZERO THEN CARRY ON WITH CALCULATIONS
Iterations = (EndArrRate - StartArrRate) / ArrRateInterval
ArrivalRate = StartArrRate

' VARIABLE INITIALISATION
Dim Dial(1, 5) As Double            ' M3-M4-M7-A2-A4-A5
Dim WatchCase(1, 5) As Double       ' M1-M2-M6-A1-A4-A5
Dim Bezel(1, 5) As Double           ' M5-M1-M2-M6-A4-A5
Dim BackCase(1, 5) As Double        ' M5-M1-M2-A4-A5
Dim Crown(1, 5) As Double           ' M5-M1-M6-A3-A4-A5
Dim Spacer(1, 5) As Double          ' M8-A4-A5
MinutesPerDay = Range("B1").Value   ' Minutes per shift
Dim WSMCapacities(7) As Double      ' Machining workstation capacities
Dim WSACapacities(4) As Double      ' Assembly workstation capacities

' INITIAL WORKSTATION AND OPERATOR CAPACITIES
For i = 0 To 7
WSMCapacities(i) = 1
Next i
For i = 0 To 4
WSACapacities(i) = 1
Next i
WSMOperatorCapacity = 1
WSAOperatorCapacity = 1
' WS setup/operator times
Dim WSMSetupTimes(7) As Double
Dim WSASetupTimes(4) As Double
' WS-M1 though 8 processing times and utilisations
Dim WSMProcessingTimes(7) As Double
Dim WSAProcessingTimes(4) As Double
' WS-M1 though 8 processing times and utilisations
Dim WSMUtil(7) As Double
Dim WSAUtil(4) As Double

' READ IN DIAL SETUP & PROCESSING TIMES
Range("C22").Activate
For j = 0 To 1
    For i = 0 To 5
        Dial(j, i) = ActiveCell.Value
        Cells(ActiveCell.Row, ActiveCell.Column + 1).Activate
    Next i
    Cells(ActiveCell.Row + 1, ActiveCell.Column - i).Activate
Next j

' READ IN CASE SETUP & PROCESSING TIMES
Range("C27").Activate
For j = 0 To 1
```

```
    For i = 0 To 5
        WatchCase(j, i) = ActiveCell.Value
        Cells(ActiveCell.Row, ActiveCell.Column + 1).Activate
    Next i
    Cells(ActiveCell.Row + 1, ActiveCell.Column - i).Activate
Next j

' READ IN BEZEL SETUP & PROCESSING TIMES
Range("C32").Activate
For j = 0 To 1
    For i = 0 To 5
        Bezel(j, i) = ActiveCell.Value
        Cells(ActiveCell.Row, ActiveCell.Column + 1).Activate
    Next i
    Cells(ActiveCell.Row + 1, ActiveCell.Column - i).Activate
Next j

' READ IN BACK CASE SETUP & PROCESSING TIMES
Range("C37").Activate
For j = 0 To 1
    For i = 0 To 5
        BackCase(j, i) = ActiveCell.Value
        Cells(ActiveCell.Row, ActiveCell.Column + 1).Activate
    Next i
    Cells(ActiveCell.Row + 1, ActiveCell.Column - i).Activate
Next j

' READ IN CROWN SETUP & PROCESSING TIMES
Range("C42").Activate
For j = 0 To 1
    For i = 0 To 5
        Crown(j, i) = ActiveCell.Value
        Cells(ActiveCell.Row, ActiveCell.Column + 1).Activate
    Next i
    Cells(ActiveCell.Row + 1, ActiveCell.Column - i).Activate
Next j

' READ IN SPACER SETUP & PROCESSING TIMES
Range("C47").Activate
For j = 0 To 1
    For i = 0 To 2
        Spacer(j, i) = ActiveCell.Value
        Cells(ActiveCell.Row, ActiveCell.Column + 1).Activate
    Next i
    Cells(ActiveCell.Row + 1, ActiveCell.Column - i).Activate
Next j

' -------------------------- MAIN LOOP STARTS HERE ------------------------
--- '

' ITERATION COUNTER FOR MAIN LOOP
For counter = 0 To (Iterations)
```

```
ArrivalRate = StartArrRate + (counter * ArrRateInterval)
Cells(21, 10).Value = ArrivalRate
' PRINT OUT ARRIVAL RATE VALUES FOR OPTIMISATION CALCULATIONS
Worksheets("Capacity Optimisation").Cells(19, 2 + (2 * counter)).Value =
ArrivalRate
' PRINT OUT ARRIVAL RATE VALUES VALUES FOR PLOTTING
Worksheets("Plotting").Cells(3 + counter, 1).Value = ArrivalRate

' CALCULATE TOTAL MACHINING WORKSTATION OCCUPIED TIMES
WSMProcessingTimes(0) = (WatchCase(0, 0) + WatchCase(1, 0) + Bezel(0, 1) +
Bezel(1, 1) + BackCase(0, 1) + BackCase(1, 1) + Crown(0, 1) + Crown(1, 1)) *
ArrivalRate
WSMProcessingTimes(1) = (WatchCase(0, 1) + WatchCase(1, 1) + Bezel(0, 2) +
Bezel(1, 2) + BackCase(0, 2) + BackCase(1, 2)) * ArrivalRate
WSMProcessingTimes(2) = (Dial(0, 0) + Dial(1, 0)) * ArrivalRate
WSMProcessingTimes(3) = (Dial(0, 1) + Dial(1, 1)) * ArrivalRate
WSMProcessingTimes(4) = (Bezel(0, 0) + Bezel(1, 0) + BackCase(0, 0) +
BackCase(1, 0) + Crown(0, 0) + Crown(1, 0)) * ArrivalRate
WSMProcessingTimes(5) = (WatchCase(0, 2) + WatchCase(1, 2) + Bezel(0, 3) +
Bezel(1, 3) + Crown(0, 2) + Crown(1, 2)) * ArrivalRate
WSMProcessingTimes(6) = (Dial(0, 2) + Dial(1, 2)) * ArrivalRate
WSMProcessingTimes(7) = (Spacer(0, 0) + Spacer(1, 0)) * ArrivalRate

' CALCULATE TOTAL ASSEMBLY WORKSTATION OCCUPIED TIMES
WSAProcessingTimes(0) = (WatchCase(0, 3) + WatchCase(1, 3)) * ArrivalRate
WSAProcessingTimes(1) = (Dial(0, 3) + Dial(1, 3)) * ArrivalRate
WSAProcessingTimes(2) = (Crown(0, 3) + Crown(1, 3)) * ArrivalRate
WSAProcessingTimes(3) = (WatchCase(0, 4) + WatchCase(1, 4)) * ArrivalRate
WSAProcessingTimes(4) = (WatchCase(0, 5) + WatchCase(1, 5)) * ArrivalRate

' PRINT OUT CALCULATED MACHINING TIME VALUES IN MODEL SHEET
Cells(3, 9).Activate
For i = 0 To 7
    ActiveCell.Value = WSMProcessingTimes(i)
    Cells(4 + i, 9).Activate
Next i

' PRINT OUT CALCULATED ASSEMBLY TIME VALUES
For i = 0 To 4
    ActiveCell.Value = WSAProcessingTimes(i)
    Cells(12 + i, 9).Activate
Next i

' CALCULATE MACHINING WORKSTATION UTILISATION VALUES
Worksheets("Initial Capacity Model").Activate
Cells(3, 10).Activate
For i = 0 To 7
    WSMUtil(i) = WSMProcessingTimes(i) / (WSMCapacities(i) * MinutesPerDay)
    Do While WSMUtil(i) > 1
        WSMCapacities(i) = WSMCapacities(i) + 1
        WSMUtil(i) = WSMProcessingTimes(i) / (WSMCapacities(i) *
MinutesPerDay)
```

```
    Loop
    ActiveCell.Value = WSMUtil(i)
' PRINT OUT MACHINING WORKSTATION UTILISATION VALUES FOR OPTIMISATION
CALCULATIONS
    Worksheets("Capacity Optimisation").Cells(2 + i, 3 + (2 * counter)).Value
= WSMUtil(i)
' PRINT OUT MACHINING WORKSTATION UTILISATION VALUES FOR PLOTTING
    Worksheets("Plotting").Cells(3 + counter, 3 + (2 * i)).Value = WSMUtil(i)
    Cells(4 + i, 10).Activate
Next i

' CALCULATE ASSEMBLY WORKSTATION UTILISATION VALUES
For i = 0 To 4
    WSAUtil(i) = WSAProcessingTimes(i) / (WSACapacities(i) * MinutesPerDay)
    Do While WSAUtil(i) > 1
        WSACapacities(i) = WSACapacities(i) + 1
        WSAUtil(i) = WSAProcessingTimes(i) / (WSACapacities(i) *
MinutesPerDay)
    Loop
    ActiveCell.Value = WSAUtil(i)
' PRINT OUT ASSEMBLY WORKSTATION UTILISATION VALUES FOR OPTIMISATION
CALCULATIONS
    Worksheets("Capacity Optimisation").Cells(10 + i, 3 + (2 *
counter)).Value = WSAUtil(i)
' PRINT OUT ASSEMBLY WORKSTATION UTILISATION VALUES FOR PLOTTING
    Worksheets("Plotting").Cells(3 + counter, 19 + (2 * i)).Value =
WSAUtil(i)
    Cells(12 + i, 10).Activate
Next i

' PRINT OUT MACHINING WORKSTATION CAPACITIES
Cells(3, 2).Activate
For i = 0 To 7
    ActiveCell.Value = WSMCapacities(i)
' -- FOR OPTIMISATION CALCULATIONS
    Worksheets("Capacity Optimisation").Cells(2 + i, 2 + (2 * counter)).Value
= WSMCapacities(i)
' -- FOR PLOTTING
    Worksheets("Plotting").Cells(3 + counter, 2 + (2 * i)).Value =
WSMCapacities(i)
    Cells(4 + i, 2).Activate
Next i

' PRINT OUT ASSEMBLY WORKSTATION CAPACITIES
For i = 0 To 4
    ActiveCell.Value = WSACapacities(i)
' -- FOR OPTIMISATION CALCULATIONS
    Worksheets("Capacity Optimisation").Cells(10 + i, 2 + (2 *
counter)).Value = WSACapacities(i)
' -- FOR PLOTTING
    Worksheets("Plotting").Cells(3 + counter, 18 + (2 * i)).Value =
WSACapacities(i)
```

```
    Cells(12 + i, 2).Activate
Next i

' CALCULATE MACHINING WORKSTATION SETUP TIMES
WSMSetupTimes(0) = (WatchCase(0, 0) + Bezel(0, 1) + BackCase(0, 1) + Crown(0,
1)) * ArrivalRate
WSMSetupTimes(1) = (WatchCase(0, 1) + Bezel(0, 2) + BackCase(0, 2)) *
ArrivalRate
WSMSetupTimes(2) = (Dial(0, 0)) * ArrivalRate
WSMSetupTimes(3) = (Dial(0, 1)) * ArrivalRate
WSMSetupTimes(4) = (Bezel(0, 0) + BackCase(0, 0) + Crown(0, 0)) * ArrivalRate
WSMSetupTimes(5) = (WatchCase(0, 2) + Bezel(0, 3) + Crown(0, 2)) *
ArrivalRate
WSMSetupTimes(6) = (Dial(0, 2)) * ArrivalRate
WSMSetupTimes(7) = (Spacer(0, 0)) * ArrivalRate

' TOTAL AND PRINT OUT THE SETUP TIME FOR MACHINE OPERATOR UTILISATION
TotalWSMSetupTime = 0
For i = 0 To 7
    TotalWSMSetupTime = TotalWSMSetupTime + WSMSetupTimes(i)
Next i

Range("I17").Activate
ActiveCell.Value = TotalWSMSetupTime
'Worksheets("Capacity Optimisation").Cells(16, 3 + (3 * counter)).Value =
TotalWSMSetupTime

' CALCULATE AND PRINT OUT MACHINE OPERATOR UTILISATION AND CAPACITY
WSMOperatorUtil = TotalWSMSetupTime / (WSMOperatorCapacity * MinutesPerDay)
Do While WSMOperatorUtil > 1
    WSMOperatorCapacity = WSMOperatorCapacity + 1
    WSMOperatorUtil = TotalWSMSetupTime / (WSMOperatorCapacity *
MinutesPerDay)
Loop

Range("J17").Activate
ActiveCell.Value = WSMOperatorUtil
Worksheets("Capacity Optimisation").Cells(16, 3 + (2 * counter)).Value =
WSMOperatorUtil
Range("B17").Activate
ActiveCell.Value = WSMOperatorCapacity
Worksheets("Capacity Optimisation").Cells(16, 2 + (2 * counter)).Value =
WSMOperatorCapacity

' CALCULATE TOTAL ASSEMBLY WORKSTATION OCCUPIED TIMES
WSASetupTimes(0) = (WatchCase(0, 3) + WatchCase(1, 3)) * ArrivalRate
WSASetupTimes(1) = (Dial(0, 3) + Dial(1, 3)) * ArrivalRate
WSASetupTimes(2) = (Crown(0, 3) + Crown(1, 3)) * ArrivalRate
WSASetupTimes(3) = (WatchCase(0, 4) + WatchCase(1, 4)) * ArrivalRate
WSASetupTimes(4) = (WatchCase(0, 5) + WatchCase(1, 5)) * ArrivalRate

' TOTAL AND PRINT OUT THE ASSEMBLY TIME FOR OPERATOR UTILISATION
```

```
TotalWSASetupTime = 0
For i = 0 To 4
    TotalWSASetupTime = TotalWSASetupTime + WSASetupTimes(i)
Next i

Range("I18").Activate
ActiveCell.Value = TotalWSASetupTime
'Worksheets("Capacity Optimisation").Cells(17, 3 + (3 * counter)).Value =
TotalWSASetupTime

' CALCULATE AND PRINT OUT ASSEMBLY OPERATOR UTILISATION AND CAPACITY
WSAOperatorUtil = TotalWSASetupTime / (WSAOperatorCapacity * MinutesPerDay)
Do While WSAOperatorUtil > 1
    WSAOperatorCapacity = WSAOperatorCapacity + 1
    WSAOperatorUtil = TotalWSASetupTime / (WSAOperatorCapacity *
MinutesPerDay)
Loop

Range("J18").Activate
ActiveCell.Value = WSAOperatorUtil
Worksheets("Capacity Optimisation").Cells(17, 3 + (2 * counter)).Value =
WSAOperatorUtil
Range("B18").Activate
ActiveCell.Value = WSAOperatorCapacity
Worksheets("Capacity Optimisation").Cells(17, 2 + (2 * counter)).Value =
WSAOperatorCapacity
Next counter

' IF THE REMAINDER IS NOT ZERO THEN DISPLAY AN ERROR MESSAGE
Else
MsgBox ("ERROR: Your Arrival Rate range must be divisible by the Interval
without a remainder")
End If

End Function
```

C.2. Optimal capacity plan search code

```
Function optimise()

' VARIABLE INITIALISATION
Dim Dial(1, 5) As Double            ' M3-M4-M7-A2-A4-A5
Dim WatchCase(1, 5) As Double       ' M1-M2-M6-A1-A4-A5
Dim Bezel(1, 5) As Double           ' M5-M1-M2-M6-A4-A5
Dim BackCase(1, 5) As Double        ' M5-M1-M2-A4-A5
Dim Crown(1, 5) As Double           ' M5-M1-M6-A3-A4-A5
Dim Spacer(1, 5) As Double          ' M8-A4-A5
Dim i As Integer                    ' Iterating counter 1
Dim j As Integer                    ' Iterating counter 2
ArrivalRate = Range("J21").Value    ' Order arrival rate [orders/day]
MinutesPerDay = Range("B1").Value   ' 480 minutes in an 8-hour shift
Dim WSMCapacities(7) As Double      ' Machining workstation capacities
Dim WSACapacities(4) As Double      ' Assembly workstation capacities

' INITIAL WORKSTATION AND OPERATOR CAPACITIES
For i = 0 To 7
WSMCapacities(i) = 1
Next i
For i = 0 To 4
WSACapacities(i) = 1
Next i
WSMOperatorCapacity = 1
WSAOperatorCapacity = 1
' WS setup/operator times
Dim WSMSetupTimes(7) As Double
Dim WSASetupTimes(4) As Double
' WS-M1 though 8 processing times and utilisations
Dim WSMProcessingTimes(7) As Double
Dim WSAProcessingTimes(4) As Double
' WS-M1 though 8 processing times and utilisations
Dim WSMUtil(7) As Double
Dim WSAUtil(4) As Double

' Activate the correct worksheet
Worksheets("Initial Capacity Model").Activate

' READ IN DIAL SETUP & PROCESSING TIMES
Range("C22").Activate
For j = 0 To 1
    For i = 0 To 5
        Dial(j, i) = ActiveCell.Value
        Cells(ActiveCell.Row, ActiveCell.Column + 1).Activate
    Next i
        Cells(ActiveCell.Row + 1, ActiveCell.Column - i).Activate
Next j

' READ IN CASE SETUP & PROCESSING TIMES
Range("C27").Activate
```

```
For j = 0 To 1
    For i = 0 To 5
        WatchCase(j, i) = ActiveCell.Value
        Cells(ActiveCell.Row, ActiveCell.Column + 1).Activate
    Next i
        Cells(ActiveCell.Row + 1, ActiveCell.Column - i).Activate
Next j

' READ IN BEZEL SETUP & PROCESSING TIMES
Range("C32").Activate
For j = 0 To 1
    For i = 0 To 5
        Bezel(j, i) = ActiveCell.Value
        Cells(ActiveCell.Row, ActiveCell.Column + 1).Activate
    Next i
        Cells(ActiveCell.Row + 1, ActiveCell.Column - i).Activate
Next j

' READ IN BACK CASE SETUP & PROCESSING TIMES
Range("C37").Activate
For j = 0 To 1
    For i = 0 To 5
        BackCase(j, i) = ActiveCell.Value
        Cells(ActiveCell.Row, ActiveCell.Column + 1).Activate
    Next i
        Cells(ActiveCell.Row + 1, ActiveCell.Column - i).Activate
Next j

' READ IN CROWN SETUP & PROCESSING TIMES
Range("C42").Activate
For j = 0 To 1
    For i = 0 To 5
        Crown(j, i) = ActiveCell.Value
        Cells(ActiveCell.Row, ActiveCell.Column + 1).Activate
    Next i
        Cells(ActiveCell.Row + 1, ActiveCell.Column - i).Activate
Next j

' READ IN SPACER SETUP & PROCESSING TIMES
Range("C47").Activate
For j = 0 To 1
    For i = 0 To 2
        Spacer(j, i) = ActiveCell.Value
        Cells(ActiveCell.Row, ActiveCell.Column + 1).Activate
    Next i
        Cells(ActiveCell.Row + 1, ActiveCell.Column - i).Activate
Next j

' CALCULATE TOTAL MACHINING WORKSTATION OCCUPIED TIMES
WSMProcessingTimes(0) = (WatchCase(0, 0) + WatchCase(1, 0) + Bezel(0, 1) +
Bezel(1, 1) + BackCase(0, 1) + BackCase(1, 1) + Crown(0, 1) + Crown(1, 1)) *
ArrivalRate
```

```
WSMProcessingTimes(1) = (WatchCase(0, 1) + WatchCase(1, 1) + Bezel(0, 2) +
Bezel(1, 2) + BackCase(0, 2) + BackCase(1, 2)) * ArrivalRate
WSMProcessingTimes(2) = (Dial(0, 0) + Dial(1, 0)) * ArrivalRate
WSMProcessingTimes(3) = (Dial(0, 1) + Dial(1, 1)) * ArrivalRate
WSMProcessingTimes(4) = (Bezel(0, 0) + Bezel(1, 0) + BackCase(0, 0) +
BackCase(1, 0) + Crown(0, 0) + Crown(1, 0)) * ArrivalRate
WSMProcessingTimes(5) = (WatchCase(0, 2) + WatchCase(1, 2) + Bezel(0, 3) +
Bezel(1, 3) + Crown(0, 2) + Crown(1, 2)) * ArrivalRate
WSMProcessingTimes(6) = (Dial(0, 2) + Dial(1, 2)) * ArrivalRate
WSMProcessingTimes(7) = (Spacer(0, 0) + Spacer(1, 0)) * ArrivalRate

' CALCULATE TOTAL ASSEMBLY WORKSTATION OCCUPIED TIMES
WSAProcessingTimes(0) = (WatchCase(0, 3) + WatchCase(1, 3)) * ArrivalRate
WSAProcessingTimes(1) = (Dial(0, 3) + Dial(1, 3)) * ArrivalRate
WSAProcessingTimes(2) = (Crown(0, 3) + Crown(1, 3)) * ArrivalRate
WSAProcessingTimes(3) = (WatchCase(0, 4) + WatchCase(1, 4)) * ArrivalRate
WSAProcessingTimes(4) = (WatchCase(0, 5) + WatchCase(1, 5)) * ArrivalRate

' PRINT OUT CALCULATED MACHINING TIME VALUES
Cells(3, 9).Activate
For i = 0 To 7
    ActiveCell.Value = WSMProcessingTimes(i)
    Cells(4 + i, 9).Activate
Next i

' PRINT OUT CALCULATED ASSEMBLY TIME VALUES
For i = 0 To 4
    ActiveCell.Value = WSAProcessingTimes(i)
    Cells(12 + i, 9).Activate
Next i

' CALCULATE AND PRINT OUT MACHINING WORKSTATION UTILISATION VALUES
Cells(3, 10).Activate
For i = 0 To 7
    WSMUtil(i) = WSMProcessingTimes(i) / (WSMCapacities(i) * MinutesPerDay)
    Do While WSMUtil(i) > 1
        WSMCapacities(i) = WSMCapacities(i) + 1
        WSMUtil(i) = WSMProcessingTimes(i) / (WSMCapacities(i) *
MinutesPerDay)
    Loop
    ActiveCell.Value = WSMUtil(i)
    Cells(4 + i, 10).Activate
Next i

' CALCULATE AND PRINT OUT ASSEMBLY WORKSTATION UTILISATION VALUES
For i = 0 To 4
    WSAUtil(i) = WSAProcessingTimes(i) / (WSACapacities(i) * MinutesPerDay)
    Do While WSAUtil(i) > 1
        WSACapacities(i) = WSACapacities(i) + 1
        WSAUtil(i) = WSAProcessingTimes(i) / (WSACapacities(i) *
MinutesPerDay)
    Loop
    ActiveCell.Value = WSAUtil(i)
    Cells(12 + i, 10).Activate
Next i

' PRINT OUT MACHINING WORKSTATION CAPACITIES
Cells(3, 2).Activate
For i = 0 To 7
    ActiveCell.Value = WSMCapacities(i)
    Cells(4 + i, 2).Activate
Next i

' PRINT OUT ASSEMBLY WORKSTATION CAPACITIES
For i = 0 To 4
    ActiveCell.Value = WSACapacities(i)
    Cells(12 + i, 2).Activate
Next i

' CALCULATE TOTAL MACHINING WORKSTATION OCCUPIED TIMES
WSMSetupTimes(0) = (WatchCase(0, 0) + Bezel(0, 1) + BackCase(0, 1) + Crown(0,
1)) * ArrivalRate
WSMSetupTimes(1) = (WatchCase(0, 1) + Bezel(0, 2) + BackCase(0, 2)) *
ArrivalRate
WSMSetupTimes(2) = (Dial(0, 0)) * ArrivalRate
WSMSetupTimes(3) = (Dial(0, 1)) * ArrivalRate
WSMSetupTimes(4) = (Bezel(0, 0) + BackCase(0, 0) + Crown(0, 0)) * ArrivalRate
WSMSetupTimes(5) = (WatchCase(0, 2) + Bezel(0, 3) + Crown(0, 2)) *
ArrivalRate
WSMSetupTimes(6) = (Dial(0, 2)) * ArrivalRate
WSMSetupTimes(7) = (Spacer(0, 0)) * ArrivalRate

' TOTAL AND PRINT OUT THE SETUP TIME FOR MACHINE OPERATOR UTILISATION
For i = 0 To 7
    TotalWSMSetupTime = TotalWSMSetupTime + WSMSetupTimes(i)
Next i
Range("I17").Activate
ActiveCell.Value = TotalWSMSetupTime

' CALCULATE AND PRINT OUT MACHINE OPERATOR UTILISATION AND CAPACITY
WSMOperatorUtil = TotalWSMSetupTime / (WSMOperatorCapacity * MinutesPerDay)
Do While WSMOperatorUtil > 1
    WSMOperatorCapacity = WSMOperatorCapacity + 1
    WSMOperatorUtil = TotalWSMSetupTime / (WSMOperatorCapacity *
MinutesPerDay)
Loop
Range("J17").Activate
ActiveCell.Value = WSMOperatorUtil
Range("B17").Activate
ActiveCell.Value = WSMOperatorCapacity

' CALCULATE TOTAL ASSEMBLY WORKSTATION OCCUPIED TIMES
WSASetupTimes(0) = (WatchCase(0, 3) + WatchCase(1, 3)) * ArrivalRate
WSASetupTimes(1) = (Dial(0, 3) + Dial(1, 3)) * ArrivalRate
WSASetupTimes(2) = (Crown(0, 3) + Crown(1, 3)) * ArrivalRate
WSASetupTimes(3) = (WatchCase(0, 4) + WatchCase(1, 4)) * ArrivalRate
WSASetupTimes(4) = (WatchCase(0, 5) + WatchCase(1, 5)) * ArrivalRate

' TOTAL AND PRINT OUT THE ASSEMBLY TIME FOR OPERATOR UTILISATION
For i = 0 To 4
    TotalWSASetupTime = TotalWSASetupTime + WSASetupTimes(i)
Next i
Range("I18").Activate
ActiveCell.Value = TotalWSASetupTime

' CALCULATE AND PRINT OUT ASSEMBLY OPERATOR UTILISATION AND CAPACITY
WSAOperatorUtil = TotalWSASetupTime / (WSAOperatorCapacity * MinutesPerDay)
Do While WSAOperatorUtil > 1
    WSAOperatorCapacity = WSAOperatorCapacity + 1
    WSAOperatorUtil = TotalWSASetupTime / (WSAOperatorCapacity *
MinutesPerDay)
Loop
Range("J18").Activate
ActiveCell.Value = WSAOperatorUtil
Range("B18").Activate
ActiveCell.Value = WSAOperatorCapacity

End Function
```

Appendix D Aggregate Production Planning Optimisation Algorithm Code

D.1. Simulated Annealing algorithm code

```
/*
Main programme source file
Author:             Louw Butler
Created:     09/03/2015
Modified:    27/04/2015
*/

/* User-defined header (includes all standard library headers) */
#include "sources.h"
#include <array>

using namespace std;
using std::cout;

/* MAIN PROGRAMME */
int main() {
	/** Reading and storing production cost input variables from file **/
	cout << "Production costs have been recorded as follows:" << endl;
	ifstream costsFile;
	costsFile.open("ProductionCostData.txt");
	// Material cost [$/unit]
	string MaterialCostName;
	readString(MaterialCostName, costsFile);
	float MaterialCost;
	readValue(MaterialCost, costsFile, "[$/unit]");
	// Holding cost [$/unit]
	string HoldingCostName;
	readString(HoldingCostName, costsFile);
	float HoldingCost;
	readValue(HoldingCost, costsFile, "[$/unit]");
	// Stockout cost [$/unit]
	string StockoutRateName;
	readString(StockoutRateName, costsFile);
	float StockoutRate;
	readValue(StockoutRate, costsFile, "[$/unit]");
	// Worker hiring and training cost [$/worker]
	string WorkerHiringCostName;
	readString(WorkerHiringCostName, costsFile);
	float WorkerHiringCost;
	readValue(WorkerHiringCost, costsFile, "[$/worker]");
	// Worker layoff cost [$/worker]
	string WorkerLayoffCostName;
	readString(WorkerLayoffCostName, costsFile);
	float WorkerLayoffCost;
	readValue(WorkerLayoffCost, costsFile, "[$/worker]");
	// Straight time [$/hr]
	string StraightTimeRateName;
	readString(StraightTimeRateName, costsFile);
	float StraightTimeRate;
	readValue(StraightTimeRate, costsFile, "[$/hr]");
	// Over time [$/hr]
	string OverTimeRateName;
	readString(OverTimeRateName, costsFile);
	float OverTimeRate;
	readValue(OverTimeRate, costsFile, "[$/hr]");
	// Down time [$/hr]
	string DownTimeRateName;
	readString(DownTimeRateName, costsFile);
	float DownTimeRate;
	readValue(DownTimeRate, costsFile, "[$/hr]");
	// Close cost input file
	costsFile.close();
	cout << endl;

	/** Reading and storing production requirement data from file **/
	cout << "Production requirements have been recorded as follows:" <<
endl;
	ifstream productionFile;
	productionFile.open("ProductionData.txt");
	// Production time [hrs/unit]
	string ProductionTimeName;
	readString(ProductionTimeName, productionFile);
	float ProductionTime;
	readValue(ProductionTime, productionFile, "[min/unit]");
	ProductionTime = ProductionTime / 60;
	// Convert from minutes to hours
	// Planning horizon [months]
	string PlanningHorizonName;
	productionFile >> PlanningHorizonName;
	cout << PlanningHorizonName << " ";
	int PlanningHorizon;
	productionFile >> PlanningHorizon;
	cout << PlanningHorizon << " [months]" << endl;

	/* Production data array initialisation based on planning horizon */
	float* WorkingDays = new float[PlanningHorizon];
	float* DemandForecast = new float[PlanningHorizon];
	float* SafetyStock = new float[PlanningHorizon];
	float* StartInventory = new float[PlanningHorizon];
	float* ProductionReqs = new float[PlanningHorizon];
	float* EndInventory = new float[PlanningHorizon];
	/*-----------------------------------------------------------------*/

	// Working days per month
	string WorkingDaysName;
	productionFile >> WorkingDaysName;
	cout << WorkingDaysName << " ";
	for (int i = 0; i < PlanningHorizon; i++) {
        productionFile >> WorkingDays[i];
        cout << WorkingDays[i] << "\t";
}
cout << endl;

// Calculate total working days
float TotalWorkingDays = 0;
for (int i = 0; i < PlanningHorizon; i++) {
        TotalWorkingDays += WorkingDays[i];
}

// Demand forecasts
string DemandForecastName;
productionFile >> DemandForecastName;
cout << DemandForecastName << "\t   ";
for (int i = 0; i < PlanningHorizon; i++) {
        productionFile >> DemandForecast[i];
        cout << DemandForecast[i] << "\t";
}
cout << endl;
// Calculate total demand for the planning horizon
float TotalDemand = 0;
for (int i = 0; i < PlanningHorizon; i++) {
        TotalDemand = TotalDemand + DemandForecast[i];
}

// Safety stock requirement
string SafetyStockName;
productionFile >> SafetyStockName;
cout << SafetyStockName << "\t ";
float SafetyStockPC;
productionFile >> SafetyStockPC;
cout << SafetyStockPC << "% of Demand" << endl;
// Initial inventory
string InitialInventoryName;
readString(InitialInventoryName, productionFile);
float InitialInventory;
productionFile >> InitialInventory;
cout << InitialInventory << " [units]" << endl;
// Starting workforce
string InitialWorkforceName;
readString(InitialWorkforceName, productionFile);
float InitialWorkforce;
readValue(InitialWorkforce, productionFile, " Workers");
cout << endl << endl;
// Close production inputs file
productionFile.close();

// Calculate monthly safety stock requirements
for (int i = 0; i < PlanningHorizon; i++) {
        SafetyStock[i] = round((SafetyStockPC / 100)*DemandForecast[i]);
}
```

```
	/* Calculate PRELIMINARY monthly production requirements and ending
inventories */
	for (int i = 0; i < PlanningHorizon; i++) {
		if (i == 0) {
			StartInventory[i] = InitialInventory;
		}
		else {
			StartInventory[i] = EndInventory[i - 1];
		}
		ProductionReqs[i] = DemandForecast[i] + SafetyStock[i] -
StartInventory[i];
		EndInventory[i] = StartInventory[i] + ProductionReqs[i] -
DemandForecast[i];
	}

	/* Print out table of production requirement data */
	cout << "PRELIMINARY PRODUCTION DATA ARE AS FOLLOWS:" << endl;
	// Starting inventories
	display("StartInventory:", PlanningHorizon, StartInventory);
	// Demand forecast
	display("DemandForecast:", PlanningHorizon, DemandForecast);
	// Safety stock
	display("SafetyStock:", PlanningHorizon, SafetyStock);
	// Production requirement
	display("ProductionReqs:", PlanningHorizon, ProductionReqs);
	// Ending inventories
	display("EndInventory:", PlanningHorizon, EndInventory);
	cout << endl;

	/* Defining remaining variables required for cost calculations in
dynamic memory */
	float* ProdHrsReq = new float[PlanningHorizon];			//
Production hours required = ProductionReqs * ProductionTime
	float* HrsPWorkerPM = new float[PlanningHorizon];		// Hours
per worker per month = WorkingDays * 8hrs/day
	float* Workforce = new float[PlanningHorizon];			//
Workers required = ProdHrsReq / HrsPWorkerPM
	float* WorkforceChange = new float[PlanningHorizon]; // Changes to
workforce level month on month
	float* NewHires = new float[PlanningHorizon];			// New
workers hired (From initial workforce to first month and onwards)
	float* HiringCost = new float[PlanningHorizon];			//
Hiring cost = NewHires * WorkerHiringCost
	float* Layoffs = new float[PlanningHorizon];			// Workers
laid off (From initial workforce to first month and onwards)
	float* LayoffCost = new float[PlanningHorizon];			//
Laying off cost = Layoffs * WorkerLayoffCost
	float* StraightTimeHrs = new float[PlanningHorizon]; // Straight time
worked
	float* StraightTimeCost = new float[PlanningHorizon];		// Straight
time cost = StraightTimeHrs * StraightTimeRate
```

```
	float* ProdHrsAvail = new float[PlanningHorizon];		//
Production hours available = WorkingDays * 8hrs/day * No. of workers
	float* ActualProd = new float[PlanningHorizon];			//
Actual production = ProdHrsAvail / ProductionTime
	float* UnitsShort = new float[PlanningHorizon];			//
Stock shortage after regular shift production
	float* ShortageCost = new float[PlanningHorizon];		// Shortage
cost = Units short(EndInventory) * StockoutRate (only if EndInventory is -ve,
else 0)
	float* UnitsExcess = new float[PlanningHorizon];		// Excess
stock produced = EndInventory - SafetyStock (only if +ve, else 0)
	float* InventoryCost = new float[PlanningHorizon];		//
Inventory cost = UnitsExcess * HoldingCost
	float* RegularShiftProd = new float[PlanningHorizon];	// Regular
shift production = ProdHrsAvail / ProductionTime
	float* UnitsAvail = new float[PlanningHorizon];			//
Units available after regular shift
	float* UnitsPreOvertime = new float[PlanningHorizon];	// Units
available before overtime = StartInventory + RegularShiftProd -
DemandForecast
	float* OvertimeProd = new float[PlanningHorizon];		// Units
produced during overtime = (-)UnitsPreOvertime (0 if UnitsPreOvertime is +ve)
	float* OvertimeHrs = new float[PlanningHorizon];		// Overtime
hours = OvertimeProd * ProductionTime
	float* OvertimeCost = new float[PlanningHorizon];		// Overtime
cost = OvertimeHrs * OvertimeRate
	float* DowntimeHrs = new float[PlanningHorizon];		// Downtime
hours
	float* DowntimeCost = new float[PlanningHorizon];		// Downtime
cost = DowntimeHrs * DowntimeRate
	float* MonthlyCost = new float[PlanningHorizon];		// Sum of
costs for each month
	float* MaxStockout = new float[PlanningHorizon];		// Maximum
allowable stockout per month
	float* MaxInventory = new float[PlanningHorizon];		// Maimum
allowable inventory per month
	float* MaxOvertime = new float[PlanningHorizon];		// Maximum
allowable overtime per month
	float* currentStartInv = new float[PlanningHorizon];	// Monthly
starting inventory of current plan
	float* currentProdReqs = new float[PlanningHorizon];	// Monthly
production requiremens for current plan
	float* currentActualProd = new float[PlanningHorizon];	// Monthly
production for current plan
	float* currentEndInv = new float[PlanningHorizon];		// Montly
ending inventory for current plan
	float* currentWorkforce = new float[PlanningHorizon];	// Monthly
workforce for current plan
	float* currentMonthlyCost = new float[PlanningHorizon];	// Monthly
cost for current plan
	float* BestStartInv = new float[PlanningHorizon];		// Monthly
starting inventory of optimal plan
```

```
	float* BestProdReqs = new float[PlanningHorizon];	// Monthly
production requiremens for optimal plan
	float* BestActualProd = new float[PlanningHorizon];	// Monthly
production for optimal plan
	float* BestEndInv = new float[PlanningHorizon];		//
Montly ending inventory for optimal plan
	float* BestWorkforce = new float[PlanningHorizon];	// Monthly
workforce for optimal plan
	float* BestMonthlyCost = new float[PlanningHorizon];// Monthly cost
for optimal plan
	float* InvtoProdRatio = new float[PlanningHorizon]; // Array of random
numbers

	// Defining additional variable required for cost calculations
	float PureLowestCost;			// Lowest plan cost out of the
pure planning strategies
	float WorkforceDiff = 0;	// Difference between current
workforce and proposed workforce
	float TotalCost = 0;			// Intialisation for total cost
calculation
	int PureStratFlag = 1;			// Pure strategy selection flag

	// Calculating the required workforce level for the constant workforce
with varying inventory and stockout strategy
	float AvgReqWorkforce = round((TotalDemand * ProductionTime) /
(TotalWorkingDays * 8 /*hrs/day*/));
	// Calculating the lowest required workforce level
	float LowestProdReq = *min_element(ProductionReqs, ProductionReqs +
PlanningHorizon);
	int minIndex = MinElementIndex(ProductionReqs, PlanningHorizon);
	float LowestReqWorkforce = round(((LowestProdReq*ProductionTime) /
(WorkingDays[minIndex] * 8 /*hrs/day*/)));
	// Calculating the highest required workforce level
	float HighestProdReq = *max_element(ProductionReqs, ProductionReqs +
PlanningHorizon);
	int maxIndex = MaxElementIndex(ProductionReqs, PlanningHorizon);
	float HighestReqWorkforce = round(((HighestProdReq*ProductionTime) /
(WorkingDays[maxIndex] * 8 /*hrs/day*/)));
	float InitialWorkforceCost = 0;		// Initialising the initial
workforce change cost variable

	// Pure Strategy #1: Exact production with varying workforce
	for (int i = 0; i < PlanningHorizon; i++) {
		// Run through the months up to the planning horizon
		ProdHrsReq[i] = ProductionReqs[i] * ProductionTime;
	// Calculate the required production hours
		HrsPWorkerPM[i] = WorkingDays[i] * 8 /*hrs/day*/;
	// Calculate the available hours per worker per month
		Workforce[i] = round(ProdHrsReq[i] / HrsPWorkerPM[i]);
	// Calculate the number of workers required
```

```
            if (i == 0) {
                        // Boundary conditions for monthly required
workforce calculation
                  WorkforceDiff = Workforce[i] - InitialWorkforce;
            }
            else {
                  WorkforceDiff = Workforce[i] - Workforce[i - 1];
            }
            if (WorkforceDiff == 0) {
                  // Workforce changes - hiring and laying off
                  NewHires[i] = 0;
                  Layoffs[i] = 0;
            }
            else if (WorkforceDiff < 0) {
                  NewHires[i] = 0;
                  Layoffs[i] = -WorkforceDiff;
            }
            else {
                  NewHires[i] = WorkforceDiff;
                  Layoffs[i] = 0;
            }
            StraightTimeCost[i] = ProdHrsReq[i] * StraightTimeRate;
                              // Cost calculations
            HiringCost[i] = NewHires[i] * WorkerHiringCost;
            LayoffCost[i] = Layoffs[i] * WorkerLayoffCost;
            MonthlyCost[i] = StraightTimeCost[i] + HiringCost[i] +
LayoffCost[i];
            // Sum total cost
            TotalCost = TotalCost + MonthlyCost[i];
      }
      cout << "Pure Strategy #1 cost: $" << TotalCost << endl;
      // Set current plan variables to best plan variables
      copyArray(StartInventory, BestStartInv, PlanningHorizon);
      copyArray(ProductionReqs, BestProdReqs, PlanningHorizon);
      copyArray(ActualProd, BestActualProd, PlanningHorizon);
      copyArray(EndInventory, BestEndInv, PlanningHorizon);
      copyArray(Workforce, BestWorkforce, PlanningHorizon);
      copyArray(MonthlyCost, BestMonthlyCost, PlanningHorizon);
      // Set current cost to best initial cost
      PureLowestCost = TotalCost;
      TotalCost = 0;

      // Pure Strategy #2: Constant workforce with varying work hours
      if (InitialWorkforce < AvgReqWorkforce) {
            InitialWorkforceCost = (AvgReqWorkforce -
InitialWorkforce)*WorkerHiringCost;
      }
      else if (InitialWorkforce > AvgReqWorkforce) {
            InitialWorkforceCost = (InitialWorkforce -
AvgReqWorkforce)*WorkerLayoffCost;
      }
      TotalCost = InitialWorkforceCost;
```

```
	for (int i = 0; i < PlanningHorizon; i++) {
		Workforce[i] = AvgReqWorkforce;
		ProdHrsAvail[i] = WorkingDays[i] * 8 /*hrs/day*/ *
AvgReqWorkforce;
		ProdHrsReq[i] = ProductionReqs[i] * ProductionTime;
		if (ProdHrsAvail[i] < ProdHrsReq[i]) {
			StraightTimeHrs[i] = ProdHrsAvail[i];
			OvertimeHrs[i] = ProdHrsReq[i] - ProdHrsAvail[i];
			DowntimeHrs[i] = 0;
		}
		else if (ProdHrsAvail[i] > ProdHrsReq[i]) {
			StraightTimeHrs[i] = ProdHrsAvail[i];
			OvertimeHrs[i] = 0;
			DowntimeHrs[i] = ProdHrsAvail[i] - ProdHrsReq[i];
		}
		else {
			StraightTimeHrs[i] = ProdHrsReq[i];
			OvertimeHrs[i] = 0;
			DowntimeHrs[i] = 0;
		}
		ActualProd[i] = (StraightTimeHrs[i] + OvertimeHrs[i]) /
ProductionTime;
		StraightTimeCost[i] = StraightTimeHrs[i] * StraightTimeRate;
		OvertimeCost[i] = OvertimeHrs[i] * OverTimeRate;
		DowntimeCost[i] = DowntimeHrs[i] * DownTimeRate;
		MonthlyCost[i] = StraightTimeCost[i] + OvertimeCost[i] +
DowntimeCost[i];
		TotalCost = TotalCost + MonthlyCost[i];
	}
	cout << "Pure strategy #2 cost: $" << TotalCost << endl;

	if (TotalCost < PureLowestCost) {
		// Set current plan variables to best plan variables
		copyArray(StartInventory, BestStartInv, PlanningHorizon);
		copyArray(ProductionReqs, BestProdReqs, PlanningHorizon);
		copyArray(ActualProd, BestActualProd, PlanningHorizon);
		copyArray(EndInventory, BestEndInv, PlanningHorizon);
		copyArray(Workforce, BestWorkforce, PlanningHorizon);
		copyArray(MonthlyCost, BestMonthlyCost, PlanningHorizon);
		// Set current cost to best initial cost
		PureLowestCost = TotalCost;
		PureStratFlag = 2;
	}
	TotalCost = 0;
	InitialWorkforceCost = 0;

	// Pure Strategy #3: Constant average workforce with varying inventory
and stockouts
	if (InitialWorkforce < AvgReqWorkforce) {
		InitialWorkforceCost = (AvgReqWorkforce -
InitialWorkforce)*WorkerHiringCost;
	}
	else if (InitialWorkforce > AvgReqWorkforce) {
		InitialWorkforceCost = (InitialWorkforce -
AvgReqWorkforce)*WorkerLayoffCost;
	}
	TotalCost = InitialWorkforceCost;
	for (int i = 0; i < PlanningHorizon; i++) {
		Workforce[i] = AvgReqWorkforce;
		if (i == 0)
			StartInventory[i] = InitialInventory;
		else
			StartInventory[i] = EndInventory[i - 1];
		ProdHrsAvail[i] = WorkingDays[i] * 8 /*hrs/day*/ *
AvgReqWorkforce;
		ActualProd[i] = round(ProdHrsAvail[i] / ProductionTime);
		EndInventory[i] = StartInventory[i] + ActualProd[i] -
DemandForecast[i];
		if (EndInventory[i] < 0)
			ShortageCost[i] = (-EndInventory[i])*StockoutRate;
		else
			ShortageCost[i] = 0;
		if ((EndInventory[i] - SafetyStock[i]) <= 0)
			UnitsExcess[i] = 0;
		else
			UnitsExcess[i] = EndInventory[i] - SafetyStock[i];
		InventoryCost[i] = UnitsExcess[i] * HoldingCost;
		StraightTimeCost[i] = ProdHrsAvail[i] * StraightTimeRate;
		MonthlyCost[i] = ShortageCost[i] + InventoryCost[i] +
StraightTimeCost[i];
		TotalCost = TotalCost + MonthlyCost[i];
	}
	cout << "Pure strategy #3 cost: $" << TotalCost << endl;

	if (TotalCost < PureLowestCost) {
		// Set current plan variables to best plan variables
		copyArray(StartInventory, BestStartInv, PlanningHorizon);
		copyArray(ProductionReqs, BestProdReqs, PlanningHorizon);
		copyArray(ActualProd, BestActualProd, PlanningHorizon);
		copyArray(EndInventory, BestEndInv, PlanningHorizon);
		copyArray(Workforce, BestWorkforce, PlanningHorizon);
		copyArray(MonthlyCost, BestMonthlyCost, PlanningHorizon);
		// Set current cost to best initial cost
		PureLowestCost = TotalCost;
		PureStratFlag = 3;
	}

	/* Definition of optimisation algorithm parameters and initial
solution */
	bool ConstWorkforce = 1;			// [0/1] Determines
whether the workforce will be held constant through the planning horizon
	float WorkforceParam = 0;			// [0-1] Determines the
degree to which the workforce will be adjusted for each iteration
```

```
	//float InvtoProdRatio = 0;				// [0-1]
Determines the affinity to inventory/stockout as compared to production

	cout << "\nThe pure strategy with the lowest cost is:" << endl;
	switch (PureStratFlag) {
	case 1:
		ConstWorkforce = 0;
		//InvtoProdRatio = 0;
		cout << "Exact production with varying workforce." << endl <<
endl;
		break;
	case 2:
		ConstWorkforce = 1;
		//InvtoProdRatio = 0;
		cout << "Constant average workforce with varying working hours,
including downtime and overtime." << endl << endl;
		break;
	case 3:
		ConstWorkforce = 1;
		//InvtoProdRatio = 1;
		cout << "Constant average workforce with varying inventory and
stockout." << endl << endl;
	default:
		break;
	}
	/* Print out table of production data */
	// Starting inventories per month
	display("StartInventory:", PlanningHorizon, BestStartInv);
	// Demand forecast per month
	display("DemandForecast:", PlanningHorizon, DemandForecast);
	// Safety stock per month
	display("SafetyStock:", PlanningHorizon, SafetyStock);
	// Production requirement per month
	display("ProductionReq:", PlanningHorizon, BestProdReqs);
	// Actual production per month
	display("ActualProd:", PlanningHorizon, BestActualProd);
	// Ending inventories per month
	display("EndInventory:", PlanningHorizon, BestEndInv);
	// Workforce per month
	display("Workforce:", PlanningHorizon, BestWorkforce);
	// Monthly costs
	display("Monthly Cost:", PlanningHorizon, BestMonthlyCost);
	cout << endl;
	cout << "The total cost of this strategy is : $" << PureLowestCost <<
endl << endl;

/********************************** END OF PURE STRATEGY COST CALCULATIONS
*************************************/

/********************************* START OF SIMULATED ANNEALING OPTIMISATION
***********************************/
	/*	Algorithm:
```

```
             1. Make lowest cost pure plan best solution
             2. While temperature > min_temperature do:
             2.1 Make random change to strategy based on temperature
             2.2 Check that the new plan is valid, otherwise redo step 2.1
             2.3 Calculate cost of new plan
             Compare new cost with current best cost
             if new cost < current best cost
             set best cost to new cost
             if new cost >= best cost
             set best cost to new cost only if acceptance condition is met
(eg. random number < acceptance probability based on temperature)
             else discard new plan
             2.4 Repeat Steps 2.1 to 2.3 for X iterations
             2.5 Change temperature according to change schedule and repeat
2.1 to 2.3 with new temperature
             */

      // Reset plan variables ahead of optimisation algorithm
      TotalCost = 0;
      InitialWorkforceCost = 0;

      float currentCost = PureLowestCost;                 // Set current
best cost to lowest pure strategy plan cost with strategy parameters set in
switch statement above
      int maxIteration = 5000;                            // Maximum
iterations per temperature setting
      float maxTemp = 800000;                                    // Maximum
temperature of simulated annealing algorithm
      float newCost = 500000;                                    // Cost of
new plan based on new strategy initialised to $500,000.00
      float temp = maxTemp;                                      //
Temperature for simulated annealing algorithm initialised to maximum
temperature
      float minTemp = 500000;                                    // Minimum
temperature of simulated annealing algorithm
      float tempSched = 0.99;                                    //
Temperature change schedule of simulated annealing algorithm
      float NewWorkforce = AvgReqWorkforce;         // Workforce for new
plan initialised to average workforce required for total production over
planning horizon
      float BestCost = currentCost;                       // Cost of overall
best plan initialised to cost of current plan
      float AccProb = 0;                                         //
Probability of simulated annealing algorithm accepting a worse plan than the
current best plan
      float random = 0;                                          // Random
number regenerated at each iteration for comaprison with AccProb

      ofstream resultsFile;                                      // Link to
a results file to be written to at each temperature in the simulated
annealing algorithm
      resultsFile.open("SAresults.txt");
```

```
	// Temperature loop of simulated annealing algorithm starts here
	while (temp > minTemp) {
		// Iteration loop of simulated annealing algorithm starts here
		for (int iteration = 0; iteration < maxIteration; iteration++) {
			// Reset new plan variables
			TotalCost = 0;
			newCost = 0;
			for (int i = 0; i < PlanningHorizon; i++) {
				InvtoProdRatio[i] = ((float)(rand() % 50 - 25)) /
150;
			}
		/* CONSTANT WORKFORCE */
			if (ConstWorkforce) {
				// Calculate the cost of initial workforce changes
				if (InitialWorkforce < NewWorkforce) {
					InitialWorkforceCost = (NewWorkforce -
InitialWorkforce)*WorkerHiringCost;
				}
				else if (InitialWorkforce > NewWorkforce) {
					InitialWorkforceCost = (InitialWorkforce -
NewWorkforce)*WorkerLayoffCost;
				}
				// Run through the months up to the planning
horizon
				for (int i = 0; i < PlanningHorizon; i++) {
					// Workforce remains constant
					Workforce[i] = NewWorkforce;
					ProdHrsAvail[i] = WorkingDays[i] * 8
/*hrs/day*/ * Workforce[i];
					// Starting inventory calculated from
previous month's ending inventory
					if (i == 0) {
						StartInventory[i] = InitialInventory;
						MonthlyCost[i] = InitialWorkforceCost;
					}
					else {
						StartInventory[i] = EndInventory[i -
1];

						MonthlyCost[i] = 0;
					}
					// Production calculations
					// Number of production hours required,
based on the required production for the month
					ProductionReqs[i] = (DemandForecast[i] +
SafetyStock[i] - StartInventory[i]);
					ProdHrsReq[i] = round((1-
InvtoProdRatio[i])*(ProductionReqs[i] * ProductionTime));
					// Production hours calculations
					if (ProdHrsReq[i] < ProdHrsAvail[i]) {
							StraightTimeHrs[i] =
ProdHrsReq[i];
```

```
                                        OvertimeHrs[i] = 0;
                                        DowntimeHrs[i] =
(ProdHrsAvail[i] - ProdHrsReq[i]);
                                }
                                else if (ProdHrsReq[i] > ProdHrsAvail[i]) {
                                        StraightTimeHrs[i] = ProdHrsAvail[i];
                                        OvertimeHrs[i] = ProdHrsReq[i] -
ProdHrsAvail[i];
                                        DowntimeHrs[i] = 0;
                                }
                                else {
                                        StraightTimeHrs[i] = ProdHrsAvail[i];
                                        OvertimeHrs[i] = 0;
                                        DowntimeHrs[i] = 0;
                                }
                                RegularShiftProd[i] =
round(StraightTimeHrs[i] / ProductionTime);
                                OvertimeProd[i] = round(OvertimeHrs[i] /
ProductionTime);
                                ActualProd[i] = RegularShiftProd[i] +
OvertimeProd[i];
                                EndInventory[i] = StartInventory[i] +
ActualProd[i] - DemandForecast[i];
                                // Inventory/Stockout cost calculations
                                if (EndInventory[i] < 0)
                                        ShortageCost[i] =
abs(EndInventory[i])*StockoutRate;
                                else
                                        ShortageCost[i] = 0;
                                if ((EndInventory[i] - SafetyStock[i]) <= 0)
                                        UnitsExcess[i] = 0;
                                else
                                        UnitsExcess[i] = EndInventory[i] -
SafetyStock[i];
                                InventoryCost[i] = UnitsExcess[i] *
HoldingCost;
                                // Production costs per month
                                StraightTimeCost[i] = StraightTimeHrs[i] *
StraightTimeRate;
                                OvertimeCost[i] = OvertimeHrs[i] *
OverTimeRate;
                                DowntimeCost[i] = DowntimeHrs[i] *
DownTimeRate;
                                // Summation of costs per month
                                MonthlyCost[i] += StraightTimeCost[i] +
OvertimeCost[i] + DowntimeCost[i] + ShortageCost[i] + InventoryCost[i];
                                // Summation of monthly costs
                                TotalCost += MonthlyCost[i];
                        }
                }
        /* NON-CONSTANT WORKFORCE */
                else {
                        // Run through the months up to the planning
horizon
                        for (int i = 0; i < PlanningHorizon; i++) {
                              // Starting inventory boundary condition
                              if (i == 0)
                                    StartInventory[i] = InitialInventory;
                              else
                                    StartInventory[i] = EndInventory[i -
1];
                              // Calculation of production hours required
                              ProductionReqs[i] = DemandForecast[i] +
SafetyStock[i] - StartInventory[i];
                              ProdHrsReq[i] = round((1 -
InvtoProdRatio[i])*(ProductionReqs[i] * ProductionTime));  // Calculate the
required production hours
                              HrsPWorkerPM[i] = WorkingDays[i] * 8
/*hrs/day*/;                                                         //
Calculate the available hours per worker per month
                              Workforce[i] = round(ProdHrsReq[i] /
HrsPWorkerPM[i]) + WorkforceParam;                             //
Calculate the number of workers required
                              ProdHrsAvail[i] = HrsPWorkerPM[i] *
Workforce[i];
                              ActualProd[i] = round(ProdHrsAvail[i] /
ProductionTime);
                              if (i == 0) {// Boundary conditions for
monthly required workforce calculation
                                    WorkforceDiff = Workforce[i] -
InitialWorkforce;
                              }
                              else {
                                    WorkforceDiff = Workforce[i] -
Workforce[i - 1];
                              }
                              if (WorkforceDiff == 0) { // Workforce
changes - hiring and laying off
                                    NewHires[i] = 0;
                                    Layoffs[i] = 0;
                              }
                              else if (WorkforceDiff < 0) {
                                    NewHires[i] = 0;
                                    Layoffs[i] = -WorkforceDiff;
                              }
                              else {
                                    NewHires[i] = WorkforceDiff;
                                    Layoffs[i] = 0;
                              }
                              EndInventory[i] = StartInventory[i] +
ActualProd[i] - DemandForecast[i];
                              if (EndInventory[i] < 0)
                                    ShortageCost[i] = (-
EndInventory[i])*StockoutRate;
```

```
                                else
                                      ShortageCost[i] = 0;
                                if ((EndInventory[i] - SafetyStock[i]) < 0)
                                      UnitsExcess[i] = 0;
                                else
                                      UnitsExcess[i] = EndInventory[i] -
SafetyStock[i];
                                InventoryCost[i] = UnitsExcess[i] *
HoldingCost;
                                StraightTimeCost[i] = ProdHrsAvail[i] *
StraightTimeRate;                                   // Cost calculations
                                HiringCost[i] = NewHires[i] *
WorkerHiringCost;
                                LayoffCost[i] = Layoffs[i] *
WorkerLayoffCost;
                                MonthlyCost[i] = StraightTimeCost[i] +
HiringCost[i] + LayoffCost[i] + ShortageCost[i] + InventoryCost[i];
                                // Sum total cost
                                TotalCost += MonthlyCost[i];
                          }
                    }
                    newCost = TotalCost;

                    // Generate new plan startegy parameters
                    WorkforceParam = round(((rand() % 2) * 2 -
1)*((float)(rand() % 5)));
                    NewWorkforce = NewWorkforce + WorkforceParam;

                    // Set new plan variable to current plan variables if new
plan cost is less than or equal to current plan cost
                    if (EndInventory[PlanningHorizon - 1] > 0) {
                          if (newCost <= currentCost) {
                                currentCost = newCost;
                                copyArray(StartInventory, currentStartInv,
PlanningHorizon);
                                copyArray(ProductionReqs, currentProdReqs,
PlanningHorizon);
                                copyArray(ActualProd, currentActualProd,
PlanningHorizon);
                                copyArray(EndInventory, currentEndInv,
PlanningHorizon);
                                copyArray(Workforce, currentWorkforce,
PlanningHorizon);
                                copyArray(MonthlyCost, currentMonthlyCost,
PlanningHorizon);
                                // Set current plan variables to best plan
variables if current plan cost is less than or equal to best plan cost
                                if (currentCost <= BestCost) {
                                      BestCost = currentCost;
                                      copyArray(currentStartInv,
BestStartInv, PlanningHorizon);
```

```
                                        copyArray(currentProdReqs,
BestProdReqs, PlanningHorizon);
                                        copyArray(currentActualProd,
BestActualProd, PlanningHorizon);
                                        copyArray(currentEndInv, BestEndInv,
PlanningHorizon);
                                        copyArray(currentWorkforce,
BestWorkforce, PlanningHorizon);
                                        copyArray(currentMonthlyCost,
BestMonthlyCost, PlanningHorizon);
                                  }
                          }
                          // Set new plan variable to current plan variables
if new plan cost is more than current plan cost and
                          // acceptance probability is smaller than a
randomly generated number
                          else if (newCost > currentCost) {
                                  AccProb = pow(AcceptProb(newCost,
currentCost, temp), 1);
                                  random = (static_cast <float> (rand()) /
static_cast <float> (RAND_MAX));
                                  if (AccProb > random) {
                                        currentCost = newCost;
                                        copyArray(StartInventory,
currentStartInv, PlanningHorizon);
                                        copyArray(ProductionReqs,
currentProdReqs, PlanningHorizon);
                                        copyArray(ActualProd,
currentActualProd, PlanningHorizon);
                                        copyArray(EndInventory, currentEndInv,
PlanningHorizon);
                                        copyArray(Workforce, currentWorkforce,
PlanningHorizon);
                                        copyArray(MonthlyCost,
currentMonthlyCost, PlanningHorizon);
                                        if (ConstWorkforce)
                                              ConstWorkforce = 0;
                                        else
                                              ConstWorkforce = 1;
                                  }
                                  else {
                                        if (ConstWorkforce)
                                              ConstWorkforce = 1;
                                        else
                                              ConstWorkforce = 0;
                                  }
                          }
                    }
                    else
                          continue;

                    if (iteration % 50 == 0)
```

```
                        resultsFile << ConstWorkforce << "\t" << newCost <<
"\t" << currentCost << "\t" << BestCost << "\t" << temp << endl;

                }/************ END OF INNER LOOP ************/

                // Set next step in temperture schedule
                temp = tempSched*temp;
                cout << ConstWorkforce << '\t' << currentCost << '\t' <<
BestCost << "\t" << temp << endl;

        }/*********** END OF OUTER LOOP ***********/

        resultsFile.close();
        cout << endl;
        /* Print out table of production data */
        // Starting inventories per month
        display("StartInventory:", PlanningHorizon, currentStartInv);
        // Demand forecast per month
        display("DemandForecast:", PlanningHorizon, DemandForecast);
        // Safety stock per month
        display("SafetyStock:", PlanningHorizon, SafetyStock);
        // Production requirement per month
        display("ProductionReq:", PlanningHorizon, currentProdReqs);
        // Actual production per month
        display("ActualProd:", PlanningHorizon, currentActualProd);
        // Ending inventories per month
        display("EndInventory:", PlanningHorizon, currentEndInv);
        // Workforce per month
        display("Workforce:", PlanningHorizon, currentWorkforce);
        // Monthly costs
        display("Monthly Cost:", PlanningHorizon, currentMonthlyCost);

        cout << endl;
        /* Print out table of production data */
        cout << "The most optimal plan is configured as follows:\n" << endl;
        // Starting inventories per month
        display("StartInventory:", PlanningHorizon, BestStartInv);
        // Demand forecast per month
        display("DemandForecast:", PlanningHorizon, DemandForecast);
        // Safety stock per month
        display("SafetyStock:", PlanningHorizon, SafetyStock);
        // Production requirement per month
        display("ProductionReq:", PlanningHorizon, BestProdReqs);
        // Actual production per month
        display("ActualProd:", PlanningHorizon, BestActualProd);
        // Ending inventories per month
        display("EndInventory:", PlanningHorizon, BestEndInv);
        // Workforce per month
        display("Workforce:", PlanningHorizon, BestWorkforce);
        // Monthly costs
        display("Monthly Cost:", PlanningHorizon, BestMonthlyCost);
        cout << "\nThe total cost of this plan is: $" << BestCost << endl;
```

```
	cout << "\nThat is a saving of " << (((PureLowestCost - BestCost) /
PureLowestCost) * 100) << "%" << " of $" << PureLowestCost << endl;

/* Delete arrays dynamically allocated in memory */
	delete[] WorkingDays, DemandForecast, SafetyStock, StartInventory,
ProductionReqs, EndInventory;
	delete[] ProdHrsReq, HrsPWorkerPM, Workforce, WorkforceChange,
NewHires, HiringCost, Layoffs, LayoffCost, StraightTimeHrs;
	delete[] StraightTimeCost, ProdHrsAvail, ActualProd, UnitsShort,
ShortageCost, UnitsExcess, InventoryCost;
	delete[] RegularShiftProd, UnitsAvail, UnitsPreOvertime, OvertimeProd,
OvertimeCost, DowntimeHrs, DowntimeCost, MonthlyCost;
	delete[] MaxStockout, MaxInventory, MaxOvertime, currentActualProd,
currentEndInv, currentMonthlyCost, currentProdReqs;
	delete[] currentStartInv, currentWorkforce, BestActualProd,
BestEndInv, BestMonthlyCost, BestProdReqs, BestStartInv, BestWorkforce;

/** End matter of main programme **/
	cout << endl << endl;
	std::system("PAUSE");
	return 0;
}
```

D.2. Biogeography-Based Optimisation algorithm code

```
/*
Main programme source file
Author:             Louw Butler
Created:      09/03/2015
Modified:     03/06/2015
*/

/* User-defined header (includes all standard library headers) */
#include "sources.h"
#include <array>

using namespace std;
using std::cout;

/* MAIN PROGRAMME */
int main() {
      /** Reading and storing production cost input variables from file **/
      cout << "Production costs have been recorded as follows:" << endl;
      ifstream costsFile;
      costsFile.open("ProductionCostData.txt");
      // Material cost [$/unit]
      string MaterialCostName;
      readString(MaterialCostName, costsFile);
      float MaterialCost;
      readValue(MaterialCost, costsFile, "[$/unit]");
      // Holding cost [$/unit]
      string HoldingCostName;
      readString(HoldingCostName, costsFile);
      float HoldingCost;
      readValue(HoldingCost, costsFile, "[$/unit]");
      // Stockout cost [$/unit]
      string StockoutRateName;
      readString(StockoutRateName, costsFile);
      float StockoutRate;
      readValue(StockoutRate, costsFile, "[$/unit]");
      // Worker hiring and training cost [$/worker]
      string WorkerHiringCostName;
      readString(WorkerHiringCostName, costsFile);
      float WorkerHiringCost;
      readValue(WorkerHiringCost, costsFile, "[$/worker]");
      // Worker layoff cost [$/worker]
      string WorkerLayoffCostName;
      readString(WorkerLayoffCostName, costsFile);
      float WorkerLayoffCost;
      readValue(WorkerLayoffCost, costsFile, "[$/worker]");
      // Straight time [$/hr]
      string StraightTimeRateName;
      readString(StraightTimeRateName, costsFile);
      float StraightTimeRate;
      readValue(StraightTimeRate, costsFile, "[$/hr]");
```

```
	// Over time [$/hr]
	string OverTimeRateName;
	readString(OverTimeRateName, costsFile);
	float OverTimeRate;
	readValue(OverTimeRate, costsFile, "[$/hr]");
	// Down time [$/hr]
	string DownTimeRateName;
	readString(DownTimeRateName, costsFile);
	float DownTimeRate;
	readValue(DownTimeRate, costsFile, "[$/hr]");
	// Close cost input file
	costsFile.close();
	cout << endl;

	/** Reading and storing production requirement data from file **/
	cout << "Production requirements have been recorded as follows:" <<
endl;
	ifstream productionFile;
	productionFile.open("ProductionData.txt");
	// Production time [hrs/unit]
	string ProductionTimeName;
	readString(ProductionTimeName, productionFile);
	float ProductionTime;
	readValue(ProductionTime, productionFile, "[min/unit]");
	ProductionTime = ProductionTime / 60;
	// Convert from minutes to hours
	// Planning horizon [months]
	string PlanningHorizonName;
	productionFile >> PlanningHorizonName;
	cout << PlanningHorizonName << " ";
	int PlanningHorizon;
	productionFile >> PlanningHorizon;
	cout << PlanningHorizon << " [months]" << endl;

	/* Production data array initialisation based on planning horizon */
	float* WorkingDays = new float[PlanningHorizon];
	float* DemandForecast = new float[PlanningHorizon];
	float* SafetyStock = new float[PlanningHorizon];
	float* StartInventory = new float[PlanningHorizon];
	float* ProductionReqs = new float[PlanningHorizon];
	float* EndInventory = new float[PlanningHorizon];
	/*---------------------------------------------------------------*/

	// Working days per month
	string WorkingDaysName;
	productionFile >> WorkingDaysName;
	cout << WorkingDaysName << " ";
	for (int i = 0; i < PlanningHorizon; i++) {
		productionFile >> WorkingDays[i];
		cout << WorkingDays[i] << "\t";
	}
	cout << endl;
	// Calculate total working days
	float TotalWorkingDays = 0;
	for (int i = 0; i < PlanningHorizon; i++) {
		TotalWorkingDays += WorkingDays[i];
	}

	// Demand forecasts
	string DemandForecastName;
	productionFile >> DemandForecastName;
	cout << DemandForecastName << "\t   ";
	for (int i = 0; i < PlanningHorizon; i++) {
		productionFile >> DemandForecast[i];
		cout << DemandForecast[i] << "\t";
	}
	cout << endl;
	// Calculate total demand for the planning horizon
	float TotalDemand = 0;
	for (int i = 0; i < PlanningHorizon; i++) {
		TotalDemand = TotalDemand + DemandForecast[i];
	}

	// Safety stock requirement
	string SafetyStockName;
	productionFile >> SafetyStockName;
	cout << SafetyStockName << "\t ";
	float SafetyStockPC;
	productionFile >> SafetyStockPC;
	cout << SafetyStockPC << "% of Demand" << endl;
	// Initial inventory
	string InitialInventoryName;
	readString(InitialInventoryName, productionFile);
	float InitialInventory;
	productionFile >> InitialInventory;
	cout << InitialInventory << " [units]" << endl;
	// Starting workforce
	string InitialWorkforceName;
	readString(InitialWorkforceName, productionFile);
	float InitialWorkforce;
	readValue(InitialWorkforce, productionFile, " Workers");
	cout << endl << endl;
	// Close production inputs file
	productionFile.close();

	// Calculate monthly safety stock requirements
	for (int i = 0; i < PlanningHorizon; i++) {
		SafetyStock[i] = round((SafetyStockPC / 100)*DemandForecast[i]);
	}

	/* Calculate PRELIMINARY monthly production requirements and ending
inventories */
	for (int i = 0; i < PlanningHorizon; i++) {
		if (i == 0) {
			StartInventory[i] = InitialInventory;
		}
		else {
			StartInventory[i] = EndInventory[i - 1];
		}
		ProductionReqs[i] = DemandForecast[i] + SafetyStock[i] -
StartInventory[i];
		EndInventory[i] = StartInventory[i] + ProductionReqs[i] -
DemandForecast[i];
	}

	/* Print out table of production requirement data */
	cout << "PRELIMINARY PRODUCTION DATA ARE AS FOLLOWS:" << endl;
	// Starting inventories
	displayArray("StartInventory:", PlanningHorizon, StartInventory);
	// Demand forecast
	displayArray("DemandForecast:", PlanningHorizon, DemandForecast);
	// Safety stock
	displayArray("SafetyStock:", PlanningHorizon, SafetyStock);
	// Production requirement
	displayArray("ProductionReqs:", PlanningHorizon, ProductionReqs);
	// Ending inventories
	displayArray("EndInventory:", PlanningHorizon, EndInventory);
	cout << endl;

	/* Define remaining variables required for cost calculations in
dynamic memory */
	float* ProdHrsReq = new float[PlanningHorizon];			//
Production hours required = ProductionReqs * ProductionTime
	float* HrsPWorkerPM = new float[PlanningHorizon];		// Hours
per worker per month = WorkingDays * 8hrs/day
	float* Workforce = new float[PlanningHorizon];			//
Workers required = ProdHrsReq / HrsPWorkerPM
	float* WorkforceChange = new float[PlanningHorizon];	// Changes to
workforce level month on month
	float* NewHires = new float[PlanningHorizon];			// New
workers hired (From initial workforce to first month and onwards)
	float* HiringCost = new float[PlanningHorizon];			//
Hiring cost = NewHires * WorkerHiringCost
	float* Layoffs = new float[PlanningHorizon];			// Workers
laid off (From initial workforce to first month and onwards)
	float* LayoffCost = new float[PlanningHorizon];			//
Laying off cost = Layoffs * WorkerLayoffCost
	float* StraightTimeHrs = new float[PlanningHorizon];	// Straight time
worked
	float* StraightTimeCost = new float[PlanningHorizon];	// Straight
time cost = StraightTimeHrs * StraightTimeRate
	float* ProdHrsAvail = new float[PlanningHorizon];		//
Production hours available = WorkingDays * 8hrs/day * No. of workers
	float* ActualProd = new float[PlanningHorizon];			//
Actual production = ProdHrsAvail / ProductionTime
```

```
	float* UnitsShort = new float[PlanningHorizon];			//
Stock shortage after regular shift production
	float* ShortageCost = new float[PlanningHorizon];		// Shortage
cost = Units short(EndInventory) * StockoutRate (only if EndInventory is -ve,
else 0)
	float* UnitsExcess = new float[PlanningHorizon];		// Excess
stock produced = EndInventory - SafetyStock (only if +ve, else 0)
	float* InventoryCost = new float[PlanningHorizon];		//
Inventory cost = UnitsExcess * HoldingCost
	float* RegularShiftProd = new float[PlanningHorizon];	// Regular
shift production = ProdHrsAvail / ProductionTime
	float* UnitsAvail = new float[PlanningHorizon];			//
Units available after regular shift
	float* UnitsPreOvertime = new float[PlanningHorizon];	// Units
available before overtime = StartInventory + RegularShiftProd -
DemandForecast
	float* OvertimeProd = new float[PlanningHorizon];		// Units
produced during overtime = (-)UnitsPreOvertime (0 if UnitsPreOvertime is +ve)
	float* OvertimeHrs = new float[PlanningHorizon];		// Overtime
hours = OvertimeProd * ProductionTime
	float* OvertimeCost = new float[PlanningHorizon];		// Overtime
cost = OvertimeHrs * OvertimeRate
	float* DowntimeHrs = new float[PlanningHorizon];		// Downtime
hours
	float* DowntimeCost = new float[PlanningHorizon];		// Downtime
cost = DowntimeHrs * DowntimeRate
	float* MonthlyCost = new float[PlanningHorizon];		// Sum of
costs for each month
	float* MaxStockout = new float[PlanningHorizon];		// Maximum
allowable stockout per month
	float* MaxInventory = new float[PlanningHorizon];		// Maimum
allowable inventory per month
	float* MaxOvertime = new float[PlanningHorizon];		// Maximum
allowable overtime per month
	float* currentStartInv = new float[PlanningHorizon]; // Monthly
starting inventory of current plan
	float* currentProdReqs = new float[PlanningHorizon]; // Monthly
production requiremens for current plan
	float* currentActualProd = new float[PlanningHorizon];	// Monthly
production for current plan
	float* currentEndInv = new float[PlanningHorizon];		// Montly
ending inventory for current plan
	float* currentWorkforce = new float[PlanningHorizon];	// Monthly
workforce for current plan
	float* currentMonthlyCost = new float[PlanningHorizon];	// Monthly
cost for current plan
	float* BestStartInv = new float[PlanningHorizon];		// Monthly
starting inventory of optimal plan
	float* BestProdReqs = new float[PlanningHorizon];		// Monthly
production requiremens for optimal plan
	float* BestActualProd = new float[PlanningHorizon];		// Monthly
production for optimal plan
```

```
	float* BestEndInv = new float[PlanningHorizon];			//
Montly ending inventory for optimal plan
	float* BestWorkforce = new float[PlanningHorizon];		// Monthly
workforce for optimal plan
	float* BestMonthlyCost = new float[PlanningHorizon];// Monthly cost
for optimal plan
	float* InvtoProdRatio = new float[PlanningHorizon];		// Array of
random numbers

	// Define additional variables required for cost calculations
	float PureLowestCost;			// Lowest plan cost out of the
pure planning strategies
	float WorkforceDiff = 0;		// Difference between current
workforce and proposed workforce
	float TotalCost = 0;			// Intialisation for total cost
calculation
	int PureStratFlag = 1;			// Pure strategy selection flag

	// Calculate the required workforce level for the constant workforce
with varying inventory and stockout strategy
	float AvgReqWorkforce = round((TotalDemand * ProductionTime) /
(TotalWorkingDays * 8 /*hrs/day*/));
	// Calculating the lowest required workforce level
	float LowestProdReq = *min_element(ProductionReqs, ProductionReqs +
PlanningHorizon);
	int minIndex = MinElementIndex(ProductionReqs, PlanningHorizon);
	float LowestReqWorkforce = round(((LowestProdReq*ProductionTime) /
(WorkingDays[minIndex] * 8 /*hrs/day*/)));
	// Calculating the highest required workforce level
	float HighestProdReq = *max_element(ProductionReqs, ProductionReqs +
PlanningHorizon);
	int maxIndex = MaxElementIndex(ProductionReqs, PlanningHorizon);
	float HighestReqWorkforce = round(((HighestProdReq*ProductionTime) /
(WorkingDays[maxIndex] * 8 /*hrs/day*/)));
	float InitialWorkforceCost = 0;		// Initialising the initial
workforce change cost variable

	// Pure Strategy #1: Exact production with varying workforce
	for (int i = 0; i < PlanningHorizon; i++) {
		// Run through the months up to the planning horizon
		ProdHrsReq[i] = ProductionReqs[i] * ProductionTime;
	// Calculate the required production hours
		HrsPWorkerPM[i] = WorkingDays[i] * 8 /*hrs/day*/;
	// Calculate the available hours per worker per month
		Workforce[i] = round(ProdHrsReq[i] / HrsPWorkerPM[i]);
	// Calculate the number of workers required
		if (i == 0) {
				// Boundary conditions for monthly required
workforce calculation
			WorkforceDiff = Workforce[i] - InitialWorkforce;
		}
		else {
```

```
                WorkforceDiff = Workforce[i] - Workforce[i - 1];
            }
            if (WorkforceDiff == 0) {
                // Workforce changes - hiring and laying off
                NewHires[i] = 0;
                Layoffs[i] = 0;
            }
            else if (WorkforceDiff < 0) {
                NewHires[i] = 0;
                Layoffs[i] = -WorkforceDiff;
            }
            else {
                NewHires[i] = WorkforceDiff;
                Layoffs[i] = 0;
            }
            StraightTimeCost[i] = ProdHrsReq[i] * StraightTimeRate;
                                // Cost calculations
            HiringCost[i] = NewHires[i] * WorkerHiringCost;
            LayoffCost[i] = Layoffs[i] * WorkerLayoffCost;
            MonthlyCost[i] = StraightTimeCost[i] + HiringCost[i] +
LayoffCost[i];
            // Sum total cost
            TotalCost = TotalCost + MonthlyCost[i];
        }
        cout << "Pure Strategy #1 cost: $" << TotalCost << endl;
        // Set current plan variables to best plan variables
        copyArray(StartInventory, BestStartInv, PlanningHorizon);
        copyArray(ProductionReqs, BestProdReqs, PlanningHorizon);
        copyArray(ActualProd, BestActualProd, PlanningHorizon);
        copyArray(EndInventory, BestEndInv, PlanningHorizon);
        copyArray(Workforce, BestWorkforce, PlanningHorizon);
        copyArray(MonthlyCost, BestMonthlyCost, PlanningHorizon);
        // Set current cost to best initial cost
        PureLowestCost = TotalCost;
        TotalCost = 0;

        // Pure Strategy #2: Constant workforce with varying work hours
        if (InitialWorkforce < AvgReqWorkforce) {
            InitialWorkforceCost = (AvgReqWorkforce -
InitialWorkforce)*WorkerHiringCost;
        }
        else if (InitialWorkforce > AvgReqWorkforce) {
            InitialWorkforceCost = (InitialWorkforce -
AvgReqWorkforce)*WorkerLayoffCost;
        }
        TotalCost = InitialWorkforceCost;
        for (int i = 0; i < PlanningHorizon; i++) {
            Workforce[i] = AvgReqWorkforce;
            ProdHrsAvail[i] = WorkingDays[i] * 8 /*hrs/day*/ *
AvgReqWorkforce;
            ProdHrsReq[i] = ProductionReqs[i] * ProductionTime;
            if (ProdHrsAvail[i] < ProdHrsReq[i]) {
```

```
				StraightTimeHrs[i] = ProdHrsAvail[i];
				OvertimeHrs[i] = ProdHrsReq[i] - ProdHrsAvail[i];
				DowntimeHrs[i] = 0;
			}
			else if (ProdHrsAvail[i] > ProdHrsReq[i]) {
				StraightTimeHrs[i] = ProdHrsReq[i];
				OvertimeHrs[i] = 0;
				DowntimeHrs[i] = ProdHrsAvail[i] - ProdHrsReq[i];
			}
			else {
				StraightTimeHrs[i] = ProdHrsReq[i];
				OvertimeHrs[i] = 0;
				DowntimeHrs[i] = 0;
			}
			ActualProd[i] = (StraightTimeHrs[i] + OvertimeHrs[i]) / 
ProductionTime;
			StraightTimeCost[i] = StraightTimeHrs[i] * StraightTimeRate;
			OvertimeCost[i] = OvertimeHrs[i] * OverTimeRate;
			DowntimeCost[i] = DowntimeHrs[i] * DownTimeRate;
			MonthlyCost[i] = StraightTimeCost[i] + OvertimeCost[i] + 
DowntimeCost[i];
			TotalCost = TotalCost + MonthlyCost[i];
		}
		cout << "Pure strategy #2 cost: $" << TotalCost << endl;

		if (TotalCost < PureLowestCost) {
			// Set current plan variables to best plan variables
			copyArray(StartInventory, BestStartInv, PlanningHorizon);
			copyArray(ProductionReqs, BestProdReqs, PlanningHorizon);
			copyArray(ActualProd, BestActualProd, PlanningHorizon);
			copyArray(EndInventory, BestEndInv, PlanningHorizon);
			copyArray(Workforce, BestWorkforce, PlanningHorizon);
			copyArray(MonthlyCost, BestMonthlyCost, PlanningHorizon);
			// Set current cost to best initial cost
			PureLowestCost = TotalCost;
			PureStratFlag = 2;
		}
		TotalCost = 0;
		InitialWorkforceCost = 0;

		// Pure Strategy #3: Constant average workforce with varying inventory 
and stockouts
		if (InitialWorkforce < AvgReqWorkforce) {
			InitialWorkforceCost = (AvgReqWorkforce - 
InitialWorkforce)*WorkerHiringCost;
		}
		else if (InitialWorkforce > AvgReqWorkforce) {
			InitialWorkforceCost = (InitialWorkforce - 
AvgReqWorkforce)*WorkerLayoffCost;
		}
		TotalCost = InitialWorkforceCost;
		for (int i = 0; i < PlanningHorizon; i++) {
			Workforce[i] = AvgReqWorkforce;
			if (i == 0)
				StartInventory[i] = InitialInventory;
			else
				StartInventory[i] = EndInventory[i - 1];
			ProdHrsAvail[i] = WorkingDays[i] * 8 /*hrs/day*/ *
AvgReqWorkforce;
			ActualProd[i] = round(ProdHrsAvail[i] / ProductionTime);
			EndInventory[i] = StartInventory[i] + ActualProd[i] -
DemandForecast[i];
			if (EndInventory[i] < 0)
				ShortageCost[i] = abs(EndInventory[i])*StockoutRate;
			else
				ShortageCost[i] = 0;
			if ((EndInventory[i] - SafetyStock[i]) <= 0)
				UnitsExcess[i] = 0;
			else
				UnitsExcess[i] = EndInventory[i] - SafetyStock[i];
			InventoryCost[i] = UnitsExcess[i] * HoldingCost;
			StraightTimeCost[i] = ProdHrsAvail[i] * StraightTimeRate;
			MonthlyCost[i] = ShortageCost[i] + InventoryCost[i] +
StraightTimeCost[i];
			TotalCost = TotalCost + MonthlyCost[i];
		}
		cout << "Pure strategy #3 cost: $" << TotalCost << endl;

		if (TotalCost < PureLowestCost) {
			// Set current plan variables to best plan variables
			copyArray(StartInventory, BestStartInv, PlanningHorizon);
			copyArray(ProductionReqs, BestProdReqs, PlanningHorizon);
			copyArray(ActualProd, BestActualProd, PlanningHorizon);
			copyArray(EndInventory, BestEndInv, PlanningHorizon);
			copyArray(Workforce, BestWorkforce, PlanningHorizon);
			copyArray(MonthlyCost, BestMonthlyCost, PlanningHorizon);
			// Set current cost to best initial cost
			PureLowestCost = TotalCost;
			PureStratFlag = 3;
		}

		/* Determine benchmark solution */
		bool ConstWorkforce = 1;			// [0/1] Determines
whether the workforce will be held constant through the planning horizon
		float WorkforceParam = 0;			// [0-1] Determines the
degree to which the workforce will be adjusted for each iteration

		cout << "\nThe pure strategy with the lowest cost is:" << endl;
		switch (PureStratFlag) {
		case 1:
			cout << "Exact production with varying workforce." << endl <<
endl;
			break;
		case 2:
```

```
             cout << "Constant average workforce with varying working hours, 
including downtime and overtime." << endl << endl;
             break;
       case 3:
             cout << "Constant average workforce with varying inventory and 
stockout." << endl << endl;
       default:
             break;
       }
       /* Print out table of production data */
       // Starting inventories per month
       displayArray("StartInventory:", PlanningHorizon, BestStartInv);
       // Demand forecast per month
       displayArray("DemandForecast:", PlanningHorizon, DemandForecast);
       // Safety stock per month
       displayArray("SafetyStock:", PlanningHorizon, SafetyStock);
       // Production requirement per month
       displayArray("ProductionReq:", PlanningHorizon, BestProdReqs);
       // Actual production per month
       displayArray("ActualProd:", PlanningHorizon, BestActualProd);
       // Ending inventories per month
       displayArray("EndInventory:", PlanningHorizon, BestEndInv);
       // Workforce per month
       displayArray("Workforce:", PlanningHorizon, BestWorkforce);
       // Monthly costs
       displayArray("Monthly Cost:", PlanningHorizon, BestMonthlyCost);
       cout << endl;
       cout << "The total cost of this strategy is : $" << PureLowestCost << 
endl << endl;

/*********************************** END OF PURE STRATEGY COST CALCULATIONS 
**************************************/

/********************************** START OF BIOGEOGRAPHY-BASED OPTIMISATION 
************************************/
       /*    Algorithm:
             1. Set lowest cost pure plan to best solution
             2.1 Initialise algorithm parameters
             2.2 Initialise the population of habitats
             3. For each habitat map its HSI to the number of species, its 
immigrations rate, and its emigration rate
             4 Probabilistically adjust each non-elite habitat using 
migration operators
             5.1 For each habitat update the probability of its species count 
through Eq 2 [Simon2008]
             5.2 Mutate each non-elite habitat according to its probability 
as per Eq 14 [Simon2008]
             5.3 Recompute HSIs
             6. Return to Step 3 for next iteration. Stop at predefined 
number of iterations, or after an acceptable problem solution has been 
reached
       */
```

```
	ofstream resultsFile;							// Link to
a results file to be written to at specific intervals
	resultsFile.open("BBOresults.txt");

	// Reset plan variables ahead of optimisation algorithm
	TotalCost = 0;
	InitialWorkforceCost = 0;

	// Initialise algorithm parameters
	float currentCost = PureLowestCost;				// Set current
best cost to lowest pure strategy plan cost with strategy parameters set in
switch statement above
	int maxIteration = 150;							// Maximum
number of iterations
	int habitats = 100;								// Number
of habitats in the population
	int NoOfElites = 2;								// Number
of elite plans to keep after each iteration
	float mutationProbability = 0.05;				// Mutation
probability per solution per independent variable
	float random = 0;								// Random
number regenerated

	// Initialise the plan parameters
	float planParams[100][14];						// Matrix holding
all plan parameters; rows = habitats, columns = plan parameters
		// [0-5] = InvtoProdRatio per month;
		// [6] = Constant/Non-constant workforce;
		// [7-12] = WorkforceParam per month;
		// [13] = Plan cost;
	float elitePlans[2][14];						// Matrix to keep
elite plans after each iteration
	float eliteParam;								//
Temporary variable for use in elitism operation

	// Initialise an array for holding the minimum cost per iteration
	float minCostPlans[150][14];
	float minCosts[150];

	// Set initial plan parameters
	for (int j = 0; j < habitats; j++) {
		random = (static_cast <float> (rand()) / static_cast <float>
(RAND_MAX));
		for (int i = 0; i < PlanningHorizon; i++) {
			planParams[j][i] = ((float)(rand() % 50 - 25)) / 150;
		}
		if (random < 0.5) { // Constant workforce
			planParams[j][PlanningHorizon] = 1;
		}
		else { // Non-constant workforce
			planParams[j][PlanningHorizon] = 0;
```

```
            }
            for (int i = (PlanningHorizon + 1); i < (2 * PlanningHorizon +
1); i++) {
                        planParams[j][i] = round(((rand() % 2) * 2 -
1)*((float)(rand() % 2)));
            }
      }

/* CALCULATE PLAN COSTS BASED ON INITIAL POPULATION OF HABITATS */
      for (int cnt = 0; cnt < habitats; cnt++) {
            TotalCost = 0;
      /* CONSTANT WORKFORCE */
            if (planParams[cnt][6]) {
                  // Calculate the cost of initial workforce changes
                  InitialWorkforceCost = 0;
                  if (InitialWorkforce < (AvgReqWorkforce +
planParams[cnt][7])) {
                        InitialWorkforceCost = ((AvgReqWorkforce +
planParams[cnt][7]) - InitialWorkforce)*WorkerHiringCost;
                  }
                  else if (InitialWorkforce >(AvgReqWorkforce +
planParams[cnt][7])) {
                        InitialWorkforceCost = (InitialWorkforce -
(AvgReqWorkforce + planParams[cnt][7]))*WorkerLayoffCost;
                  }
                  // Run through the months up to the planning horizon
                  for (int i = 0; i < PlanningHorizon; i++) {
                        // Workforce remains constant
                        Workforce[i] = AvgReqWorkforce +
planParams[cnt][7];
                        ProdHrsAvail[i] = WorkingDays[i] * 8 /*hrs/day*/ *
Workforce[i];
                        // Starting inventory calculated from previous
month's ending inventory
                        if (i == 0) {
                              StartInventory[i] = InitialInventory;
                              MonthlyCost[i] = InitialWorkforceCost;
                        }
                        else {
                              StartInventory[i] = EndInventory[i - 1];
                              MonthlyCost[i] = 0;
                        }
                        // Production calculations
                        // Number of production hours required, based on
the required production for the month
                        ProductionReqs[i] = (DemandForecast[i] +
SafetyStock[i] - StartInventory[i]);
                        ProdHrsReq[i] = round((1 -
planParams[cnt][i])*(ProductionReqs[i] * ProductionTime));
                        // Production hours calculations
                        if (ProdHrsReq[i] < ProdHrsAvail[i]) {
                              StraightTimeHrs[i] = ProdHrsReq[i];
```

```
                                OvertimeHrs[i] = 0;
                                DowntimeHrs[i] = (ProdHrsAvail[i] - 
ProdHrsReq[i]);
                          }
                          else if (ProdHrsReq[i] > ProdHrsAvail[i]) {
                                StraightTimeHrs[i] = ProdHrsAvail[i];
                                OvertimeHrs[i] = ProdHrsReq[i] - 
ProdHrsAvail[i];
                                DowntimeHrs[i] = 0;
                          }
                          else {
                                StraightTimeHrs[i] = ProdHrsAvail[i];
                                OvertimeHrs[i] = 0;
                                DowntimeHrs[i] = 0;
                          }
                          RegularShiftProd[i] = round(StraightTimeHrs[i] / 
ProductionTime);
                          OvertimeProd[i] = round(OvertimeHrs[i] / 
ProductionTime);
                          ActualProd[i] = RegularShiftProd[i] + 
OvertimeProd[i];
                          EndInventory[i] = StartInventory[i] + ActualProd[i] 
- DemandForecast[i];
                          // Inventory/Stockout cost calculations
                          if (EndInventory[i] < 0)
                                ShortageCost[i] = 
abs(EndInventory[i])*StockoutRate;
                          else
                                ShortageCost[i] = 0;
                          if ((EndInventory[i] - SafetyStock[i]) <= 0)
                                UnitsExcess[i] = 0;
                          else
                                UnitsExcess[i] = EndInventory[i] - 
SafetyStock[i];
                          InventoryCost[i] = UnitsExcess[i] * HoldingCost;
                          // Production costs per month
                          StraightTimeCost[i] = StraightTimeHrs[i] * 
StraightTimeRate;
                          OvertimeCost[i] = OvertimeHrs[i] * OverTimeRate;
                          DowntimeCost[i] = DowntimeHrs[i] * DownTimeRate;
                          // Summation of costs per month
                          MonthlyCost[i] += StraightTimeCost[i] + 
OvertimeCost[i] + DowntimeCost[i] + ShortageCost[i] + InventoryCost[i];
                          // Summation of monthly costs
                          TotalCost += MonthlyCost[i];
                    }
                    planParams[cnt][13] = TotalCost;
              }
        /* NON-CONSTANT WORKFORCE */
              else {
                    // Run through the months up to the planning horizon
                    for (int i = 0; i < PlanningHorizon; i++) {
                          // Starting inventory boundary condition
                          if (i == 0)
                                StartInventory[i] = InitialInventory;
                          else
                                StartInventory[i] = EndInventory[i - 1];
                          // Calculation of production hours required
                          ProductionReqs[i] = DemandForecast[i] +
SafetyStock[i] - StartInventory[i];
                          ProdHrsReq[i] = round((1 -
planParams[cnt][i])*(ProductionReqs[i] * ProductionTime)); // Calculate the
required production hours
                          HrsPWorkerPM[i] = WorkingDays[i] * 8 /*hrs/day*/;
                                                                  //
Calculate the available hours per worker per month
                          Workforce[i] = round(ProdHrsReq[i] /
HrsPWorkerPM[i]) + planParams[cnt][7 + i];              // Calculate the
number of workers required
                          ProdHrsAvail[i] = HrsPWorkerPM[i] * Workforce[i];
                          ActualProd[i] = round(ProdHrsAvail[i] /
ProductionTime);
                          if (i == 0) {// Boundary conditions for monthly
required workforce calculation
                                WorkforceDiff = Workforce[i] -
InitialWorkforce;
                          }
                          else {
                                WorkforceDiff = Workforce[i] - Workforce[i -
1];
                          }
                          if (WorkforceDiff == 0) { // Workforce changes -
hiring and laying off
                                NewHires[i] = 0;
                                Layoffs[i] = 0;
                          }
                          else if (WorkforceDiff < 0) {
                                NewHires[i] = 0;
                                Layoffs[i] = -WorkforceDiff;
                          }
                          else {
                                NewHires[i] = WorkforceDiff;
                                Layoffs[i] = 0;
                          }
                          EndInventory[i] = StartInventory[i] + ActualProd[i]
- DemandForecast[i];
                          if (EndInventory[i] < 0)
                                ShortageCost[i] = (-
EndInventory[i])*StockoutRate;
                          else
                                ShortageCost[i] = 0;
                          if ((EndInventory[i] - SafetyStock[i]) < 0)
                                UnitsExcess[i] = 0;
                          else
```

```
                                 UnitsExcess[i] = EndInventory[i] -
SafetyStock[i];
                          InventoryCost[i] = UnitsExcess[i] * HoldingCost;
                          StraightTimeCost[i] = ProdHrsAvail[i] *
StraightTimeRate;                                 // Cost calculations
                          HiringCost[i] = NewHires[i] * WorkerHiringCost;
                          LayoffCost[i] = Layoffs[i] * WorkerLayoffCost;
                          MonthlyCost[i] = StraightTimeCost[i] +
HiringCost[i] + LayoffCost[i] + ShortageCost[i] + InventoryCost[i];
                          // Sum total cost
                          TotalCost += MonthlyCost[i];
                   }
                   planParams[cnt][13] = TotalCost;
            }
      }
/************ End of initial cost calculations *************/

      // Sort plans according to their calculated cost
      sortMatrix(planParams, habitats);

      // Initialise migration variables and parameters
      float mu[100];
      float lambda[100];
      float muSum = 0;
      float select;
      int selectIndex;
      float randomNum;
      float migrator;
      for (int i = 0; i < habitats; i++) {
            mu[i] = (habitats + 1 - (i + 1)) / ((i + 1) + 1);
            lambda[i] = 1 - mu[i];
            muSum += mu[i];
      }

/********************************* MAIN BBO ALGORITHM START HERE
***************************************/
      for (int iter = 0; iter < maxIteration; iter++){
            // Save elite plans
            for (int elites = 0; elites < NoOfElites; elites++) {
                   for (int param = 0; param < 14; param++) {
                          eliteParam = planParams[elites][param];
                          elitePlans[elites][param] = eliteParam;
                   }
            }

      // Migration rates used to determine variable exchange
            for (int it = 0; it < habitats; it++) {
            retry:              // Re-entry point for when a plan fails the
feasibility check
                   // Probabilistic migration of plan parameters
                   for (int i = 0; i < (2*PlanningHorizon+1); i++) {
```

```
                                random = (static_cast <float> (rand()) /
static_cast <float> (RAND_MAX));
                                if (random < lambda[it]) {
                                        randomNum = random*muSum;
                                        select = mu[0];
                                        selectIndex = 0;
                                        while (randomNum > select && selectIndex <
habitats) {
                                                selectIndex++;
                                                select += mu[selectIndex];
                                        }
                                        migrator = planParams[selectIndex][i];
                                        planParams[it][i] = migrator;
                                }
                        }
                // Mutation
                        // Inventory to production ratio mutation
                        for (int paramIndex = 0; paramIndex < PlanningHorizon;
paramIndex++) {
                                random = (static_cast <float> (rand()) /
static_cast <float> (RAND_MAX));
                                if (random < mutationProbability)
                                        planParams[it][paramIndex] = ((float)(rand()
% 50 - 25)) / 150;
                        }
                        // Workforce parameter mutation
                        for (int paramIndex = (PlanningHorizon+1); paramIndex <
(2*PlanningHorizon+1); paramIndex++) {
                                random = (static_cast <float> (rand()) /
static_cast <float> (RAND_MAX));
                                if (random < mutationProbability)
                                        planParams[it][paramIndex] = round(((rand()
% 2) * 2 - 1)*((float)(rand() % 2)));
                        }

        /*************** CALCULATE COST OF NEWLY CONFIGURED PLANS
****************/
                        TotalCost = 0;
                /* CONSTANT WORKFORCE */
                        if (planParams[it][6]) {
                                // Calculate the cost of initial workforce changes
                                InitialWorkforceCost = 0;
                                if (InitialWorkforce < (AvgReqWorkforce +
planParams[it][7])) {
                                        InitialWorkforceCost = ((AvgReqWorkforce +
planParams[it][7]) - InitialWorkforce)*WorkerHiringCost;
                                }
                                else if (InitialWorkforce >(AvgReqWorkforce +
planParams[it][7])) {
                                        InitialWorkforceCost = (InitialWorkforce -
(AvgReqWorkforce + planParams[it][7]))*WorkerLayoffCost;
                                }
```

```
                          // Run through the months up to the planning
horizon
                          for (int i = 0; i < PlanningHorizon; i++) {
                                // Workforce remains constant
                                Workforce[i] = AvgReqWorkforce +
planParams[it][7];
                                ProdHrsAvail[i] = WorkingDays[i] * 8
/*hrs/day*/ * Workforce[i];
                                // Starting inventory calculated from
previous month's ending inventory
                                if (i == 0) {
                                      StartInventory[i] = InitialInventory;
                                      MonthlyCost[i] = InitialWorkforceCost;
                                }
                                else {
                                      StartInventory[i] = EndInventory[i -
1];
                                      MonthlyCost[i] = 0;
                                }
                                // Production calculations
                                // Number of production hours required,
based on the required production for the month
                                ProductionReqs[i] = (DemandForecast[i] +
SafetyStock[i] - StartInventory[i]);
                                ProdHrsReq[i] = round((1 -
planParams[it][i])*(ProductionReqs[i] * ProductionTime));
                                // Production hours calculations
                                if (ProdHrsReq[i] < ProdHrsAvail[i]) {
                                      StraightTimeHrs[i] = ProdHrsReq[i];
                                      OvertimeHrs[i] = 0;
                                      DowntimeHrs[i] = (ProdHrsAvail[i] -
ProdHrsReq[i]);
                                }
                                else if (ProdHrsReq[i] > ProdHrsAvail[i]) {
                                      StraightTimeHrs[i] = ProdHrsAvail[i];
                                      OvertimeHrs[i] = ProdHrsReq[i] -
ProdHrsAvail[i];
                                      DowntimeHrs[i] = 0;
                                }
                                else {
                                      StraightTimeHrs[i] = ProdHrsAvail[i];
                                      OvertimeHrs[i] = 0;
                                      DowntimeHrs[i] = 0;
                                }
                                RegularShiftProd[i] =
round(StraightTimeHrs[i] / ProductionTime);
                                OvertimeProd[i] = round(OvertimeHrs[i] /
ProductionTime);
                                ActualProd[i] = RegularShiftProd[i] +
OvertimeProd[i];
                                EndInventory[i] = StartInventory[i] +
ActualProd[i] - DemandForecast[i];
```

```
                                // Inventory/Stockout cost calculations
                                if (EndInventory[i] < 0)
                                      ShortageCost[i] =
abs(EndInventory[i])*StockoutRate;
                                else
                                      ShortageCost[i] = 0;
                                if ((EndInventory[i] - SafetyStock[i]) <= 0)
                                      UnitsExcess[i] = 0;
                                else
                                      UnitsExcess[i] = EndInventory[i] -
SafetyStock[i];
                                InventoryCost[i] = UnitsExcess[i] *
HoldingCost;
                                // Production costs per month
                                StraightTimeCost[i] = StraightTimeHrs[i] *
StraightTimeRate;
                                OvertimeCost[i] = OvertimeHrs[i] *
OverTimeRate;
                                DowntimeCost[i] = DowntimeHrs[i] *
DownTimeRate;
                                // Summation of costs per month
                                MonthlyCost[i] += StraightTimeCost[i] +
OvertimeCost[i] + DowntimeCost[i] + ShortageCost[i] + InventoryCost[i];
                                // Summation of monthly costs
                                TotalCost += MonthlyCost[i];
                          }
                          planParams[it][13] = TotalCost;
                    }
             /* NON-CONSTANT WORKFORCE */
                    else {
                          // Run through the months up to the planning
horizon
                          for (int i = 0; i < PlanningHorizon; i++) {
                                // Starting inventory boundary condition
                                if (i == 0)
                                      StartInventory[i] = InitialInventory;
                                else
                                      StartInventory[i] = EndInventory[i -
1];
                                // Calculation of production hours required
                                ProductionReqs[i] = DemandForecast[i] +
SafetyStock[i] - StartInventory[i];
                                ProdHrsReq[i] = round((1 -
planParams[it][i])*(ProductionReqs[i] * ProductionTime));  // Calculate the
required production hours
                                HrsPWorkerPM[i] = WorkingDays[i] * 8
/*hrs/day*/;                                                          //
Calculate the available hours per worker per month
                                Workforce[i] = round(ProdHrsReq[i] /
HrsPWorkerPM[i]) + planParams[it][7 + i];                 // Calculate the
number of workers required
```

```
                                ProdHrsAvail[i] = HrsPWorkerPM[i] *
Workforce[i];
                                ActualProd[i] = round(ProdHrsAvail[i] /
ProductionTime);
                                if (i == 0) {// Boundary conditions for
monthly required workforce calculation
                                      WorkforceDiff = Workforce[i] -
InitialWorkforce;
                                }
                                else {
                                      WorkforceDiff = Workforce[i] -
Workforce[i - 1];
                                }
                                if (WorkforceDiff == 0) { // Workforce
changes - hiring and laying off
                                      NewHires[i] = 0;
                                      Layoffs[i] = 0;
                                }
                                else if (WorkforceDiff < 0) {
                                      NewHires[i] = 0;
                                      Layoffs[i] = -WorkforceDiff;
                                }
                                else {
                                      NewHires[i] = WorkforceDiff;
                                      Layoffs[i] = 0;
                                }
                                EndInventory[i] = StartInventory[i] +
ActualProd[i] - DemandForecast[i];
                                if (EndInventory[i] < 0)
                                      ShortageCost[i] = (-
EndInventory[i])*StockoutRate;
                                else
                                      ShortageCost[i] = 0;
                                if ((EndInventory[i] - SafetyStock[i]) < 0)
                                      UnitsExcess[i] = 0;
                                else
                                      UnitsExcess[i] = EndInventory[i] -
SafetyStock[i];
                                InventoryCost[i] = UnitsExcess[i] *
HoldingCost;
                                StraightTimeCost[i] = ProdHrsAvail[i] *
StraightTimeRate;                                  // Cost calculations
                                HiringCost[i] = NewHires[i] *
WorkerHiringCost;
                                LayoffCost[i] = Layoffs[i] *
WorkerLayoffCost;
                                MonthlyCost[i] = StraightTimeCost[i] +
HiringCost[i] + LayoffCost[i] + ShortageCost[i] + InventoryCost[i];
                                // Sum total cost
                                TotalCost += MonthlyCost[i];
                          }
                          planParams[it][13] = TotalCost;
```

```
                    }
                    // Plan feasibility check
                    if (EndInventory[PlanningHorizon - 1] < 0)
                          goto retry;
              }
/*********************** End of newly configure plan cost calculations
************************/

              // Sort plans from lowest to highest cost
              sortMatrix(planParams, habitats);
              // Replace worst plans with elite plans from previous iteration
              for (int elites = 0; elites < NoOfElites; elites++) {
                    for (int param = 0; param < 14; param++) {
                          eliteParam = elitePlans[elites][param];
                          planParams[(habitats - 1) - elites][param] =
eliteParam;
                    }
              }
              // Sort plans from lowest to highest cost
              sortMatrix(planParams, habitats);
              // Record lowest cost plan of current iteration
              for (int param = 0; param < 14; param++) {
                    minCostPlans[iter][param] = planParams[0][param];
              }
              minCosts[iter] = minCostPlans[iter][13];
              for (int i = 0; i < 14; i++)
                    resultsFile << minCostPlans[iter][i] << '\t';
              resultsFile << endl;
       }
/***************************** END OF BBO ALGORITHM
***********************************/

       // Sort minimum cost plans matrix from lowest cost to highest cost
       sortMatrix(minCostPlans, maxIteration);

       for (int i = 0; i < 14; i++) {
              cout << minCostPlans[0][i] << '\t';
       }
       cout << endl;

/* Calculate best plan variables before printing them out*/
       TotalCost = 0;
/* CONSTANT WORKFORCE */
       if (minCostPlans[0][6]) {
              // Calculate the cost of initial workforce changes
              InitialWorkforceCost = 0;
              if (InitialWorkforce < (AvgReqWorkforce + minCostPlans[0][7])) {
                    InitialWorkforceCost = ((AvgReqWorkforce +
minCostPlans[0][7]) - InitialWorkforce)*WorkerHiringCost;
              }
              else if (InitialWorkforce > (AvgReqWorkforce +
minCostPlans[0][7])) {
```

```
                    InitialWorkforceCost = (InitialWorkforce - 
(AvgReqWorkforce + minCostPlans[0][7]))*WorkerLayoffCost;
            }
            // Run through the months up to the planning horizon
            for (int i = 0; i < PlanningHorizon; i++) {
                    // Workforce remains constant
                    Workforce[i] = AvgReqWorkforce + minCostPlans[0][7];
                    ProdHrsAvail[i] = WorkingDays[i] * 8 /*hrs/day*/ * 
Workforce[i];
                    // Starting inventory calculated from previous month's 
ending inventory
                    if (i == 0) {
                          StartInventory[i] = InitialInventory;
                          MonthlyCost[i] = InitialWorkforceCost;
                    }
                    else {
                          StartInventory[i] = EndInventory[i - 1];
                          MonthlyCost[i] = 0;
                    }
                    // Production calculations
                    // Number of production hours required, based on the 
required production for the month
                    ProductionReqs[i] = (DemandForecast[i] + SafetyStock[i] - 
StartInventory[i]);
                    ProdHrsReq[i] = round((1 - 
minCostPlans[0][i])*(ProductionReqs[i] * ProductionTime));
                    // Production hours calculations
                    if (ProdHrsReq[i] < ProdHrsAvail[i]) {
                          StraightTimeHrs[i] = ProdHrsReq[i];
                          OvertimeHrs[i] = 0;
                          DowntimeHrs[i] = (ProdHrsAvail[i] - ProdHrsReq[i]);
                    }
                    else if (ProdHrsReq[i] > ProdHrsAvail[i]) {
                          StraightTimeHrs[i] = ProdHrsAvail[i];
                          OvertimeHrs[i] = ProdHrsReq[i] - ProdHrsAvail[i];
                          DowntimeHrs[i] = 0;
                    }
                    else {
                          StraightTimeHrs[i] = ProdHrsAvail[i];
                          OvertimeHrs[i] = 0;
                          DowntimeHrs[i] = 0;
                    }
                    RegularShiftProd[i] = round(StraightTimeHrs[i] / 
ProductionTime);
                    OvertimeProd[i] = round(OvertimeHrs[i] / ProductionTime);
                    ActualProd[i] = RegularShiftProd[i] + OvertimeProd[i];
                    EndInventory[i] = StartInventory[i] + ActualProd[i] - 
DemandForecast[i];
                    // Inventory/Stockout cost calculations
                    if (EndInventory[i] < 0)
                          ShortageCost[i] = 
abs(EndInventory[i])*StockoutRate;
```

```
                    else
                          ShortageCost[i] = 0;
                    if ((EndInventory[i] - SafetyStock[i]) <= 0)
                          UnitsExcess[i] = 0;
                    else
                          UnitsExcess[i] = EndInventory[i] - SafetyStock[i];
                    InventoryCost[i] = UnitsExcess[i] * HoldingCost;
                    // Production costs per month
                    StraightTimeCost[i] = StraightTimeHrs[i] *
StraightTimeRate;
                    OvertimeCost[i] = OvertimeHrs[i] * OverTimeRate;
                    DowntimeCost[i] = DowntimeHrs[i] * DownTimeRate;
                    // Summation of costs per month
                    MonthlyCost[i] += StraightTimeCost[i] + OvertimeCost[i] +
DowntimeCost[i] + ShortageCost[i] + InventoryCost[i];
                    // Summation of monthly costs
                    TotalCost += MonthlyCost[i];
             }

             minCostPlans[0][13] = TotalCost;
      }
/* NON-CONSTANT WORKFORCE */
      else {
             // Run through the months up to the planning horizon
             for (int i = 0; i < PlanningHorizon; i++) {
                    // Starting inventory boundary condition
                    if (i == 0)
                          StartInventory[i] = InitialInventory;
                    else
                          StartInventory[i] = EndInventory[i - 1];
                    // Calculation of production hours required
                    ProductionReqs[i] = DemandForecast[i] + SafetyStock[i] -
StartInventory[i];
                    ProdHrsReq[i] = round((1 -
minCostPlans[0][i])*(ProductionReqs[i] * ProductionTime)); // Calculate the
required production hours
                    HrsPWorkerPM[i] = WorkingDays[i] * 8 /*hrs/day*/;
                                                              // Calculate the
available hours per worker per month
                    Workforce[i] = round(ProdHrsReq[i] / HrsPWorkerPM[i]) +
minCostPlans[0][7 + i];                             // Calculate the number
of workers required
                    ProdHrsAvail[i] = HrsPWorkerPM[i] * Workforce[i];
                    ActualProd[i] = round(ProdHrsAvail[i] / ProductionTime);
                    if (i == 0) {// Boundary conditions for monthly required
workforce calculation
                          WorkforceDiff = Workforce[i] - InitialWorkforce;
                    }
                    else {
                          WorkforceDiff = Workforce[i] - Workforce[i - 1];
                    }
```

```
			if (WorkforceDiff == 0) { // Workforce changes - hiring
and laying off
				NewHires[i] = 0;
				Layoffs[i] = 0;
			}
			else if (WorkforceDiff < 0) {
				NewHires[i] = 0;
				Layoffs[i] = -WorkforceDiff;
			}
			else {
				NewHires[i] = WorkforceDiff;
				Layoffs[i] = 0;
			}
			EndInventory[i] = StartInventory[i] + ActualProd[i] -
DemandForecast[i];
			if (EndInventory[i] < 0)
				ShortageCost[i] = (-EndInventory[i])*StockoutRate;
			else
				ShortageCost[i] = 0;
			if ((EndInventory[i] - SafetyStock[i]) < 0)
				UnitsExcess[i] = 0;
			else
				UnitsExcess[i] = EndInventory[i] - SafetyStock[i];
			InventoryCost[i] = UnitsExcess[i] * HoldingCost;
			StraightTimeCost[i] = ProdHrsAvail[i] * StraightTimeRate;
						// Cost calculations
			HiringCost[i] = NewHires[i] * WorkerHiringCost;
			LayoffCost[i] = Layoffs[i] * WorkerLayoffCost;
			MonthlyCost[i] = StraightTimeCost[i] + HiringCost[i] +
LayoffCost[i] + ShortageCost[i] + InventoryCost[i];
			// Sum total cost
			TotalCost += MonthlyCost[i];
		}
		minCostPlans[0][13] = TotalCost;
	}
/****************** End of lowest plan cost calculations
***********************/

	int LowestBBOCostElement = MinElementIndex(minCosts, maxIteration);
	float LowestBBOCost;
	LowestBBOCost = minCosts[LowestBBOCostElement];
	cout << "Lowest cost found by BBO is: $" << LowestBBOCost << endl;

	resultsFile.close();

	cout << endl;
	/* Print out table of production data for optimal plan */
	cout << "The most optimal plan is configured as follows:\n" << endl;
	// Starting inventories per month
	displayArray("StartInventory:", PlanningHorizon, StartInventory);
	// Demand forecast per month
	displayArray("DemandForecast:", PlanningHorizon, DemandForecast);
	// Safety stock per month
	displayArray("SafetyStock:", PlanningHorizon, SafetyStock);
	// Production requirement per month
	displayArray("ProductionReq:", PlanningHorizon, ProductionReqs);
	// Actual production per month
	displayArray("ActualProd:", PlanningHorizon, ActualProd);
	// Ending inventories per month
	displayArray("EndInventory:", PlanningHorizon, EndInventory);
	// Workforce per month
	displayArray("Workforce:", PlanningHorizon, Workforce);
	// Monthly costs
	displayArray("Monthly Cost:", PlanningHorizon, MonthlyCost);
	cout << "\nThe total cost of this plan is: $" << LowestBBOCost <<
endl;

	cout << "\nSaving of " << (((PureLowestCost - LowestBBOCost) /
PureLowestCost) * 100) << "%" << " of $" << PureLowestCost << endl;

/* Delete arrays dynamically allocated in memory */
	delete[] WorkingDays, DemandForecast, SafetyStock, StartInventory,
ProductionReqs, EndInventory;
	delete[] ProdHrsReq, HrsPWorkerPM, Workforce, WorkforceChange,
NewHires, HiringCost, Layoffs, LayoffCost, StraightTimeHrs;
	delete[] StraightTimeCost, ProdHrsAvail, ActualProd, UnitsShort,
ShortageCost, UnitsExcess, InventoryCost;
	delete[] RegularShiftProd, UnitsAvail, UnitsPreOvertime, OvertimeProd,
OvertimeCost, DowntimeHrs, DowntimeCost, MonthlyCost;
	delete[] MaxStockout, MaxInventory, MaxOvertime, currentActualProd,
currentEndInv, currentMonthlyCost, currentProdReqs;
	delete[] currentStartInv, currentWorkforce, BestActualProd,
BestEndInv, BestMonthlyCost, BestProdReqs, BestStartInv, BestWorkforce,
InvtoProdRatio;

/** End matter of main programme **/
	cout << endl << endl;
	std::system("PAUSE");
	return 0;
}
```

Appendix E Dynamic Simulation Model Data

Table E.1. Simio simulation model travel times table.

TransferPath	TravelTime [mins]
TP1_CmbA4_CmbA5	Random.Pert(1.8,2.,2.4)
TP2_CmbA5_CmbOrderCompletion	Random.Pert(1.5,2.5,4.)
TP3_CmbA1_CmbA4	Random.Pert(4.6,5.,5.4)
TP4_Cmb2_CmbA4	Random.Pert(4.5,5.,5.5)
TP5_CmbA3_CmbA4	Random.Pert(4.7,5.,5.4)
TP6_DialMat_WSM3	Random.Pert(1.2,2.5,3.9)
TP7_WSM3_WSM4	Random.Pert(2.1,2.6,2.9)
TP8_WSM4_WSM7	Random.Pert(2.4,2.8,3.3)
TP9_Movement_CmbA2	Random.Pert(6.3,7.5,8.8)
TP10_SpacerMat_WSM8	Random.Pert(7.,8.1,9.2)
TP11_WSM8_CmbA4	Random.Pert(6.4,7.,7.9)
TP12_BezelMat_WSM5	Random.Pert(.6,3.,4.5)
TP13_WSM5_WSM1	Random.Pert(3.,3.1,3.3)
TP14_WSM1_WSM2	Random.Pert(3.6,4.,4.5)
TP15_WSM1_WSM6	Random.Pert(5.8,6.,6.4)
TP16_WSM2_WSM6	Random.Pert(1.9,2.4,2.9)
TP17_WSM6_CmbA1	Random.Pert(3.3,4.1,5.)
TP18_WSM6_CmbA3	Random.Pert(3.4,4.1,5.1)
TP19_WSM6_CmbA4	Random.Pert(4.3,5.2,6.)
TP20_WSM7_CmbA2	Random.Pert(1.4,2.2,3.1)
TP21_CrownMat_WSM5	Random.Pert(1.6,2.9,3.8)
TP22_BackCaseMat_WSM5	Random.Pert(1.8,2.8,3.9)
TP23_WSM2_CmbA4	Random.Pert(7.2,8.,9.)
TP24_CaseMat_WSM1	Random.Pert(4.3,5.5,6.8)
TP25_BackCaseGasket_CmbA4	Random.Pert(4.6,6.1,7.4)
TP26_GlassGasket_CmbA4	Random.Pert(6.3,7.6,8.9)
TP27_Glass_CmbA4	Random.Pert(7.,8.2,9.5)
TP28_Bracelet_CmbA5	Random.Pert(3.9,5.2,6.6)
TP29_BraceletPin_CmbA5	Random.Pert(4.,5.3,6.9)
TP30_CrownCT_CmbA1	Random.Pert(6.8,8.,9.5)
TP31_PusherScrew_CmbA1	Random.Pert(6.5,7.9,9.4)
TP32_Pusher_CmbA1	Random.Pert(6.4,7.8,9.3)
TP33_PusherSeal_CmbA1	Random.Pert(7.,8.1,9.2)
TP34_PusherSpring_CmbA1	Random.Pert(6.6,8.,9.1)
TP35_PusherCT_CmbA1	Random.Pert(6.4,7.8,9.3)

TP36_Stem_CmbA3	Random.Pert(4.4,5.8,7.3)
TP37_CrownSeal_CmbA3	Random.Pert(4.3,5.6,7.1)
TP38_dialHP_CmbA2	Random.Pert(1.4,2.8,4.1)
TP39_HourHand_CmbA2	Random.Pert(1.3,2.7,4.)
TP40_MinuteHand_CmbA2	Random.Pert(1.6,2.9,4.4)
TP41_SecondHand_CmbA2	Random.Pert(1.2,2.7,3.9)
TP42_SmallHand_CmbA2	Random.Pert(1.5,2.9,4.2)
TP43_Orders_OrderProcessing	Random.Pert(4.7,5.,5.2)

Table E.2. Simio simulation model part routing table.

PartType	Sequence	SetupTimes [mins]	ProcessingTimes [mins]	Remaining Times [mins]	Remaining Steps
EntOrder	SvrOrderProcessing	0	ModelEntity.StaUnitsPerOrder*Random.Uniform(0,1)	0	0
EntOrder	ParentInput@CmbOrderCompletion	0	ModelEntity.StaUnitsPerOrder*Random.Uniform(10,15)	0	0
EntBezel	SvrBezelMaterial	0	0	0	6
EntBezel	WSM5	Random.LogNormal(2.3978452753,.00999975)	Random.LogNormal(2.2512418011,.00999975)	16.5	5
EntBezel	WSM1	Math.If(ModelEntity.StaChrono!=StaIntChronoM1, Random.LogNormal(1.3860944011,.0199980004), 0)	Random.LogNormal(.6929472205,.0199980004)	7	4
EntBezel	WSM2	Random.LogNormal(.9162407344,.00999975)	Random.LogNormal(.6930971831,.00999975)	5	3
EntBezel	WSM6	Math.If(ModelEntity.StaPVDd, Random.LogNormal(.6918987405,.0499687922), 0)	Math.If(ModelEntity.StaPVDd, Random.LogNormal(1.0973638486,.0499687922), 0)	3	2
EntBezel	MemberInput@CmbA4	0	0	0	1
EntCase	SvrCaseMaterial	0	0	0	5
EntCase	WSM1	Math.If(ModelEntity.StaChrono!=StaIntChronoM1, Random.LogNormal(3.0443224777,.0199980004), 0)	Random.LogNormal(3.295636906,.0199980004)	33	4
EntCase	WSM2	Random.LogNormal(.9162407344,.00999975)	Random.LogNormal(1.0985622912,.00999975)	6	3
EntCase	WSM6	Math.If(ModelEntity.StaPVDd, Random.LogNormal(.6918987405,.0499687922), 0)	Math.If(ModelEntity.StaPVDd, Random.LogNormal(1.0973638486,.0499687922), 0)	3	2
EntCase	ParentInput@CmbA1	Math.If(ModelEntity.StaChrono!=StaChronoCaseSAssy, Random.LogNormal(-.0049751654,.0997513451), 0)	Random.LogNormal(1.8668270115,.0997513451)	0	1

PartType	Sequence	SetupTimes [mins]	ProcessingTimes [mins]	Remaining Times [mins]	Remaining Steps
EntCrown	SvrCrownMaterial	0	0	0	5
EntCrown	WSM5	Random.LogNormal(1.9458601516,.00999975)	Random.LogNormal(2.3025350955,.00999975)	17	4
EntCrown	WSM1	Math.If(ModelEntity.StaChrono!=StaIntChronoM1, Random.LogNormal(1.7915595092,.0199980004), 0)	Random.LogNormal(1.3860944011,.0199980004)	7	3
EntCrown	WSM6	Math.If(ModelEntity.StaPVDd, Random.LogNormal(.6918987405,.0499687922), 0)	Math.If(ModelEntity.StaPVDd, Random.LogNormal(1.0973638486,.0499687922), 0)	3	2
EntCrown	ParentInput@CmbA3	Random.LogNormal(-.0049751654,.0997513451)	Random.LogNormal(1.0936371232,.0997513451)	0	1
EntBackCase	SvrBackCaseMaterial	0	0	0	5
EntBackCase	WSM5	Random.LogNormal(2.3025350955,.00999975)	Random.LogNormal(2.3025350955,.00999975)	17	4
EntBackCase	WSM1	Math.If(ModelEntity.StaChrono!=StaIntChronoM1, Random.LogNormal(1.3860944011,.0199980004), 0)	Random.LogNormal(1.6092379524,.0199980004)	7	3
EntBackCase	WSM2	Random.LogNormal(.6930971831,.00999975)	Random.LogNormal(.6930971831,.00999975)	2	2
EntBackCase	MemberInput@CmbA4	0	0	0	1
EntDial	SvrDialMaterial	0	0	0	5
EntDial	WSM3	Random.LogNormal(.9160907719,.0199980004)	Random.LogNormal(-.00019996,.0199980004)	9.5	4
EntDial	WSM4	Math.If(ModelEntity.StaChrono!=StaIntChronoM4, Random.LogNormal(1.6092379524,.0199980004), 0)	Random.LogNormal(2.0147030605,.0199980004)	8.5	3
EntDial	WSM7	Math.If(ModelEntity.StaChrono!=StaIntChronoM7, Random.LogNormal(.6929472205,.0199980004), 0)	Random.LogNormal(-.00019996,.0199980004)	1	2

PartType	Sequence	SetupTimes [mins]	ProcessingTimes [mins]	Remaining Times [mins]	Remaining Steps
EntDial	MemberInput@CmbA2	0	0	0	1
EntSpacer	SvrSpacerMaterial	0	0	0	3
EntSpacer	WSM8	Math.If(ModelEntity.StaChrono!=StaIntChronoM8, Random.LogNormal(.6907031635,.0699144768), 0)	Random.LogNormal(3.2164318078,.0699144768)	25	2
EntSpacer	MemberInput@CmbA4	0	0	0	1
EntPusherCaseTube	SvrPusherCaseTube	0	0	0	0
EntPusherCaseTube	MemberInput@CmbA1	0	0	0	0
EntPusherSpring	SvrPusherSpring	0	0	0	0
EntPusherSpring	MemberInput@CmbA1	0	0	0	0
EntPusherSeal	SvrPusherSeal	0	0	0	0
EntPusherSeal	MemberInput@CmbA1	0	0	0	0
EntPusher	SvrPusher	0	0	0	0
EntPusher	MemberInput@CmbA1	0	0	0	0
EntPusherScrew	SvrPusherScrew	0	0	0	0
EntPusherScrew	MemberInput@CmbA1	0	0	0	0
EntCrownCaseTube	SvrCrownCaseTube	0	0	0	0

PartType	Sequence	SetupTimes [mins]	ProcessingTimes [mins]	Remaining Times [mins]	Remaining Steps
EntCrownCaseTube	MemberInput@CmbA1	0	0	0	0
EntMovement	SvrMovement	0	0	0	0
EntMovement	ParentInput@CmbA2	Math.If(ModelEntity.StaChrono!=StaChronoTimeSAssy, Random.LogNormal(-.0196103566,.1980422004), 0)	Random.LogNormal(1.9262997925,.1980422004)	0	0
EntStem	SvrStem	0	0	0	0
EntStem	MemberInput@CmbA3	0	0	0	0
EntCrownSeal	SvrCrownSeal	0	0	0	0
EntCrownSeal	MemberInput@CmbA3	0	0	0	0
EntDialHoldingPin	SvrDialHoldingPin	0	0	0	0
EntDialHoldingPin	MemberInput@CmbA2	0	0	0	0
EntHourHand	SvrHourHand	0	0	0	0
EntHourHand	MemberInput@CmbA2	0	0	0	0
EntMinuteHand	SvrMinuteHand	0	0	0	0
EntMinuteHand	MemberInput@CmbA2	0	0	0	0
EntSecondHand	SvrSecondHand	0	0	0	0
EntSecondHand	MemberInput@CmbA2	0	0	0	0

PartType	Sequence	SetupTimes [mins]	ProcessingTimes [mins]	Remaining Times [mins]	Remaining Steps
EntSmallHand	SvrSmallHand	0	0	0	0
EntSmallHand	MemberInput@CmbA2	0	0	0	0
EntGlass	SvrGlass	0	0	0	0
EntGlass	MemberInput@CmbA4	0	0	0	0
EntGlassGasket	SvrGlassGasket	0	0	0	0
EntGlassGasket	MemberInput@CmbA4	0	0	0	0
EntBackCaseGasket	SvrBackCaseGasket	0	0	0	0
EntBackCaseGasket	MemberInput@CmbA4	0	0	0	0
EntBracelet	SvrBracelet	0	0	0	0
EntBracelet	MemberInput@CmbA5	0	0	0	0
EntBraceletPin	SvrBraceletPin	0	0	0	0
EntBraceletPin	MemberInput@CmbA5	0	0	0	0
EntCaseSubAssy	ParentInput@CmbA4	Random.LogNormal(.673536824,.1980422004)	Random.LogNormal(1.7721491127,.1980422004)	0	0
EntBodySubAssy	ParentInput@CmbA5	Random.LogNormal(-.0012484401,.0499687922)	Random.LogNormal(1.7905110291,.0499687922)	0	0

Appendix F   Distributed Dynamic Selection Rule Strategy Programming Code

Code adapted from Simio user extension example for user defined selction rules [68].

```
using System;
using System.Collections.Generic;
using System.Text;
using SimioAPI;
using SimioAPI.Extensions;

namespace AdaptiveSelectionRule
{
    public class AdaptiveSelectionRuleDefinition : ISelectionRuleDefinition
    {
        #region ISelectionRuleDefinition Members

        /// <summary>
        /// Property returning the name of the rule.
        /// </summary>
        public string Name
        {
            get { return "Adaptive Selection Rule"; }
        }

        /// <summary>
        /// Property returning a description of the selection rule.
        /// </summary>
        public string Description
        {
            get { return "Description text for the 'Adaptive Selection
Rule'."; }
        }

        /// <summary>
        /// Property returning an icon to display for the selection.
        /// </summary>
        public System.Drawing.Image Icon
        {
            get { return null; }
        }

        /// <summary>
        /// Property returning a unique static GUID for the selection rule.
        /// </summary>
        public Guid UniqueID
        {
            get { return MY_ID; }
        }
        static readonly Guid MY_ID = new Guid("33ef8a89-23af-4592-a12f-
697282ce610c");

        /// <summary>
        /// Method called that defines the property schema for the selection
rule.
        /// </summary>
        public void DefineSchema(IPropertyDefinitions schema)
        {
            IPropertyDefinition pd;

            // Default selection rule expression for Smallest Value First
rule implementation - EDD
            pd = schema.AddExpressionProperty("WarmUpPeriodExpression",
"Candidate.ModelEntity.StaUnitNo");
            pd.Description = "The expression used for a Smallest Value First
dynamic selection rule. " +
                "In the expression, use the keyword 'Candidate' to reference
an object in the collection of candidates (e.g.,
Candidate.Entity.Priority).";
            pd.Required = true;
            pd.DisplayName = " Warm-up Period Expression";

            // Default selection rule expression for Smallest Value First
rule implementation - EDD
            pd = schema.AddExpressionProperty("DefaultSVFExpression",
"Candidate.ModelEntity.StaTimeEntered");
            pd.Description = "EDD" +
                "In the expression, use the keyword 'Candidate' to reference
an object in the collection of candidates (e.g.,
Candidate.Entity.Priority).";
            pd.Required = true;
            pd.DisplayName = "EDD Expression";

            // Secondary selection rule expression for Smallest Value First
rule implementation - SPT
            pd = schema.AddExpressionProperty("SecondarySVFExpression",
"PartRouting.ProcessingTimes");
            pd.Description = "SPT" +
                "In the expression, use the keyword 'Candidate' to reference
an object in the collection of candidates (e.g.,
Candidate.Entity.Priority).";
            pd.Required = true;
            pd.DisplayName = "SPT Expression";

            // Default selection rule expression for Largest Value First rule
implementation - MPTR
            pd = schema.AddExpressionProperty("DefaultLVFExpression",
"PartRouting.RemainingTimes");
            pd.Description = "LPTR/MPTR" +
                "In the expression, use the keyword 'Candidate' to reference
an object in the collection of candidates (e.g.,
Candidate.Entity.Priority).";
            pd.Required = true;
            pd.DisplayName = "LPTR/MPTR Expression";

            // First system performance indicator variable: Instantaneous
order WIP vs Average order WIP
            pd = schema.AddExpressionProperty("OrderWIPExpression",
"Candidate.ModelEntity.StaOrderWIP / Candidate.ModelEntity.StaAvgOrderWIP");
            pd.Description = "None";
            pd.Required = true;
            pd.DisplayName = "Order WIP Ratio Expression";

            // Instantaneous order WIP vs Average order WIP threshold value
            pd = schema.AddExpressionProperty("OrderWIPRatioValue",
"Candidate.ModelEntity.StaOrderWIPRatioValue");
            pd.Description = "None";
            pd.Required = true;
            pd.DisplayName = "Order WIP Ratio Threshold";

            // Order arrival rate over last Harmony Memory Consideration Rate
Period
            pd = schema.AddExpressionProperty("OrderArrivalExpression",
"ArrivalPeriod");
            pd.Description = "None";
            pd.Required = true;
            pd.DisplayName = "Arrival Period Expression";

            // Local buffer level variable
            pd = schema.AddExpressionProperty("BufferLevelExpression",
"WSM1.InputBuffer.Contents.NumberWaiting /
WSM1.InputBuffer.Contents.AverageNumberWaiting");
            pd.Description = "None";
            pd.Required = true;
            pd.DisplayName = "Buffer Level Expression";

            // Upper buffer level threshold variable
            pd = schema.AddExpressionProperty("TopBufferThresholdValue",
"Candidate.ModelEntity.StaTopBufferThreshold");
            pd.Description = "None";
            pd.Required = true;
            pd.DisplayName = "Upper Buffer Threshold";

            // Lower buffer level threshold variable
            pd = schema.AddExpressionProperty("BotBufferThresholdValue",
"Candidate.ModelEntity.StaBotBufferThreshold");
            pd.Description = "None";
            pd.Required = true;
            pd.DisplayName = "Lower Buffer Threshold";

        }
        /// <summary>
        /// Method called to add a new instance of this selection rule type
to a model.
        /// Returns an instance of the class implementing the ISelectionRule
interface.
        /// </summary>
        public ISelectionRule CreateRule(IPropertyReaders properties)
        {
            return new AdaptiveSelectionRule(properties);
        }

        #endregion

    }
    public class AdaptiveSelectionRule : ISelectionRule
    {
        IPropertyReader _warmUpPeriodProperty;
        IPropertyReader _defaultSVFProperty;
        IPropertyReader _secondarySVFProperty;
        IPropertyReader _defaultLVFProperty;
        IPropertyReader _bufferLevelProperty;
        IPropertyReader _topBufferThresholdProperty;
        IPropertyReader _botBufferThresholdProperty;
        IPropertyReader _orderWIPProperty;
        IPropertyReader _orderWIPValueProperty;
        IPropertyReader _orderArrivalProperty;

        public AdaptiveSelectionRule(IPropertyReaders properties)
        {
            _warmUpPeriodProperty =
properties.GetProperty("WarmUpPeriodExpression");
            _defaultSVFProperty =
properties.GetProperty("DefaultSVFExpression");
            _secondarySVFProperty =
properties.GetProperty("SecondarySVFExpression");
            _defaultLVFProperty =
properties.GetProperty("DefaultLVFExpression");
            _bufferLevelProperty =
properties.GetProperty("BufferLevelExpression");
            _topBufferThresholdProperty =
properties.GetProperty("TopBufferThresholdValue");
            _botBufferThresholdProperty =
properties.GetProperty("BotBufferThresholdValue");
            _orderWIPProperty = properties.GetProperty("OrderWIPExpression");
            _orderWIPValueProperty =
properties.GetProperty("OrderWIPRatioValue");
            _orderArrivalProperty =
properties.GetProperty("OrderArrivalExpression");
        }

        #region ISelectionRule Members
        // Rule implementation level variables
        double oldArrivalPeriod = 0;
        double newArrivalPeriod = 0;
        double oldLocalBufferVar = 0;

        /// <summary>
        /// Method called when the selection rule is being used to select an
item from a collection of candidates.
        /// </summary>
        public IExecutionContext Select(IEnumerable<IExecutionContext>
candidates)
        {
            // Select method level variables
            bool switchTime = true;
            string switchOutcome = null;
            double unitsProcessed = 0;
            double localBufferVar = 0;
            IExecutionContext selectedCandidate = null;
            IExecutionContext chosenOne = null;
            double largestValue = double.NegativeInfinity;
            double orderWIPparam = 0;
            double orderWIPRatioValue = 0;
            bool consider = false;
            double topBufferThreshold = 0;
            double botBufferThreshold = 0;

            // load execution context-specific variables
            foreach (IExecutionContext candidate in candidates)
            {
                //unitsProcessed =
_defaultSVFProperty.GetDoubleValue(candidate);
                unitsProcessed =
_warmUpPeriodProperty.GetDoubleValue(candidate);
                localBufferVar =
_bufferLevelProperty.GetDoubleValue(candidate);
                topBufferThreshold =
_topBufferThresholdProperty.GetDoubleValue(candidate);
                botBufferThreshold =
_botBufferThresholdProperty.GetDoubleValue(candidate);
                orderWIPparam = _orderWIPProperty.GetDoubleValue(candidate);
                orderWIPRatioValue =
_orderWIPValueProperty.GetDoubleValue(candidate);
                newArrivalPeriod =
_orderArrivalProperty.GetDoubleValue(candidate);
            }

            // if the current arrival period, i.e. arrival rate, is different
from the previous one
            if (newArrivalPeriod < oldArrivalPeriod)
            {
                switchTime = true;
                oldArrivalPeriod = newArrivalPeriod;
            }
            else
            {
                switchTime = false;
                oldArrivalPeriod = newArrivalPeriod;
            }

            // Wait for the warm-up period to run through
            if (unitsProcessed < 350)
            {
                consider = false;
            }
            // Start the selection rule dynamics after warm-up period has
elapsed
            else
            {
                // Decide whether to consider switching selection rules based
on order WIP ratio variable and updated order arrival rate
                if (orderWIPparam >= orderWIPRatioValue && switchTime)
                {
                    consider = true;
                }
                else
                {
                    consider = false;
                }
            }

            // If conditions are met for considering switching rules query
the value of the local buffer variable
            if (consider)
            {
                if (localBufferVar > botBufferThreshold && localBufferVar <
topBufferThreshold)
                {
                    // Query the trend of the local buffer variable
                    if (localBufferVar < oldLocalBufferVar)
                    {
                        // Hold status quo
                    }
                    else
                    {
                        // Try one of the other rules
                        if (switchOutcome == null)
                            switchOutcome = "fsvf";
                        else if (switchOutcome == "fsvf")
                            switchOutcome = "ssvf";
                        else if (switchOutcome == "ssvf")
                            switchOutcome = "lvf";
                        else if (switchOutcome == "lvf")
                            switchOutcome = null;
                    }
                }
                else if (localBufferVar > topBufferThreshold)
                {
                    // Try one of the other rules
                    if (switchOutcome == null)
                        switchOutcome = "fsvf";
                    else if (switchOutcome == "fsvf")
                        switchOutcome = "ssvf";
                    else if (switchOutcome == "ssvf")
                        switchOutcome = "lvf";
                    else if (switchOutcome == "lvf")
                        switchOutcome = null;
                }
                else
                {
                    // Hold status quo
                }
            }

            // Save the value of the current local buffler variable
            oldLocalBufferVar = localBufferVar;

            switch (switchOutcome)
            {
                case "lvf":
                    // Loop through the collection of candidates to return
the candidate with the largest value of the default expression.
                    largestValue = double.NegativeInfinity;
                    foreach (IExecutionContext candidate in candidates)
                    {
                        double thisValue =
_defaultLVFProperty.GetDoubleValue(candidate);
                        if (thisValue > largestValue)
                        {
                            largestValue = thisValue;
                            selectedCandidate = candidate;
                        }
                    }
                    chosenOne = selectedCandidate;
                    break;

                case "fsvf":
                    // Loop through the collection of candidates to return a
selected item based on the secondary Smallest Value First selection
expression
                    double smallestValue = double.PositiveInfinity;

                    foreach (IExecutionContext candidate in candidates)
                    {
                        double thisValue =
_secondarySVFProperty.GetDoubleValue(candidate);
                        if (thisValue < smallestValue)
                        {
                            smallestValue = thisValue;
                            selectedCandidate = candidate;
                        }
                    }
                    chosenOne = selectedCandidate;
                    break;

                case "ssvf":
                    // Loop through the collection of candidates to return a
selected item based on the secondary Smallest Value First selection
expression
                    double sSmallestValue = double.PositiveInfinity;

                    foreach (IExecutionContext candidate in candidates)
                    {
                        double thisValue =
_defaultLVFProperty.GetDoubleValue(candidate);
                        if (thisValue < sSmallestValue)
                        {
                            smallestValue = thisValue;
                            selectedCandidate = candidate;
                        }
                    }
                    chosenOne = selectedCandidate;
                    break;

                default:
                    // Loop through the collection of candidates to return a
selected item based on the default smallest value first selection expression
                    double DefSmallestValue = double.PositiveInfinity;

                    foreach (IExecutionContext candidate in candidates)
                    {
                        double thisValue =
_defaultSVFProperty.GetDoubleValue(candidate);
                        if (thisValue < DefSmallestValue)
                        {
                            DefSmallestValue = thisValue;
                            selectedCandidate = candidate;
                        }
                    }
                    chosenOne = selectedCandidate;
                    break;
            }

            return chosenOne;
        }

        #endregion
    }
}
```

Appendix G Distributed Dynamic Selection Rule Strategy Optimisation Experiment Results

Table G.1. Distributed Dynamic Selection Rule Strategy optimisation experiment results. LeadTime units = hrs.

Scenario	Response	Median	Minimum	Maximum	Upper Percentile	Lower Percentile
1	NumberInSystem	47.18991	21.31053	121.0012	74.53899	34.16343
1	LeadTime	6.29372	2.89953	16.2276	9.927454	4.59824
2	NumberInSystem	50.27236	23.91717	152.1539	78.66498	33.66523
2	LeadTime	6.709231	3.252308	19.96946	10.44169	4.492619
3	NumberInSystem	54.75185	25.14768	123.7043	68.91233	38.37558
3	LeadTime	7.359213	3.410178	16.40822	9.177281	5.112283
4	NumberInSystem	58.20055	23.2403	139.8145	82.01304	38.31949
4	LeadTime	7.674123	3.172868	18.41095	10.90507	5.154579
5	NumberInSystem	55.73427	22.96691	155.1809	82.66219	40.02628
5	LeadTime	7.497569	3.134169	20.49732	10.96186	5.356662
6	NumberInSystem	48.57383	27.32445	121.2572	70.09397	38.3977
6	LeadTime	6.498349	3.690238	15.71159	9.326045	5.132793
7	NumberInSystem	50.80224	24.74704	113.8098	66.43451	36.22615
7	LeadTime	6.676455	3.404988	14.93614	8.874533	4.891784
8	NumberInSystem	54.44587	22.31514	126.823	75.99836	38.30738
8	LeadTime	7.330716	3.096369	16.41343	10.15205	5.145764
9	NumberInSystem	50.61299	24.09376	141.8291	70.61683	37.88078
9	LeadTime	6.778188	3.318568	18.59123	9.342828	5.074921
10	NumberInSystem	41.22279	22.23416	138.5455	56.91087	34.19384
10	LeadTime	5.561042	3.056578	18.29718	7.527437	4.625499

Scenario	Response	Median	Minimum	Maximum	Upper Percentile	Lower Percentile
11	NumberInSystem	47.34403	26.01876	129.9241	74.51056	41.48138
11	LeadTime	6.332349	3.537553	17.07746	9.947056	5.554419
12	NumberInSystem	51.26031	24.42605	161.0056	74.02337	40.08207
12	LeadTime	6.844293	3.325869	21.16594	9.848239	5.383057
13	NumberInSystem	46.27778	24.5841	146.4795	70.25798	35.85971
13	LeadTime	6.135549	3.361308	19.18207	9.283131	4.852237
14	NumberInSystem	49.7201	25.68204	165.7341	66.32365	38.73042
14	LeadTime	6.643765	3.511741	21.57286	8.892695	5.195665
15	NumberInSystem	46.77977	23.80841	136.0243	67.01679	35.68487
15	LeadTime	6.229515	3.234449	17.92464	8.674399	4.794589
16	NumberInSystem	51.30725	23.22187	109.0368	60.23216	32.40555
16	LeadTime	6.841738	3.172815	14.27803	7.942081	4.370878
17	NumberInSystem	55.56263	24.18633	145.9068	71.50771	39.26129
17	LeadTime	7.425856	3.301503	19.25416	9.371295	5.290711
18	NumberInSystem	47.0257	20.92081	160.7138	70.36309	34.7767
18	LeadTime	6.295165	2.878408	21.08952	9.413554	4.72285
19	NumberInSystem	50.29853	27.10589	111.7005	63.67224	33.37516
19	LeadTime	6.651331	3.67677	14.36949	8.481788	4.486457
20	NumberInSystem	55.27635	22.54477	153.658	69.43934	38.35355
20	LeadTime	7.410742	3.112063	19.86891	9.227191	5.176797
21	NumberInSystem	51.18463	22.35621	108.7889	68.72326	41.16838
21	LeadTime	6.857968	3.051531	14.32009	9.099076	5.484205
22	NumberInSystem	49.59716	25.16082	128.7469	78.54713	37.8368
22	LeadTime	6.604927	3.418297	17.06986	10.30939	5.064498

Scenario	Response	Median	Minimum	Maximum	Upper Percentile	Lower Percentile
23	NumberInSystem	53.1557	26.49573	133.586	63.70927	40.16259
23	LeadTime	7.131976	3.639055	17.40354	8.502993	5.435992
24	NumberInSystem	59.38887	23.87142	145.5534	74.33021	42.6279
24	LeadTime	7.787031	3.217559	19.13769	10.05104	5.69356
25	NumberInSystem	40.74247	23.53191	122.7473	61.95701	31.89353
25	LeadTime	5.514123	3.199266	16.20775	8.054602	4.312269
26	NumberInSystem	46.68711	28.1486	130.71	71.31524	37.77022
26	LeadTime	6.198108	3.847671	17.11797	9.466911	5.11679
27	NumberInSystem	45.45204	19.58775	157.1321	59.72797	33.53636
27	LeadTime	6.145037	2.683171	20.47421	7.961891	4.50375
28	NumberInSystem	39.62775	26.55756	163.8102	62.40869	33.86871
28	LeadTime	5.254407	3.634719	21.30016	8.31608	4.52882
29	NumberInSystem	58.49901	26.96042	153.4589	90.19759	35.9293
29	LeadTime	7.84506	3.64614	20.09118	11.89784	4.819732
30	NumberInSystem	43.24588	22.78444	153.9226	63.51044	36.88661
30	LeadTime	5.820793	3.109192	20.09415	8.389242	4.963503
31	NumberInSystem	51.17099	21.80235	80.90089	60.06092	44.5704
31	LeadTime	6.844094	2.989218	10.70017	7.921221	5.896014
32	NumberInSystem	54.70226	24.38795	155.4323	65.86062	40.3191
32	LeadTime	7.290388	3.345899	20.71114	8.700618	5.374443
33	NumberInSystem	49.15934	21.62882	176.7565	68.256	35.81773
33	LeadTime	6.615494	2.98789	23.22103	9.108965	4.862683
34	NumberInSystem	47.39068	23.14294	147.4517	61.32331	36.36838
34	LeadTime	6.247023	3.164799	19.0647	8.11325	4.929978

Scenario	Response	Median	Minimum	Maximum	Upper Percentile	Lower Percentile
35	NumberInSystem	51.01886	24.65303	121.2287	63.08266	40.08395
35	LeadTime	6.69503	3.380405	15.91755	8.396881	5.396053
36	NumberInSystem	55.69429	22.72155	132.1527	83.27704	37.69361
36	LeadTime	7.414248	3.138575	17.22436	11.10432	5.044559
37	NumberInSystem	51.71128	22.16124	142.5349	76.03009	35.79094
37	LeadTime	6.886525	3.023127	18.65926	10.11219	4.770644
38	NumberInSystem	50.94247	23.37602	131.7421	69.12074	41.08423
38	LeadTime	6.832276	3.190964	17.17971	9.206805	5.545241
39	NumberInSystem	47.52075	22.71472	167.7911	70.94806	35.27117
39	LeadTime	6.357209	3.102342	21.8092	9.465521	4.737906
40	NumberInSystem	45.88449	21.76593	127.5372	64.52064	32.38853
40	LeadTime	6.197203	3.016135	16.84122	8.665391	4.414061
41	NumberInSystem	51.76807	24.36402	195.1644	68.96151	37.93788
41	LeadTime	6.889451	3.304107	25.09042	9.092541	5.109981
42	NumberInSystem	49.61962	25.35996	125.8791	67.39074	39.1831
42	LeadTime	6.699624	3.435265	16.73603	8.932051	5.258824
43	NumberInSystem	55.64225	20.38029	157.1234	83.52963	35.66761
43	LeadTime	7.528106	2.822227	20.74808	11.10631	4.807453
44	NumberInSystem	51.75636	26.22645	103.4306	69.24423	36.20173
44	LeadTime	6.875591	3.566295	13.86356	9.042078	4.884017
45	NumberInSystem	49.83682	23.73026	168.586	83.73004	38.73735
45	LeadTime	6.634423	3.261743	21.99095	11.14509	5.172153
46	NumberInSystem	46.94089	23.55925	181.0869	64.68387	36.94655
46	LeadTime	6.304775	3.251439	23.54808	8.542337	4.985058

Scenario	Response	Median	Minimum	Maximum	Upper Percentile	Lower Percentile
47	NumberInSystem	49.59131	22.58498	116.7295	74.6539	37.33568
47	LeadTime	6.590515	3.105406	15.40892	9.831556	5.033078
48	NumberInSystem	50.5726	21.47285	125.8605	70.70321	38.86532
48	LeadTime	6.821519	2.939371	16.78228	9.423498	5.169138
49	NumberInSystem	56.74241	22.47137	129.5858	77.15819	40.45717
49	LeadTime	7.54213	3.07241	16.92936	10.23705	5.457119
50	NumberInSystem	55.88337	21.76831	150.8936	77.27105	38.73319
50	LeadTime	7.38288	2.981769	19.92786	10.19431	5.199516
51	NumberInSystem	42.50559	26.23452	117.511	54.91071	36.01094
51	LeadTime	5.634367	3.567032	15.24702	7.390152	4.830545
52	NumberInSystem	46.0718	28.55602	138.344	61.35481	36.4212
52	LeadTime	6.331372	3.884544	18.17902	8.088633	4.833361
53	NumberInSystem	56.6866	26.06138	138.2852	70.27983	42.14286
53	LeadTime	7.583137	3.592365	17.91597	9.333012	5.652745
54	NumberInSystem	46.07066	24.74544	140.8155	64.7524	36.221
54	LeadTime	6.115034	3.358889	18.69009	8.650683	4.871072
55	NumberInSystem	47.24336	23.00691	144.271	64.53615	35.36985
55	LeadTime	6.337853	3.153092	18.96297	8.670139	4.849531
56	NumberInSystem	51.61059	25.57414	156.6455	78.06199	39.77042
56	LeadTime	6.866954	3.459372	20.47694	10.54573	5.396318
57	NumberInSystem	54.29631	21.41359	115.536	73.64237	38.0786
57	LeadTime	7.238358	2.973653	15.40933	9.731887	5.142306
58	NumberInSystem	52.22529	22.19149	129.1369	75.00442	37.1515
58	LeadTime	6.991848	3.065159	17.07078	9.817184	5.017483

Scenario	Response	Median	Minimum	Maximum	Upper Percentile	Lower Percentile
59	NumberInSystem	51.80942	29.38868	147.1306	70.44429	37.13466
59	LeadTime	7.044588	4.017319	19.44387	9.403573	5.005646
60	NumberInSystem	51.46061	22.13682	113.3465	65.89276	35.371
60	LeadTime	6.838271	3.030043	14.8691	8.717653	4.762731
61	NumberInSystem	46.54967	23.1368	126.5407	64.26127	35.30211
61	LeadTime	6.22423	3.168098	16.74356	8.426912	4.776001
62	NumberInSystem	50.0509	20.96385	133.1087	59.88506	35.59922
62	LeadTime	6.72859	2.904983	17.4264	8.03952	4.820938
63	NumberInSystem	53.0657	23.50411	156.1355	80.491	39.61521
63	LeadTime	7.094587	3.209328	20.6677	10.52726	5.326159
64	NumberInSystem	53.68862	24.51739	136.4015	72.94775	35.14471
64	LeadTime	7.23141	3.324675	17.92989	9.725597	4.710847
65	NumberInSystem	49.0728	21.47405	115.5303	76.08948	37.71709
65	LeadTime	6.591992	2.939659	15.10392	10.07022	5.059545
66	NumberInSystem	49.24974	21.7215	146.7255	68.13967	34.10182
66	LeadTime	6.642193	2.982741	19.39939	9.052289	4.585093
67	NumberInSystem	60.27058	25.80516	145.0079	71.25009	41.18985
67	LeadTime	8.04442	3.512146	19.02574	9.528668	5.510429
68	NumberInSystem	56.59973	25.67896	132.9707	74.64153	38.49782
68	LeadTime	7.477789	3.548682	17.57625	9.911588	5.104295
69	NumberInSystem	57.38819	23.03283	141.5505	79.68722	34.31647
69	LeadTime	7.747871	3.15346	18.45057	10.60386	4.634214
70	NumberInSystem	55.03563	23.82917	140.7608	66.7805	41.25188
70	LeadTime	7.373092	3.21642	18.65511	8.847848	5.60046

Scenario	Response	Median	Minimum	Maximum	Upper Percentile	Lower Percentile
71	NumberInSystem	57.03823	26.80359	172.6101	76.22587	40.4898
71	LeadTime	7.5742	3.620135	22.6813	10.16855	5.408243
72	NumberInSystem	56.85529	24.00069	165.7555	75.14633	39.41744
72	LeadTime	7.60384	3.277043	21.67721	10.02291	5.266946
73	NumberInSystem	47.86218	25.57504	142.2329	66.06162	37.95719
73	LeadTime	6.431094	3.489787	18.79926	8.735243	5.132715
74	NumberInSystem	53.60249	26.53314	128.0005	72.61116	38.88171
74	LeadTime	7.15462	3.593846	17.06175	9.418079	5.214151
75	NumberInSystem	50.83562	22.92324	127.5411	67.31444	39.29965
75	LeadTime	6.833163	3.132364	16.92362	8.921906	5.272304
76	NumberInSystem	50.80926	21.40588	136.865	63.54177	36.86178
76	LeadTime	6.842555	2.962799	18.13577	8.44064	4.904589
77	NumberInSystem	51.35332	26.42601	115.8446	65.9796	39.44247
77	LeadTime	6.881619	3.556083	15.24344	8.76941	5.327706
78	NumberInSystem	51.62118	23.70787	132.7135	75.55609	35.4873
78	LeadTime	6.892295	3.230559	17.33787	10.02427	4.754339
79	NumberInSystem	52.42079	24.08544	148.0176	73.0807	43.5367
79	LeadTime	6.98597	3.304319	19.35214	9.645512	5.857012
80	NumberInSystem	46.13544	26.72977	159.8392	66.44176	37.18213
80	LeadTime	6.253367	3.614063	20.85885	8.86457	5.017627
81	NumberInSystem	49.74545	25.53109	97.5309	65.13961	38.39926
81	LeadTime	6.67168	3.468238	12.93976	8.655596	5.163665
82	NumberInSystem	47.52799	25.10375	133.6586	64.93294	35.07953
82	LeadTime	6.386401	3.39708	17.10072	8.63449	4.741451

Scenario	Response	Median	Minimum	Maximum	Upper Percentile	Lower Percentile
83	NumberInSystem	53.87671	26.90604	139.2993	70.72354	39.93253
83	LeadTime	7.175675	3.647958	18.3331	9.393401	5.399508
84	NumberInSystem	52.38632	26.56267	168.2412	75.04346	40.02762
84	LeadTime	6.966034	3.63807	22.17071	9.974429	5.313158
85	NumberInSystem	43.31438	21.61951	127.6251	64.87861	34.94866
85	LeadTime	5.785254	2.970497	16.71975	8.724349	4.706744
86	NumberInSystem	45.90316	24.64828	157.7384	73.13046	38.66895
86	LeadTime	6.20665	3.37313	20.73357	9.772339	5.226328
87	NumberInSystem	52.59038	19.23175	166.3072	76.8966	38.90971
87	LeadTime	7.036777	2.671512	21.79227	10.28741	5.238594
88	NumberInSystem	56.49366	28.61662	127.1641	79.39168	40.26774
88	LeadTime	7.487588	3.826007	16.7035	10.49356	5.416465
89	NumberInSystem	49.34843	23.2231	120.7642	67.36568	36.64799
89	LeadTime	6.611088	3.173384	16.00249	9.005435	4.916793
90	NumberInSystem	48.97966	27.54395	120.0883	69.94155	37.59747
90	LeadTime	6.512474	3.726392	15.88287	9.24579	5.063139
91	NumberInSystem	53.26292	22.86136	170.8596	80.79378	37.1874
91	LeadTime	7.116175	3.088866	22.37501	10.72585	5.010111
92	NumberInSystem	46.14657	24.57777	156.3478	68.57821	35.33321
92	LeadTime	6.206775	3.35353	20.27477	9.23722	4.810778
93	NumberInSystem	49.81492	25.25958	122.586	72.83679	37.75356
93	LeadTime	6.607935	3.462826	16.2772	9.537742	5.072711
94	NumberInSystem	53.59821	27.30757	157.7579	72.07915	43.06823
94	LeadTime	7.197414	3.716516	20.63436	9.614192	5.725697

Scenario	Response	Median	Minimum	Maximum	Upper Percentile	Lower Percentile
95	NumberInSystem	45.42331	24.43446	149.3866	67.16217	33.88044
95	LeadTime	6.122967	3.340827	19.48842	8.880972	4.570091
96	NumberInSystem	49.24928	25.39032	141.757	81.33433	37.31089
96	LeadTime	6.610146	3.458988	18.47674	10.77473	5.037488
97	NumberInSystem	58.12781	25.68783	142.3675	71.39635	38.84109
97	LeadTime	7.706039	3.502756	18.93661	9.482129	5.258362
98	NumberInSystem	53.02677	23.31307	146.0057	82.84325	37.97752
98	LeadTime	7.120324	3.223005	19.20606	10.97294	5.091405
99	NumberInSystem	45.36269	24.3285	115.8269	62.07102	35.39901
99	LeadTime	6.00544	3.339075	14.90113	8.151613	4.793197
100	NumberInSystem	54.25115	22.68398	125.7172	78.0072	38.59037
100	LeadTime	7.138667	3.113945	16.38166	10.27718	5.153447