Solving the buffer allocation problem using simulation-based optimisation

Dirk Jacobus Kotze

A dissertation in partial fulfilment of the requirements for the degree

Magister in Engineering (Industrial Engineering)

in the

Faculty of Engineering, Built Environment, and Information Technology

University of Pretoria

February 2020

Abstract

Title:	Solving the buffer allocation problem using simulation-based optimisation.
Student name:	Dirk Jacobus Kotze
Student number:	u10014642
Supervisor:	Prof. Johan W. Joubert

This dissertation solves the Buffer allocation problem (BAP), using Simulation-based optimisation (SBO). Chapter 1 describes the role buffers play within a production line. Buffer is allocated to decouple stations and reduce the effect station failure and differing process times have on the complete line throughput. Adding buffer to the line increases the working capital required to run the line, thus the optimal number of buffer to add to a line is an important design question. The problem statement is introduced. The research design is discussed in this Chapter. A Discrete event simulation (DES) is specifically designed to simulate a production line for the BAP. An inner and outer loop is used to generate various buffer configurations. The Adaptive tabu search (ATS) for the inner loop is modified by exploiting theory of constraints. The ideology states that the overall performance of a line can be improved by ensuring the bottleneck is never starved nor blocked. The proposed Theory of constraints tabu search (TOCT) uses the block or starved ratio of stations to generate neighbours, reducing the number of neighbours that need to be evaluated per iteration. Small-sized, medium and large-seized lines with various process time, failure rate and repair time is used to scrutinise the proposed method, detailed computation results are shown.

In Chapter 2 a literature study is done on approaches for solving the BAP. Three main objectives are considered when solving the BAP. This dissertation focuses on objective two, a prescribed throughput must be achieved with the minimum total buffer size. Various search methods, meta-heuristics and complete enumeration is discussed as an option for the evaluative method. The works of Demir et al. [9, 11] show that Tabu search (TS) is an effective approach as a generative method for the BAP. Analytical methods and simulation is discussed as evaluative methods. The decomposition method is most common in literature. The disadvantage is that the decomposition method has limitations on the line topology it can solve as well as distribution types for the machine parameters. Alternatively, simulation or meta-models have been employed to represent more complex lines. Simulation has a more accurate representation of the production line than a meta-model but takes longer to compute. To overcome this DES can be used to create a specific program.

In Chapter 3 a specific solution approach for the BAP is developed. The use of DES to design a simulation in Java specifically for the BAP problem addresses the computational burden of generic simulation models. Two event types are created for the model, departure

and failure events. The works of Demir et al. [9, 11] establish a solid foundation for the use of tabu search in a two-loop manner to solve the BAP.

In Chapter 4 the proposed method for creating and simulating serial production lines using the program designed in Java is comparable to a commercial program and is faster. Initial tests using DES with the ATS inner loop proved that SBO is capable of solving the BAP. However, long execution time makes it unusable for medium and large-sized problems. Two alterations are considered for the ATS and a new inner tabu loop is proposed based on theory of constraints, TOCT as well as a list that saves previously tested buffer scenarios and their throughput so that same scenarios are not re-evaluated with the simulation model. The proposed method is 18 times faster than ATS for small-sized problems and 5.5 times for medium-sized problems. The proposed method is used to solve the BAP for objective two, showing that SBO is an effective model.

Finally, in Chapter 5, the solution approach scalability is tested on an actual, complex automotive production line.

Acknowledgements

The author acknowledge the Centre for High Performance Computing (CHPC) for the computing resources that made it possible to conduct extensive SBO tests on the *Lengau* cluster.

Contents

List of Figures

List of Tables

List of acronyms

ALBP	Assembly line balancing problem
ALS	The assembly line simulator
ATS	Adaptive tabu search
BAP	Buffer allocation problem
BIW	Body-in-white
CE	Complete enumeration
CHPC	Centre for high performance computing
CN	Complete neighbour generation
DES	Discrete event simulation
FWH	Front wheel house
MTTF	Mean time to failure
SBO	Simulation-based optimisation
SSJ	Stochastic simulation in java
STS	Standard tabu search
TOCT	Theory of constraints tabu search
TS	Tabu search
TT	Tabu tenure
WIP	Work-in-progress

Chapter 1

Introduction

The automobile industry is a global economic growth driver. The industry made over 66 million cars, vans and trucks globally in 2005 [22]. The level of output is equivalent to a global turnover of almost €2 trillion worldwide. The industry continues to grow, registering a 30% increase from 1995–2005. Building 60 million vehicles requires the employment of 9 million people; those involved in making the vehicles directly as well as those required to manufacture the parts that go into them. This is over 5% of the worlds total manufacturing employment. For every direct employee, it is estimated that 5 indirect employees are involved. The industry is a major innovator, investing over €84 billion in research, development, and production. Achieving efficient utilisation of the investment is crucial to the success of the industry.

1.1 Problem background

Vehicle production is done in high volumes, with individual factories capable of producing half a million units per year. Adding to the complexity, single plants can be required to produce different models. To achieve these car manufacturers have established factories consisting of multiple facilities to fit the various components.

Sheet metal is used for the body of the car. These are pressed in *press plants* which can either form part of the factory or be outsourced to external suppliers. The press parts are welded together in the *body shop*. Highly automated lines construct the body of the car. Step by step, new press parts are added to it as it moves through the line. The end product is a steel body known as the Body-in-white (BIW). After the BIW is built, it goes to the *paint shop* where multiple layers of protective coating and paint are applied to the car. Finally, the car moves to the *assembly line*. Here workers install the various parts to the body. The end product from the assembly line is a completed car that drives off the line. Each of these facilities uses a production line to achieve the mass production needed.

A production line is a system in which a series of value-adding work stations are connected with material moving equipment. Value is added to the item as it moves through the line [11]. At each step, parts are assembled.

Each line in the automotive factory has unique characteristics based on production requirements. Body shops are highly automated and consist of multiple smaller lines (machine cells). These lines build the sub-assemblies, which then flow into the mainline. Sub-assemblies of a BIW is shown in Figure 1.1. A typical body shop production line is shown in Figure 1.2a, this line layout can be classified as a *tree structure line*. The line consists of a number of machines cells, labelled as M_k and separated by a buffer B_{k-1}

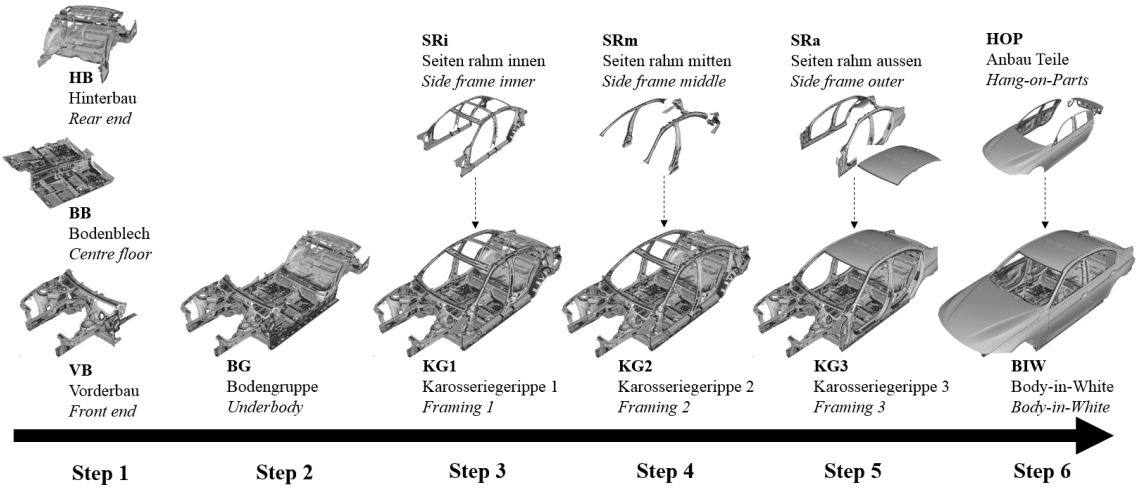


Figure 1.1: BIW assembly components [14].

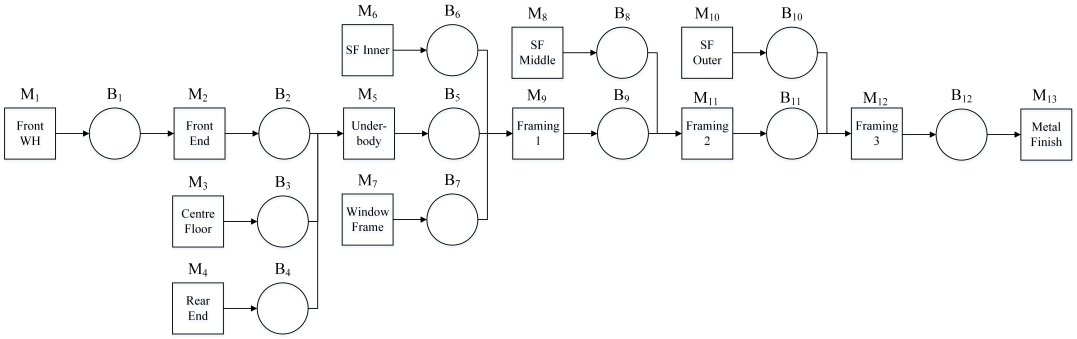
where $k \in \boldsymbol{K} = \{1, 2, ..., K\}$, $\mathbf{K}$ is the number of machine cells. The various machine cells of a body shop, shown in Figure 1.2a contain smaller machine cells each with body shop equipment such as welding robots, grippers and fixtures.

Compared to the body shop, the assembly line is a labour intensive facility where the trim components of the car are assembled. The car moves down a serial line (see Figure 1.2b) and at each station, workers assemble a specific set of parts on the car until a completed car drives off the line.

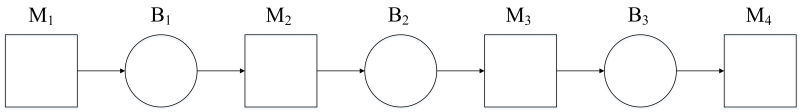
Figure 1.3 shows the content of the *Front wheel house (FWH)*, station M_1 of Figure 1.2a. The *FWH* can easily be broken down into 10 smaller machine cells and 14 buffers. It is even possible to expand station $M_{1.3}$ and $M_{1.6}$ into more, smaller stations as they also consist out of several fixtures, robots and buffers. Stations M_{1-13} in Figure 1.2a can all be expanded in a similar manner. Hence, the body shop production line is seen as a complex system.

During the design of the body shop, specific cycle time is calculated for the line. The cycle time is determined by the throughput the production line needs to achieve. Each machine cell is then developed to have a processing time as close to this desired cycle time as possible. The processing time is how long it takes the cell from receiving the parts to finishing the assembly. The aim is to never have a processing time that exceeds the cycle time. Due to space, weld locations and robot movement constraints, only certain work content can be included in each cell. Even if the cell's processing time is less than the required cycle time, it is not possible to add more content to the cell due to these constraints. This limitation makes it impossible to have the same processing time for all the cells. In a scenario where the processing time of a station is longer than those before and those after, all stations upstream of it will be *blocked*, while all downstream stations' waiting time to receive parts will increase. For the downstream stations, we say they are *starved* as no parts are arriving to feed them. Therefore the station with the longest processing time will be the bottleneck in the line.

Another factor that can influence stations becoming blocked or starved is equipment failure. The body shop equipment such as robots, grippers and fixtures are all subject to failure. Due to a large number of equipment in the body shop, it is extremely sensitive to machine failure. Figure 1.4 is used to illustrate the impact of machine failure on the total availability of a simple serial production line with five machines, each with an equipment availability of 95%. If we assume each machine fails independently, the total availability of



(a) Tree structure line typically found in body shop.



(b) Serial line typically found in assembly.

Figure 1.2: Production line topology.

the five-station line can be calculated as $95\% \times 95\% \times 95\% \times 95\% \times 95\% = (95\%)^5 = 77.37\%$. If the total assembly system stopped every time a single station failed the line would rarely function [1]. Once a station fails, it can cause all upstream stations to become blocked and downstream stations starved if the station cannot be repaired before the end of the cycle.

The impact of these factors on the throughput of the line can be reduced by introducing buffers into the production line. Buffers are temporary storage areas strategically placed in the production line to decouple segments of the line. If a station that has a buffer for its exit material fails, all downstream stations can remain operational while there are parts in their respective buffers. If the station cannot be repaired before the buffer empties, and start supplying downstream, the production of the downstream station will stop. If there is a buffer before the failed station, all upstream stations will be able to produce while there is space in the buffer. Again if the station cannot be repaired before the buffer reaches capacity, and start pulling parts, the upstream line will become blocked. The bigger the buffer, the longer the production line can run independently. This reduces the effect line reliability has on production throughput.

Allocating buffers into a production line requires additional capital, material handling and is limited by available floor space. Buffers also increase Work-in-progress (WIP) inventory. The strategic placement of buffers in the production line is, therefore, an important manufacturing design problem.

The problem of allocating buffers optimally is known as the Buffer allocation problem (BAP). BAP consists of finding the optimal buffer configuration (location and size) within the production line. For a comprehensive study of the BAP see the works of Demir et al. [10].

The BAP has three main classifications of problem objectives: maximise throughput for a given fixed number of buffer, achieve desired throughput with minimum buffer, and minimisation of the WIP subject to the specified buffer size.

The BAP is also classified based on line parameters. A production line where the processing time of all stations are similar is called a *homogeneous* production line. Conversely,

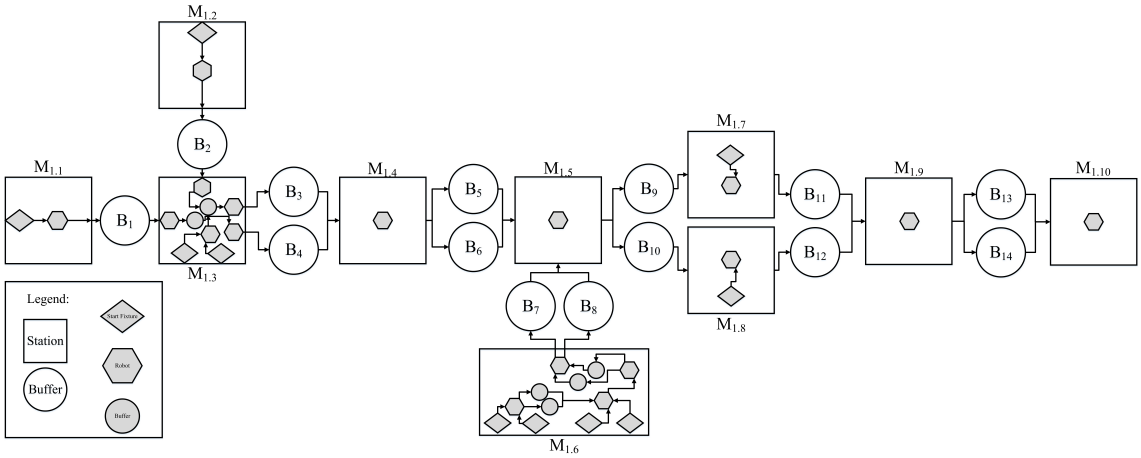


Figure 1.3: M1 FWH station explosion.

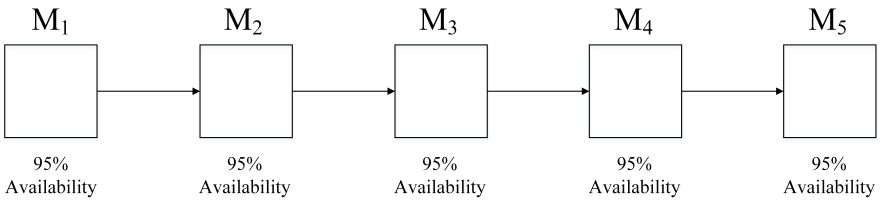


Figure 1.4: Serial production line availability.

if the process times are different, it is known as a *heterogeneous* line [29]. In some cases, literature ignores the possibility that failures may occur, or that failures are negligible, in which case the line is assumed to be *reliable*. However, when failures are explicitly addressed in the model, the line is referred to as being *unreliable* [10].

BAP has been proven to be an NP-Hard combinatorial optimisation problem [10]. That means when the number of machines in the line is increased, or the number of buffers that can be allocated to the line increases, the number of possible solutions to the problem grows exponentially. For example, a small-sized problem—say a production line with five machines, four buffer locations and a total buffer size of 25—has 3 279 possible configurations. Increasing the total buffer size to 100 (factor 4) increases the number of configurations to 176 854 (factor 54). Production lines can have more than 40 machines in the line and a total buffer size of 800.

A complex system such as a production line cannot be expressed by a simple mathematical equation where the throughput of the line can be calculated for a given buffer configuration [10]. Due to the lack of this algebraic relation and the problem being NP-Hard, a method with two elements are frequently used to solve the BAP [10].

Evaluative method: The evaluative method is used to calculate the throughput of a production line for a given buffer configuration. Various versions of the evaluative method have been employed in the literature.

Analytical methods have been used for the BAP. Exact analytical methods are only applicable to small-sized problems, not for production lines as big as the body shop. For larger systems, approximation methods can be used.

Methods employing algebra, calculus or probability theory are used to approximate the throughput of the line. A method known as the *decomposition* method is widely

used in BAP as an evaluative method. In the works of Demir et al. [9], the decomposition method is used to determine the throughput of a serial line. The decomposition method transforms the production line into a set of $K-1$ smaller two-machine lines $L(i)$, for $i = 1, ..., K-1$, as illustrated in Figure 1.5. Line $L(i)$ has an upstream machine and a downstream machine separated by a buffer. The average throughput of the production line can be obtained by modelling the line as a Markov process. The performance characteristics of a two-machine line can be obtained exactly, the decomposition method requires a derivation of a set of equations that link the decomposed two-machine lines together. The non-linear equations are solved to determine parameters of the upstream and downstream machine such as process time, failure rate and repair rate to that the behaviour of material flow into buffer $B_{(i)}$ closely matches that of the flow into buffer B_i of the original line L. The literature found on the BAP using the decomposition method is limited to serial production lines. Recall from Figure 1.2a, the topology of the body shop line is a tree structure where multiple buffers lead into a machine, and the machine can only produce when a component from each buffer is present. Due to the structure of the body shop, it cannot be decomposed in the same way as a serial line.

Alternatively, *simulation* is used to determine line throughput. Simulation is the imitation of a system being studied which is performed on computers by creating a model of it. Various line topologies can be modelled, including tree structure lines. Each station can have different random variables describing the processing times. Failure of stations can also be included in the model. The disadvantage of simulation is the time it takes to evaluate a scenario.

Generative method The generative method moves through the solution space and considers the various buffer configurations to be evaluated Demir et al. [10]. The generative method aims to find a (near) optimum buffer configuration in the shortest time possible. Methods employed include *complete enumeration*, *traditional search*, *heuristic search*, and *metaheuristics*. Complete enumeration evaluates all possible configurations and is only applicable to small problems. Traditional search methods and heuristics test only a subset of the buffer configurations. The disadvantage is that they get stuck at a local optimum. Metaheuristics add logic to the way it moves through the solution space and has the main advantage that they can escape local optimum.

The evaluative and generative methods are executed iteratively. The methods start with an initial buffer configuration. With the given buffer configuration, the evaluative method can determine the throughput of the line. Because the initial configuration might not be the optimal solution, the generative method generates new buffer configurations for the evaluative method to test. This is repeated until a (near) optimal solution is found.

Demir et al. [10] conducted a comprehensive survey of the BAP. It highlights the new trends in the field and presents ideas for future research by identifying gaps. Within the paper, 95 literature works on the BAP were studied. The studied papers are classified based on line characteristics as well as the solution approaches used. The study showed that the majority of literature did not ignore failure, and included it in the model. Only 41 papers ignored failure in their solution approach.

Out of the 54 papers that did include failure, the majority of the literature focus on simpler production lines. Serial or serial-parallel production lines were studied in 40 of the 54 papers, with the rest solving the BAP for more complex line topologies. These simpler lines are solved using approximation methods as the evaluative method. Only 14 papers

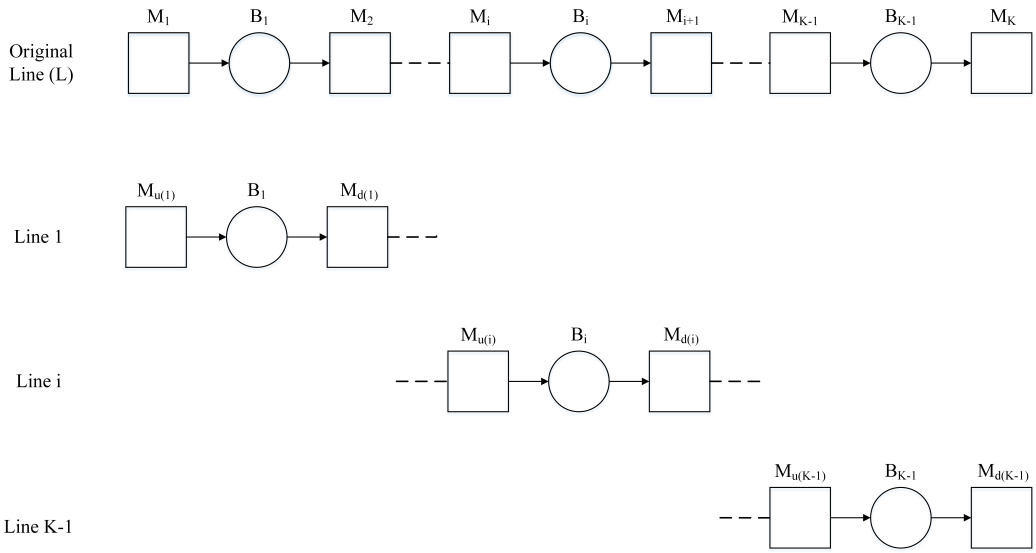


Figure 1.5: Serial line decomposition method.

conducted studies on complex lines. Out of these 14 papers, 10 of them used simulation as the evaluative method.

The complex stochastic nature of the body shop line cannot be accurately solved with approximation methods. Thus simulation needs to be used as an evaluative method. Simulation can be used to evaluate the throughput of the line for every scenario given by the generative method. The use of simulation iteratively with an optimisation technique is known in the literature as *Simulation-based optimisation (SBO)*. This dissertation aims to increase the body of literature on SBO as method to solve BAP.

1.2 Problem statement

The problem under investigation is how to determine the buffer configuration for complex production lines that require simulation to accurately represent the production line. The solution should be scalable for a system such as the body shop. The aim is to achieve its required throughput while minimising the number of buffers included in the line. To reduce the effect of equipment failure, buffers are included in the line at a great financial cost. The body shop is a complex manufacturing facility in an automotive factory as it has a non-serial, heterogeneous and unreliable production line. Appropriate methods need to be used to evaluate this complex line's throughput and generate good buffer configurations for evaluation.

For a complex system such as the body shop, analytical and approximation methods cannot be used. Simulation is capable of modelling the stochastic nature of the line and provide highly accurate throughput times for a given buffer configuration. To find the *best* solution, multiple buffer configurations need to be generated by an optimisation method and evaluated with simulation.

Thus, the problem that needs to be solved is how the method of SBO can be applied to the BAP to achieve the *best* solution. *Best* referring to finding a buffer configuration that has the smallest total buffer size while still achieving a specific throughput, within an acceptable time frame.

1.3 Research design

The dissertation solves the BAP using SBO. The iterative use of a generative method and evaluative method is employed to solve the BAP with the aim to find the minimum buffer size required to achieve a required throughput. A Discrete event simulation (DES) is specifically designed to simulate a production line for the BAP. Stochastic simulation in java (SSJ) is used to create a general framework that receives the number of machines with corresponding machine parameters as input and creates the simulation model. The design of the simulation model is focused specifically for BAP. It is tested against commercial software to prove the efficiency of the model. The generative method is based on the works of [9, 11]. An inner and outer loop is used to generate various buffer configurations. The Adaptive tabu search (ATS) for the inner loop is modified by exploiting theory of constraints. The ideology states that the overall performance of a line can be improved by ensuring the bottleneck is never starved nor blocked. The proposed Theory of constraints tabu search (TOCT) uses the block or starved ratio of stations to generate neighbours, reducing the number of neighbours that need to be evaluated per iteration. Small-sized, medium and large-seized lines with various process time, failure rate and repair time is used to scrutinize the proposed method, detailed computation results are shown. Finally the proposed methods scalability is tested on an industry problem, solving the BAP on a non-serial, large production line.

1.4 Research methodology

The methodology of design research discussed in Manson [20] is used in the paper. This methodology describes guidelines for a researcher who wishes to conduct a paper based on the field of operations research.

To solve the problem described in the problem statement, first our understanding of the BAP needs to be enhanced. In Chapter 2 a comprehensive literature review on the BAP is done. The method of using an evaluative and generative element iteratively to solve the BAP is studied. Analytical evaluative methods are compared with simulation evaluative methods and justification is provided for the use of simulation as the evaluative method of choice in this dissertation. For the generative method, various approaches are researched. Chapter 2 suggests why the use of SBO is the most applicable method to solve the BAP for the body shop line.

This idea of SBO is then developed as a solution approaches in Chapter 3, specifically for the BAP. A simulation program is created in a general programming language, Java. This program is a newly designed blueprint using the library of the SSJ. It is possible to simulate any size of a serial production line by just specifying the number of machines, random parameters, replication length and simulation length as well as the buffer vector. For the generative method, two metaheuristics are compared for finding the optimal buffer configuration for maximising throughput. The first method is based on the works of Demir et al. [9]. The second method is a proposed improvement on the above mentioned algorithm to improve the evaluation time.

In Chapter 4 this artefact is used and tested on serial production lines. First, the validity of the simulation model is checked. Then the two generative methods are compared across various production line lengths and random parameters. The best, that is the method that achieves a near-optimal solution within a reasonable amount of time, will be used with the simulation model to solve the larger and more realistic BAP.

Lastly, the proposed methodology is applied to a body shop production line. The use

of SBO in the BAP will help to increase our knowledge on its effectiveness with solving the BAP for complex lines such as the body shop.

1.5 Contribution

The dissertation presents a DES model that is not a computational burden. A TOCT is presented where theory of constraints ideology that the overall performance of a line can be improved by ensuring the bottleneck is never starved or blocked. The neighbour generation prioritise buffer allocation to stations that are mostly blocked and remove buffer from stations that are rarely starved.

Chapter 2

Literature review

Buffers are locations in which semi-completed, Work-in-progress (WIP), parts are stored within a production line. Factors such as uneven processing time and station failure create instability in the production line, which has a negative effect on throughput. Buffers are placed in the line to decouple stations and reduce instability. Buffers allow preceding and succeeding stations to operate independently.

By including buffers, the required capital to realise and operate the production line is increased. WIP is also increased, leading to higher running capital. Minimising capital investment, by reducing the number of buffers while providing sufficient buffer to reduce the effect of equipment instability on production availability, makes the optimal placement of buffers in the line a vital problem to solve.

The Buffer allocation problem (BAP) deals with finding the optimal buffer configuration to incorporate in the production line to achieve a specific objective [10]. Due to the complexity of the BAP, numerous publications are available in the literature. In this chapter, an in-depth study is conducted to discover the various methods used to solve the BAP. First, the basic formulation for the BAP is given. The available approaches to generate various buffer configurations are then explored. Once various buffer configurations can be generated, a means to evaluate them is needed. Evaluative methods are covered first for simple lines, then for the more complex lines. The concept of Simulation-based optimisation (SBO) is then introduced as the preferred solution approach in this dissertation.

2.1 Basic formulation

To explain the components of a production line, consider the example given in Figure 2.1 with K machines, each denoted by $M_k, k \in \boldsymbol{K} = \{1, \dots K\}$. Parts flow sequentially from

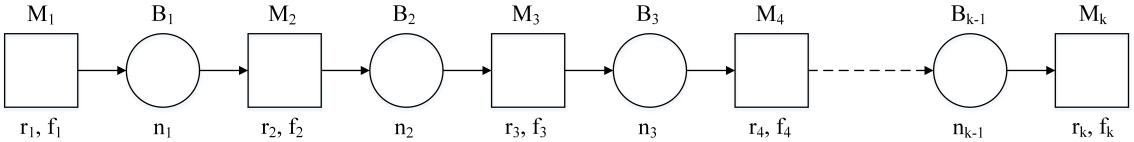


Figure 2.1: Problem notation on a serial production line [12].

M_1 to M_k. The final machine M_k is never *blocked*, meaning once a part is processed, there is always space for it to exit the system. Between consecutive machines, there are buffers where parts await processing. The set of buffers are denoted by $\boldsymbol{B} = \{B_1, B_2, \dots, B_{k-1}\}$

where B_1 represents the buffer between machines 1 and 2, B_2 the buffer between machines 2 and 3, etc. Jointly we represent the set of buffers by $\boldsymbol{B} = \{B_1, B_2, \ldots, B_{K-1}\}$.

Associated with each machine $k \in \boldsymbol{K}$ are several parameters. The first, t_k, denotes the processing time required at machine k. Secondly, f_k denotes the time between failures of machine k while r_k denotes the repair time of machine k. We denote the line's total buffer size as $N = \sum_{k=1}^{K-1} B_k$. An example of the buffer vector for a five machine line can be, $\boldsymbol{B} = \{4, 3, 2, 1\}$, where $B_1 = 4, B_2 = 3, B_3 = 2, B_4 = 1$ with a total line buffer size of $N = 10$. In the basic cases, the BAP assumes the processing times of all machines are equal and deterministic. That is, it is assumed that $t_1 = t_2 = \ldots t_K$.

The BAP is concerned with finding B_k, represented by the vector $\boldsymbol{B}$, that best achieves the organisation's objective. This goal is usually measured against the throughput of the production line. Line throughput is a function of the buffer vector, expressed as $f(\boldsymbol{B})$.

Three objectives have been proven prominent in the literature for BAP [10].

Objective 1 Maximise the throughput of the line for a given fixed number of buffer N [9, 21, 23]. The BAP formulation is then expressed as:

$$\max \; f(\boldsymbol{B})$$

subject to

$$\sum_{i=1}^{K-1} B_i \leq N$$

where N is a predefined value.

Objective 2 A prescribed throughput, $f^\star$, must be achieved with the minimum total buffer size [11, 16, 24]. The formulation then changes to:

$$\min \; \sum_{i=1}^{K-1} B_i$$

subject to

$$f(\boldsymbol{B}) \geq f^\star$$

where $f^\star$ is a predefined value.

Objective 3 Minimise the average WIP inventory, $Q(\boldsymbol{B})$, for the buffer configuration subject to the total buffer size and prescribed throughput. The problem is then formulated as:

$$\min \; Q(\boldsymbol{B})$$

subject to

$$\sum_{i=1}^{K-1} B_i \leq N$$
$$f(\boldsymbol{B}) \geq f^\star$$

where both N and $f^\star$ are predefined values.

Solving the BAP for objective 2 is the aim of this dissertation. Production plant designers are faced with the problem of achieving a balance between throughput and WIP. The solution approaches studied aim to solve objective 2 problems.

2.2 Solution approaches

The objective of the BAP is to find a buffer configuration $\boldsymbol{B}$ that meets a specific objective function. The simplest objective function is to maximise the line's throughput, $f(\boldsymbol{B})$, for a given total buffer size N. To achieve this, various permutations of the buffer configuration need to be generated and tested. All the possible permutations are called the *solution space*. To say that a solution is truly the best, all possible scenarios in the solution space need to be generated and tested. Creating these buffer scenarios is called the generative method in the literature.

2.2.1 Generative method

The simplest way to find the best buffer configuration is to generate all the possible buffer permutations and test every single one of them. This is known as complete enumeration. The BAP has been proven to be an NP-Hard combinatorial optimisation problem [10]. That is if we increase the number of machines or the total number of buffers that can be allocated among the various buffer locations, the solution space grows exponentially. Thus, the complete enumeration is only possible for very small problems.

Because it is not possible to generate all the possible buffer permutations, a subset of them can be generated in a smaller amount of time. The question is then how do we select this subset of permutations from the solution space that needs to be tested and how is the *best* then found.

Various search methods have been used in literature. Traditional search methods have been applied to the BAP. Methods such as gradient search algorithm [15, 16], knowledge-based methods as well as degrading ceiling local search heuristics have been used.

These methods start with an initial solution. They then generate all possible permutations of the initial solution that can be reached in one step. A step can be seen as increasing the buffer space limit with a specific value and decreasing another with the same value. All the permutations are tested, and the buffer configuration that results in the highest throughput is then considered as the best solution. This is repeated until no other improved buffer configuration can be found. Although these methods are more efficient than complete enumeration, they cannot escape local optima.

Unlike the previous search method, metaheuristics are allowed to select a permutation that results in a worse throughput than the current best. This allows the metaheuristics to escape a local optimum and move to other areas in the solution space in the hope to find a better local optimum or ideally the global optimum. That means metaheuristics add logic to the search, giving it the ability to escape local optimum. It is a method that controls the way we move through the solution space. Metaheuristics have been widely applied in the buffer allocation problem. The following section explains two common metaheuristics in the BAP literature.

Genetic algorithm is based on the theory of evolution that favour the reproduction of individuals with specific traits. Let N denote a specific total buffer for the line, where $N = \sum_{k=1}^{K-1} n_k$. Assume $N = 50$, the buffer vector $\boldsymbol{B}$ can have the following configuration. $\boldsymbol{B} = \{5, 6, 4, 5, 20, 10\}$. The following steps form the basis of a genetic algorithm [25].

Each iteration is referred to as a *generation* in evolutionary algorithms. For each generation, a selection of the population is made and known as the parents. The

parent solutions reproduce/mutate into new solutions, called *offspring*. The population is then updated with the new offspring. The initial population is usually randomly generated scenarios. The iteration of generation creation is repeated until some stopping criteria are met. The stopping criteria can be after a predefined number of generations, $m_{\max}$ has been evaluated.

Let m denote the iteration number in the algorithm.

Step 1 Set $m = 1$, where m is the generation number. Select s initial parents, where $s > 1$. Ideally, all the s solutions should be relatively "good" although this is not a requirement. Example, set $s = 3$, it means that 3 parents are required to start the algorithm;

- Parent 1: $\{5, 6, 4, 5, 20, 10\}$
- Parent 2: $\{8, 4, 8, 8, 10, 12\}$
- Parent 3: $\{3, 10, 15, 15, 5, 2\}$

Step 2 Identify the best and worst among the s solutions by determining the line throughput for each parent s. Denote the best by $\overrightarrow{x}_{\text{best}}$ and the worst by $\overrightarrow{x}_{\text{worst}}$. Randomly select a parent of $\overrightarrow{x}_{\text{best}}$ and call it $\overrightarrow{x}_{\text{new}}$. Replace the worst parent with the new child.

Various methods exist for generating offspring. Offspring generating mechanisms are adapted to specific problems. For the BAP this mechanism will subtract a predefined total from a specific buffer and add the same value to a different buffer.

Step 3 Increment generation, m by 1. If $m = m_{\max}$, return $\overrightarrow{x}_{\text{best}}$ as the best solution and stop. Otherwise repeat step 2.

Various variants of the format above exist in literature [13, 25]. Variants focus on aspects such as selection paradigm in which individuals with better solutions are selected with higher probability. Selected individuals can then reproduce in various ways (e.g., crossover, mutation) to generate new offspring. Various replacement schemes also exist for replacing parent solutions with offspring.

Tabu search is a widely used metaheuristic. A distinctive feature of the Tabu search (TS) algorithm is the so-called *tabu list*. The tabu list is a list of mutations that are prohibited in the algorithm to avoid cycling to recently visited solutions. The basic steps to the TS are as follows:

Let m denote the iteration number in the algorithm. Then $m_{\max}$ denotes the maximum number of iterations to be performed.

Step 1 Set $m = 1$. Select an initial solution $\overrightarrow{x}_{\text{current}}$ randomly. Set $\overrightarrow{x}_{\text{best}} \leftarrow \overrightarrow{x}_{\text{current}}$, where $\overrightarrow{x}_{\text{best}}$ is the best solution obtained so far.

Step 2 Select s number of neighbours, of $\overrightarrow{x}_{\text{current}}$. Call these neighbours $\overrightarrow{x}_{\text{new}_s}$ respectively. Determine the throughput for each scenario s.

Step 3 Rank the solutions from best to worst, based on the throughput. Compare the best of these solutions, with the items in the tabu list. If the best item is in the tabu list, skip it, and go to the second-best. Repeat this until a solution that is not in the tabu list is found.

Step 4 Replace the $\overrightarrow{x}_{\text{current}}$ with the best scenario that is not tabu as the new $\overrightarrow{x}_{\text{current}}$. Enter $\overrightarrow{x}_{\text{current}}$ at the top of the tabu list. Each item in the tabu list

moves one space down. The tabu list has a specified length, if the list exceeds this length, remove the bottom item. The length of the tabu list is the tabu tenure and is problem-dependent. There is no set rule on how to pick the tabu tenure.

Step 5 If $\overrightarrow{x}_{\text{current}}$ is better then $\overrightarrow{x}_{\text{best}}$, change $\overrightarrow{x}_{\text{best}} \leftarrow \overrightarrow{x}_{\text{current}}$, if not do not change $\overrightarrow{x}_{\text{best}}$. Increment m by 1. If $m = m_{\max}$, *STOP*, and return $\overrightarrow{x}_{\text{best}}$ as the solution. Otherwise, go to Step 2.

The tabu list thus contains a list of permutations that have been made recently. Maintaining the lists avoids the solution from moving back and forth between the same scenarios.

Demir et al. [9] used an Adaptive tabu search (ATS) to solve the BAP for an unreliable, unbalanced line. The objective is to maximise the throughput of the production line $f(\boldsymbol{B})$, for a given buffer size constraint N. The algorithm distributes a fixed number of buffer between each buffer location B_k. In their paper Demir et al. [9] proposed a new ATS and compared it to the simple TS as discussed above, referred to as Standard tabu search (STS). Various changes are made to the STS which are discussed below.

Neighbour generation The moves through neighbouring solutions are depicted by the notation i, j, meaning that a given number of buffers is added to location i and the same number is subtracted from a location j. All the possible i, j scenarios are generated from the current solution. In the STS the size of buffer change at location i, j is set to 1. In the ATS the incremental size is subject to the problem size. It is set to 1 for small and medium-sized problems, i.e. 5 and 10-machine lines. For large problems involving 20–40 machine lines, the size is set to 1% of the total buffer size, rounded up.

Tabu list The full move tabu criterion was employed in both the STS and ATS. If the move i, j produces the better objective function from the various scenarios evaluated then move j, i becomes tabu for a certain amount of iterations until it is moved off the list. The tabu tenure, which is the length of the tabu list, is set to $\sqrt{\text{ns}}$ where ns = neighbour solution space size. The tabu tenure for the ATS is tuned adaptively. Initially, the tabu tenure is set to a predefined minimum value. It is then calculated for each move. If the objective function is improved, the tabu tenure is decreased by 1. If the solution is not improved, it is increased by 1, subject to an upper and lower limit for the tabu tenure.

Intensification and diversification The ATS also employs an intensification strategy. If a solution found to be the best does not change for a certain number of iterations, the increment (decrement) size is reduced to 1 for large-sized problems.

Diversification is also employed in the ATS. After several iterations without any improvement, a new random buffer configuration is generated. The algorithm then explores from this solution onwards.

Demir et al. [9] also showed that the quality of the initial scenario on which the algorithm starts could affect the quality of the final solution. Three different methods for determining the initial allocation of buffer were compared. Either using the *ratio of failure to repair rate*, the *processing rate* or using *random initialisation*. Experiments showed that using the *ratio of failure to repair rate*, where the machine with the higher ratio receives more buffer for its exit buffer, results in a good initial solution.

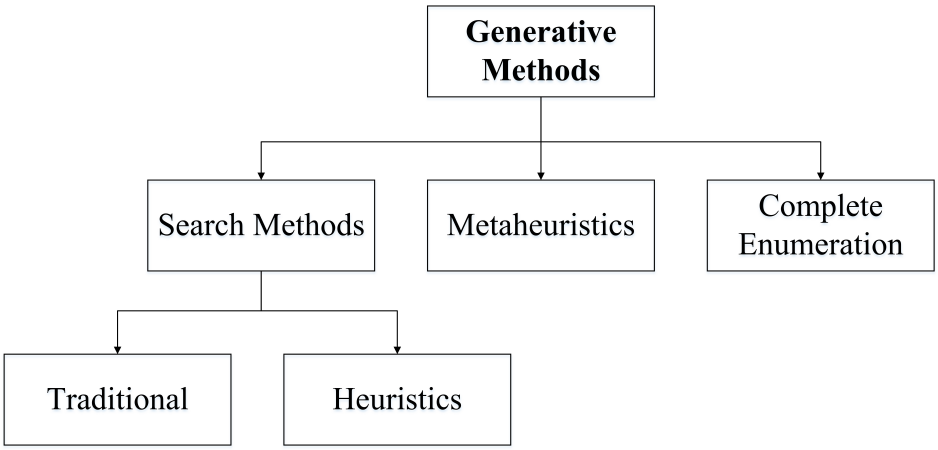


Figure 2.2: Generative methods.

The ATS can have significant performance improvement compared to the STS. In all instances an improved throughput is achieved by the ATS. The improvement is even more noteworthy for large sized problems. They found that in most large-sized problems, the STS cannot avoid local optima.

The objective function of this dissertation is not to maximise the throughput of the line for a given N, but to achieve the desired throughput with the smallest N. Demir et al. [11] extended on the work of Demir et al. [9] by introducing two control loops to find the minimal buffer size required to achieve the desired throughput. The inner loop includes a ATS to obtain a maximum throughput for a given buffer size N as discussed above. Binary search and TS was evaluated for the outer loop. The outer loop sequentially decreases the total buffer size N to find the desired throughput with a minimal buffer. First, the outer loop will use a specified N from the user. In most cases, the initial N is a big number. The inner loop will then determine the best throughput possible for N, by searching for the relevant buffer configuration. If the throughput is higher than the desired throughput, the outer loop can decrease the size of N. Again, the best throughput for the new N is determined. This is repeated until the size of N can't be decreased as it will result in a lower throughput than required.

The proposed method was tested across a wide range of possible line properties. Both binary search and TS for the outer loop were able to solve small to large scale problems. For large problems, the binary search was executed faster than TS, but the authors noted that using parallel implementation of the TS can result in faster execution time than the binary search.

Figure 2.2 summarises the possible approaches for the generative step. Recall that the BAP is an NP-hard problem. Thus complete enumeration and search methods cannot be used. The only way to obtain a near-optimal solution within a reasonable amount of time is to use a metaheuristics. Both the genetic and TS algorithms have been used in literature. The work of Demir et al. [9] shows that the TS can successfully be applied to the BAP to minimise total buffer size for a given throughput target. Therefore the same approach is considered in this dissertation.

During the generation of buffer permutations, the throughput of the various permutations needs to be tested. Various ways to evaluate the throughput of a production line exists in the literature. In the BAP this step is called the *evaluative method*.

2.2.2 Evaluative method

Every time the generative method generates a new buffer configuration $\boldsymbol{B}$, that is new values for the buffer's maximum capacity, B_k, the buffer configuration needs to be evaluated. Recall that a line's throughput is a function of its buffer vector, $f(\boldsymbol{B})$.

A large number of buffer permutations needs to be evaluated to get the *best* solution. It would have been ideal if a simple algebraic formula could be expressed for $f(\boldsymbol{B})$, as an algebraic formula would be quick to solve.

Unfortunately, machine failure and processing time are stochastic in nature. The algebraic relation between the buffer configuration and throughput of the line is very limited. The next best option is to approximate the line's throughput, using analytical methods. The line reliability and topology influences the approach we can use.

For very short, serial production lines exact results based on queuing models are possible [9]. This is not the case for the body shop.

For larger production systems, approximation methods are used. Two approximation methods studied in the BAP literature are the decomposition method [9, 11, 21] and aggregation method. The decomposition method breaks the line up into L smaller lines, while the aggregation method combines the stations. These methods are very similar in implementation. Thus only the decomposition method will be investigated in detail. The decomposition method is time-efficient in the calculation of the line's throughput and is based on the theory of queueing networks.

Decomposition Method

Gershwin [12] proposes a decomposition method to analyse a queuing system with a finite buffer. This method is applied to production lines to calculate the throughput.

To explain the components of the method, consider the example of a queuing network. Customers arrive at a service area, wait in a queue until the other customers' services are completed, obtain the services they require and sequentially move through the service areas until they leave the system. Not unlike the behaviour of a production line. It is possible to calculate both the throughput of the system and the average number of operators in each queue [12]. The following section describes the workings of the decomposition method.

Each machine M_k has two states; working or being repaired. During a time unit, if machine M_k is active and servicing, thus working, it has a probability of f_k to fail. Note f_k previously denoted the time between failure, whereas for the decomposition it is the probability to fail in the next time unit. The failure time is distributed geometrically with a Mean time to failure (MTTF) of $1/f_k$. If the machine failed, it has a probability of r_k to be repaired within the next time unit. Again, unlike before, r_k now denotes the probability to be repaired and not the time to repair as denoted previously.

For each buffer B_k as in the basic formulation, B_k can be in a_k states, $a_k = 0, 1, ...n_k$, where a_k is the number of material in B_k and n_k is its capacity.

The decomposition method takes the serial production line and creates $K-1$ lines, each with two machines and a buffer, Figure 2.3. These lines are denoted as $L_{(i)}$, $i \in \boldsymbol{K} | i \leq K-1$ The decomposed line represents the characteristics of the serial line. B_k in $L_{(i)}$ equals B_k in the K machine line.

Each line $L_{(i)}$ has two machines, denoted as $M_u(i)$ (upstream from buffer) and $M_d(i)$ (downstream). Consider line segment $L_{(i)}$: it will have two machines out of the original line $\boldsymbol{K}$. These machines are $k = 1$, M_1, and $k = 2$, M_2. Machine M_1 is shown in line $L_{(1)}$ as $M_u(1)$ while machine M_2 is $M_d(1)$.

The attributes of each inline machine segment $L_{(i)}$ is as follows:

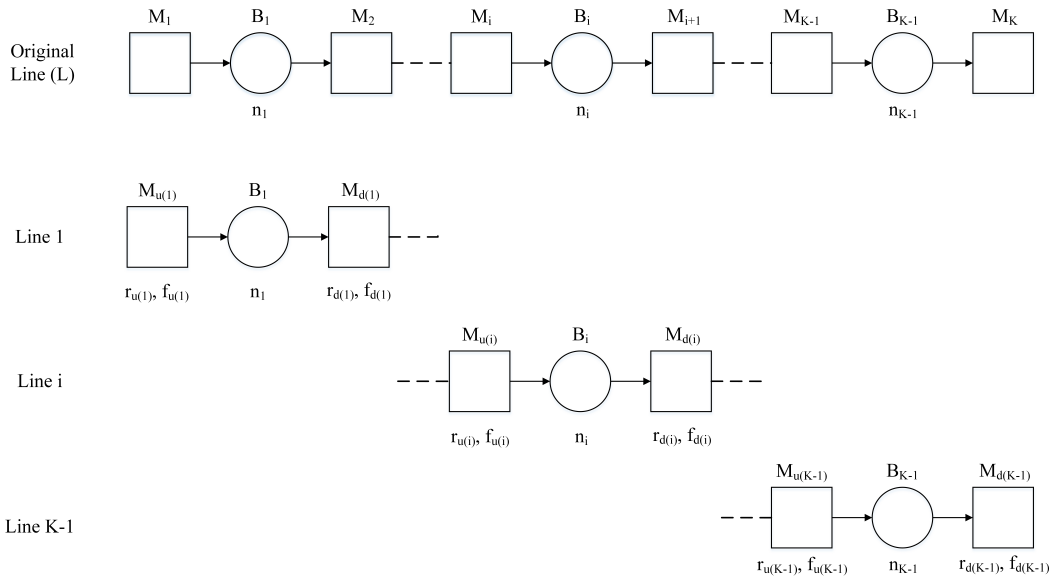


Figure 2.3: Serial production line decomposed [12].

- The first machine in line $L_{(i)}$, $M_u(i)$ has a failure probability $f_u(i)$ and repair probability $r_u(i)$ that represents all upstream stations.
- The second machine, $M_d(i)$ has a failure probability $f_d(i)$ and repair probability $r_d(i)$ that represents all downstream stations.

The machine M_k also has a global variable α_k, where α_k is 1 if the machine is operational (not failed) and 0 if the machine is not operational (failed).

The four attributes are derived by a $4(k-1)$ polynomial equation. They are derived from the algorithm developed by Gershwin [12]. The attributes represent the upstream and downstream performance of the line. Thus flow rate into $B_{(i)}$ in $L_{(i)}$ is closely matched to that of B_k in the $\boldsymbol{K}$-machine line. The probability that the buffer is empty or full is also matched. The machine is starved if its upstream buffer is empty and blocked if the downstream buffer is full. The number of material in $B_{(i)}$ at any time can be expressed as $a_{(i)}$, $0 \leqslant a_{(i)} \leqslant n_{(i)}$. The buffer will either gain or lose one unit during a time unit.

When machine M_k is under repair, $\alpha_k = 0$, it then has the probability r_k to become operational during each time unit:

$$prob[\alpha_k(t+1) = 1 | \alpha_k(t) = 0] = r_k$$

When machine K is working, it has a probability f_k of failure:

$$prob[\alpha_k(t+1) = 0 | a_k - 1(t) > 0, \alpha_k(t) = 1, a_k(t) < n_k] = f_k$$

Repair and failure occur at the beginning of a time unit and buffers change levels at the end of the time unit. Thus during periods in which starvation and blocking do not influence buffer $B_{(i)}$,

$$a_{(i)}(t+1) = a_{(i)}(t) + \alpha_k(t+1) - \alpha_k + 1(t+1)$$

More generally:

$$a_{(i)}(t+1) = a_{(i)}(t) + I_{ui}(t+1) - I_{di}(t+1)$$

where $I_{ui}(t+1)$ is the indicator of whether flow arrives at buffer i from upstream, that is:

$$a_{(i)}(t+1) = a_{(i)}(t) + I_{ui}(t+1) - I_{di}(t+1)$$

$$I_{ui}(t+1) = 1 \text{ if } \alpha_k(t+1) = 1 \text{ and } a_{(i-1)}(t) > 0 \text{ and } a_{(i)}(t) < n_{(i)},\ 0 \text{ otherwise.}$$

The indicator $I_{di}(t+1)$ of flow leaving buffer i is defined similarly. The performance measure E_i is defined by production rate (which can be *throughput*, *flow rate*, or *line efficiency*) of machine M_i in parts per time unit:

$$E_{(i)} = prob[\alpha_k = 1, a_{(i-1)} > 0, a_{(i)} < n_{(i)}]$$

Due to the conservation of flow, parts are neither destructed nor created. This means every part starts at the first machine and flows through each machine K until they exit the last machine. No parts can leave or enter in-between the start and last machines. It can thus be assumed that the steady-state production rate of one machine will equal that of all the other machines.

$$f(B) = E_1 = E_2 = ... = E_i.$$

If each machine was truly independent (thus no influence from downstream or upstream) the flow rate of each machine can be described by $e_i = \frac{r_i}{r_i + f_i}$. The actual production rate of E_k of M_k is less due to starvation and blocking.

$$E_i = e_i prob[a_{i-1} > 0, a_i < n_i]$$

Because $prob[a_{i-1} = 0, a_i = n_i] \approx 0$ as the probability of reaching states where $a_{i-1} = 1$ and $a_i = ni - 1$ by means of transition is approximately 0, the formulation is:

$$E_i \approx e_i(1 - prob(a_{i-1} = 0) - prob(a_i = n_i)$$

The decomposition method uses a set of equations that link the decomposed two-machine lines together. These non-linear equations are solved to determine the throughput of the line. The method has been employed in the BAP literature for both homogeneous and heterogeneous lines.

The method described above was improved by Burman [5], to be able to model heterogeneous lines. The research papers using the model assume that the processing time of machines M_k, that is t_k, are continues stochastic variables. All the papers used one random distribution for all the equipment, that is, $p(t_1) = p(t_2) = \ldots p(t_k)$.

The decomposition method is dependent on specific assumptions:

Reliable, unreliable lines - the decomposition method is capable of approximating production lines which are unreliable, similar to the needs of the body shop.

Line topology - the decomposition method is based on a serial production line which can be approximated by $\boldsymbol{L}$ sub-lines. Each line has a buffer with one station in front, and one after the buffer. Literature could not be found where the method is applied to non-serial production lines. In non-serial production lines, such as the body shop a buffer can have multiple stations in front as well as multiple stations afterwards. Therefore the method described above can not be applied to the body shop problem.

Parameter distributions - in the works of Demir et al. [9], the processing time for all stations is a continuous stochastic variable with the same distribution, that is $p(t_1) = p(t_2) = \ldots p(t_k)$. No paper using the decomposition method could be found where each machine could have its own unique distribution, that is $p(t_1) \neq p(t_2) \neq \ldots p(t_k)$. The body shop processing time is also a continuous stochastic variable, where not all machines have the same distribution.

The decomposition method has been widely used because of its speed and accuracy. Unfortunately, the decomposition method cannot be applied for tree structure systems with various stochastic variables such as the body shop. The only practical way to calculate the throughput of such a complex system is a simulation model.

Simulation and meta-models

An alternative method used as an evaluative method for the BAP is a simulation model. Recall that simulation is an approach of using computers to imitate the operations of various kinds of real-world systems. It is used to perform tests on the simulation instead of the real system. The imitation created with the simulation is known as a model and is built with certain assumptions. With simulation, the real system is not analysed, but the model. It must then be assumed that if the model is an accurate imitation of the system, that the results obtained with the model might also be obtained in the real-life system if similar changes are made to it. Simulation allows any line topology to be simulated. Reliable and unreliable lines can be modelled, and each station can have unique parameters. Due to this flexibility, it has been used widely in complex systems such as production lines [18].

Simulation is applied in various research contributions [2–4, 7, 8, 17, 19, 24, 26, 27]. There are two ways in which simulation is used as an evaluative method. The simulation itself can be used to determine the throughput for each buffer configuration. Alternatively, a meta-model can be made of the simulation, and this meta-model is then used to determine the throughput.

Meta Model

A simulation model is a representation of a real-world system, whereas the term meta-model refers to a mathematical approximation of a simulation model. Meta-models are developed to obtain an understanding of the relationship between the input variables and the output variables of the system under investigation.

The complexity of cellular manufacturing is also a motivation for the use of meta-models as the evaluative method when assigning buffers. Lee [19] developed an approach to find the buffer configuration that leads to the lowest cost in terms of investment and running cost. Simulation was selected for its flexibility and realism. The simulation was used to determine queuing statistics such as average utilisation of the machines, average waiting time and standard deviation of waiting time. One simulation run is executed with an infinite buffer to calculate the statistics. These were then used in a meta-model, called the *line search procedure*, to find the optimal buffer configuration.

Amiri and Mahtashami [2] proposed a multiobjective formulation to solve the BAP for an unreliable line. A detailed discrete event simulation is used to build a meta-model, which is then used for estimating the throughput. The objective is to maximise the throughput of the line and to minimise intermediate buffer storage.

Numerous methods have been used to develop meta-models. The one used by Amiri and Mahtashami [2] was a polynomial regression model. A two-level factorial design is

done in their paper. It requires $2 \times 2 \times 2.... \times 2 = 2^k$ observations and is defined as a 2^k factorial design (k denotes the number of buffer locations). For example if a line has 20 buffer storages, the full 2^k design requires $2^{20} = 1\,048\,576$ observations. The level represents the different values chosen to study the factors, thus in the instance above only two values (levels) for each buffer is considered. It is too time-consuming to use a three-level factorial, 3^k for the example of the 20 buffer locations as it will require $3^{20} = 3\,486\,784\,401$ observations. With a low number of observations, the accuracy of the meta-model is decreased. The simulation was done using MATLAB software and took 5 to 6 minutes per run. To test the validity of the meta-model, ten different buffer configurations were compared against the simulation model. The error between the two methods varied from 1.611% to 5.365%.

Can and Heavey [6] investigated the robustness and accuracy of genetic programming to create meta-models. In a subsequent paper, Can and Heavey [7] created a meta-model of a simulation of the line studied and used genetic programming and artificial neural networks to solve the meta-model.

Chan and Ng [8] made a simulation on Siemens IV to find an algebraic relation between line throughput and buffer sizes.

The use of meta-models allows for more valid representations of complex systems being modelled than the decomposition method as well as faster execution than pure simulation. The disadvantage of the meta-model is that if the algebraic relation between line throughput and buffer size is not valid for all scenarios, an invalid representation is being solved. Due to this, a meta-model cannot be used for the body shop problem.

Simulation

Most papers using analytical methods use deterministic or exponential times for the process, failure and repair times. The methods are limited to simpler line topologies. Simulation is utilised in the literature to relax these restrictions. The simulation model allows general function distributions to be used (e.g. normal, gamma, Weibull and uniform) for all parameters of the production line. Simulation is employed for a more realistic representation of the dynamic behaviour of a system. Discrete event simulation (DES) is used for its effective way of estimating almost any system performance, given that the input data is accurate. Research papers considering real-life systems usually employ simulation as the evaluative method.

Kose Simge and Deniz [17] solved the BAP for a real-world facility, namely a thermo technology company in Turkey. Due to the complexity of the system, a simulation was done using *Arena*. The system investigated had 13 stations. For these stations, data for failure and repair times were evaluated using the *Input Analyser* of *Arena* to fit appropriate distributions. The objective of the company was to improve the production rate by at least 30%. The simulation is used to obtain an average daily throughput for a specific buffer configuration. The rate is then compared to the desired rate, and a new configuration is computed by the algorithm until the termination criteria are met. Various generative methods were evaluated in the paper.

Spieckermann et al. [24] conducted a comprehensive case study at BMW AG. The design approach and topology of the line are similar to the line under investigation in this dissertation. A simulation optimisation approach was used to solve the BAP. An existing simulation model made using *SIMPLE++* was combined with commercially available optimisation packages, the *WitnessOptimizer* and *SIMPLE/GA*. Simulation is a common tool within the automotive industry. It is very convenient to use when determining a line's throughput, but very expensive as each evaluation of the objective function requires at

least one simulation run. The use of commercial optimisation tools had to be considered as black boxes, with only a small degree of configuration. Execution times can be up to several days and evaluated solutions per optimisation run had to be restricted.

Figure 2.4 shows a summary of the various evaluative methods used in the BAP literature. Analytical methods do not apply to complex systems such as the body shop. Simpler serial production lines can be modelled by the decomposition model. It has been well developed in the literature and proved its efficiency and accuracy. Unfortunately, the decomposition approach cannot be used for lines with complex tree structures line topology. Certain assumptions around processing time, failure probability and repair probability are required. Simulation allows for these restrictions to be relaxed [26]. This approach is very popular for modelling complex systems. The disadvantage is that long computational time is needed to run the simulation. The generative method and evaluative method need to be performed in an iterative manner as shown in Figure 2.5. The proposed TS metaheuristic is used to generate various buffer configurations. For each step in the generative method, where a buffer configuration is generated, a simulation run is required to evaluate the performance of the line.

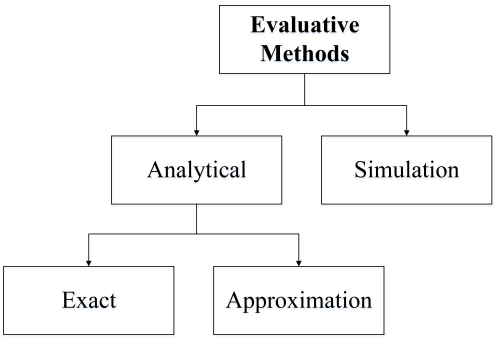


Figure 2.4: Evaluative methods.

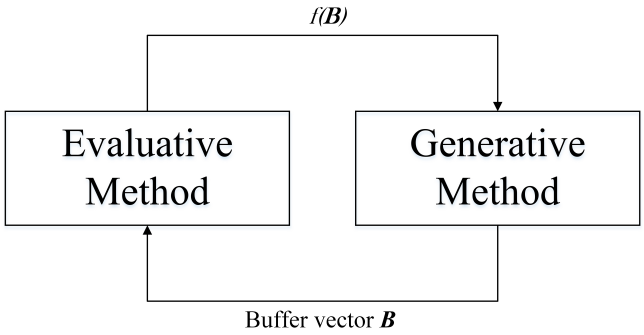


Figure 2.5: Buffer allocation problem solution approach.

The use of simulation to evaluate every output of an optimisation algorithm is known as SBO.

2.2.3 Simulation modelling

One disadvantage of simulation is the long computational time required to evaluate scenarios. This can be reduced by creating a simulation program in a general programming language instead of commercial software. Commercial software has features that allow for

the ease of model creation and report generation. Although this makes it easier to create a simulation, it also increases the processing demand due to all the extra features. If a simulation program is created specifically for the BAP without any special or additional features, it will result in shorter computational times. Therefore creating a simulation program using general programming software is considered in this dissertation.

To create a simulation program, a certain blueprint is required. DES is extensively used in the analysis of modern industrial systems. It has been used in production planning and scheduling, supply chain design and analysis as well as systems design and manufacturing operations [6]. DES is a model that executes by moving through the simulation by changing the state of variables at separate ("countable") points of time called *events* [18]. Therefore DES approach is considered in this dissertation.

The following is an example of DES. Any system can be described as a chain of events that occur. An event can be anything from a part leaving a machine, machine failure or workers becoming available. The time at which these events occur is usually stochastic. Consider the serial production line shown in Figure 2.6. The serial line has four machines (M1, M2, M3, M4) and the buffer vector is $\boldsymbol{B} = \{2, 3, 1\}$. When a simulation

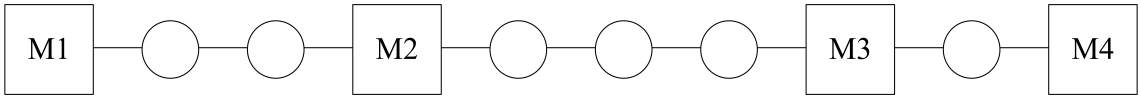


Figure 2.6: Serial production line.

model is created this structure of the line is first defined in the simulation program. For each stochastic machine process, failure and repair times are also defined in the simulation program. At this point, it is also necessary to define the simulation length that is required. This represents the system in real life. The simulation length can range from seconds to years. The advantage of using simulation on computers is that the computational time of the simulation is a lot quicker than the needed simulation length. A simulation representing months worth of production can be computed in minutes.

Once the model has been defined in the simulation program, it can initiate the simulation. The line represented in Figure 2.6 has two types of events: parts leaving a machine, and a machine failing. Upon initialisation, the simulation program schedules these events. The time of each event occurrence, t_k, is randomly generated from the stochastic variables. This is added to the current time to determine the time of occurrence. For our example, let us assume that the following times for t_k where generated. As the current simulation time is 0, the execution time of each will be $0 + t_k$. The processing time of the events was generated randomly.

- Part done M1: 15.1 s.
- Part done M2: 10.12 s.
- Part done M3: 14.31 s.
- Part done M4: 19.21 s.
- Machine fails M1: 30.25 s.
- Machine fails M2: 45.12 s.
- Machine fails M3: 60.12 s.
- Machine fails M4: 20.1 s.

At this stage in the simulation the line has a part in every machine with an expected completion time, as shown in Figure 2.7.

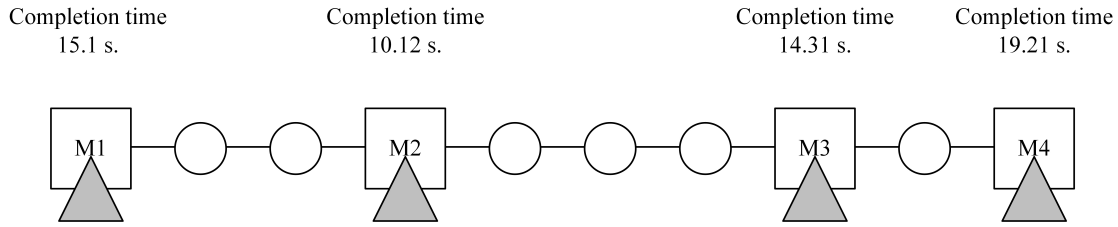


Figure 2.7: Serial production line.

Simulation time = 0 s.

Event List	Event Time
Part done M2	10.12 s.
Part done M3	14.31 s.
Part done M1	15.10 s.
Part done M4	19.21 s.
Machine fail M4	20.10 s.
Machine fail M1	30.25 s.
Machine fail M2	45.12 s.
Machine fail M3	60.12 s.
Sim End	100 000 s.

The simulation uses an event list to control all the events. Each event is scheduled in the event list upon initialisation, as shown left. The events are sorted chronologically according to their times. The simulation also schedules the simulation end event, which is the specified simulation length. A simulation clock is used to keep track of where the simulation is in the event list. At initialisation, the simulation clock is at 0 s. The simulation clock needs to move to the first event in the list, part done M2.

The simulation clock can be advanced by two approaches:

Fixed-increment, the simulation has a predefined time increment size. Assume the incremental size is 0.01 s. The simulation clock will advance with 0.01 s throughout the entire simulation. At each point, the simulation identifies if any event has occurred. This time incremental size should be small enough so that with each increment, an event cannot be overlooked. The disadvantage of this approach is that the simulation program requires time evaluating "inactive" time where events do not occur.

Next-event time overcomes this disadvantage by skipping inactive time. This approach will move from the time of one event to the time of the next event. The reason for this is that no changes to the system states occur during this inactive time. This reduces the computational time of the simulation program. This approach is used for the example.

Simulation time = 10.12 s.

Event List	Event Time
Part done M2	10.12 s.
Part done M3	14.31 s.
Part done M1	15.10 s.
Part done M4	19.21 s.
Machine fail M4	20.10 s.
Machine fail M1	30.25 s.
Machine fail M2	45.12 s.
Machine fail M3	60.12 s.
Sim End	100 000 s.

The simulation clock jumps from 0 s to the first event which is executed at 10.12 s. An event can have a range of actions it takes based on the state of the line. For simplification, let us assume the event checks if there is space in the buffer. If there is space, the completed part moves into the buffer. The event also checks if there is a part available to start work on. In this case not. Thus no new event is scheduled. The state of the serial line is now visualised in Figure 2.8.

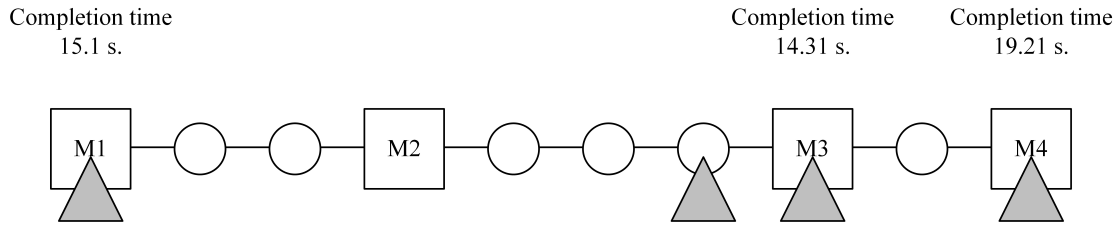


Figure 2.8: Serial production line.

Simulation time = 14.31 s.

Event List	Event Time
Part done M2	10.12 s.
Part done M3	14.31 s.
Part done M1	15.10 s.
Part done M3	16.21 s.
Part done M4	19.21 s.
Machine fail M4	20.10 s.
Machine fail M1	30.25 s.
Machine fail M2	45.12 s.
Machine fail M3	60.12 s.
Sim End	100 000 s.

The simulation clock jumps to the next event in the list, part done M3. The M3 part done event moves the completed part into the buffer. Because a part is available for M3, it can start production on it. The processing time t_k, for the part is generated from the stochastic variable. Let us assume the random time obtained is 1.9 s. The new part will thus be completed at the current simulation time plus the processing time, $14.31\,\text{s} + 1.9\,\text{s} = 16.21\,\text{s}$. This new event is pushed into the event list, by adding the completion time to the current simulation time, so that the event list ranking stays chronologically sorted. The new serial line is shown in Figure 2.9.

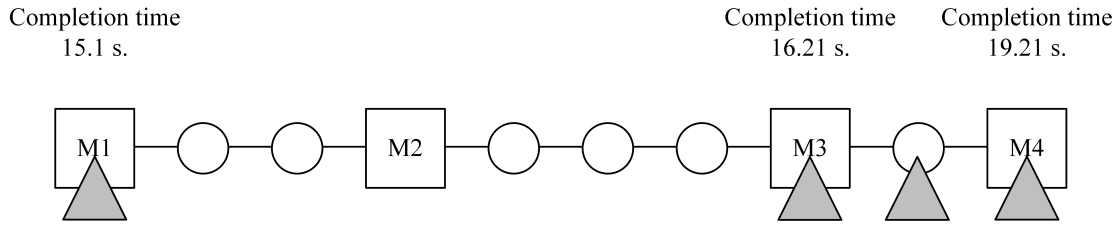


Figure 2.9: Serial production line.

This process of moving from one event to the next is the way a DES is executed. Every time a part leaves machine 4, the total throughput of the line is increased by one.

For DES, shown above, the following components are required for the program. These components and the interaction between them are shown in Figure 2.10 [18].

A *main* program is required that controls all the other components. The main starts by invoking the *initialisation* routine. This routine sets the simulation clock to 0 time units. It initialises all system states and creates the event list with all the initial events as in the first part of the above example. The simulation program then returns control to the main program, which then invokes the *timing routine*. This routine is responsible for managing the event list. It determines which event is next on the list as well as its time of occurrence. It then advances the clock to this event's time. The simulation program again returns control to the main program, which then invokes the *event routine*. There is a specific event routine for each type of event. This routine contains all the actions that need to be taken when the event is executed. The event routine updates system states, statistical counters and generates any future events. If the simulation end time is reached, the simulation stops and the *report generator* returns reports on statistical counters. If this time is not yet reached, the timing routine is again invoked to move to the next event. This action between the timing routine and the event routine is repeated until the simulation end time is reached. The *library* routine generates the random times required

during the event scheduling. These components of a simulation model are contained in

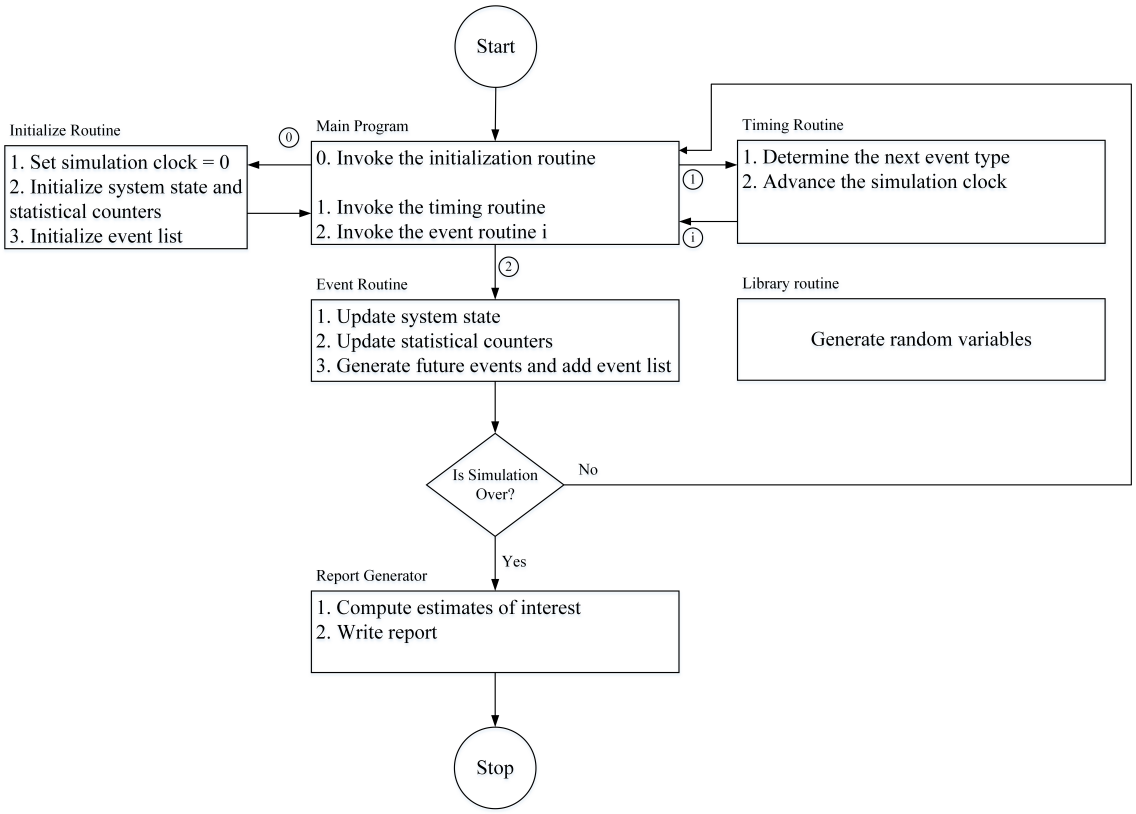


Figure 2.10: Flow control for DES with the next-event time-advance approach [18].

commercially available software packages such as *Arena*, *Simio* and *Siemens Technomatic Plant Simulation*. The software manages the various components and allows the user to create a simulation by placing and connecting various building blocks in a convenient graphical user interface. When the simulation is programmed in a general programming language such as *Java*, each of these components needs to be programmed by the model developer.

Java is a class-based, object-oriented programming language. Object-oriented programming allows the creation of classes which can contain data (*attributes*) and instructional code in the form of prescriptive procedures (*methods*). Multiple instances of the class can be instantiated and are known as *Objects* of a specific class. Object-oriented programming allows various objects to interact with one another.

Tiacci [26] not only created a simulation model of the line under investigation in their work but also a standard set of classes in Java which is capable of performing the various components of DES, called The assembly line simulator (ALS).

It is a discrete event simulation model programmed in Java which is capable of receiving line configurations and building a simulation model around these parameters. ALS was developed in Java, due to the common adoption of the Java programming language. Java programs are easily distributable and work on multiple platforms. The ALS was implemented using the Stochastic simulation in java (SSJ) library. SSJ is an organised set of code, offering general-purpose libraries for DES in Java. It contains the classes and objects required for the various components of DES.

Simulation provides the required flexibility needed to accurately represent the characteristics and parameters of a body shop. Simulation on its own, as well as analytical

methods, are not an optimisation technique. But in combination with a generative method, an optimisation technique is created. The preferred generative method for this dissertation will be the TS algorithm. The combination of simulation and a generative method is known as SBO.

The BAP has been studied intensively and the basic formulation for the BAP is found in literature. Three main objectives are considered when solving the BAP. This dissertation focus on objective two, a prescribed throughput, $f^{\star}$, must be achieved with the minimum total buffer size. The general approach to solve the BAP is using an evaluative method and generative method in an iterative manner. Various methods are employed in literature to evaluate the throughput of the production line. Most commonly the decomposition method. The disadvantage is that the decomposition method has limitations on the line topology it can solve as well as distribution types for the machine parameters. Alternatively simulation or meta-models have been employed to represent more complex lines. Simulation has a more accurate representation of the production line than a metmodel but takes longer to compute. To overcome this DES can be used to create a specific program. Various generative methods are also available. The works of Demir et al. [9, 11] shows that TS is an effective approach as a generative method for the BAP.

Chapter 3

Solution approach

Recall that solving the Buffer allocation problem (BAP) requires finding the *best* size and configuration of buffers in the production line. What is considered *best* will depend on the objective function that is specified. In this dissertation, the *best* buffer configuration will be the one that achieves the desired throughput with the minimum total number of buffers.

The lines under investigation are subject to machine failure and different processing times, and these are stochastic in nature. The throughput of these lines can be evaluated using simulation as the *evaluative method*. In the first part of the chapter, the detail of the simulation program created for this dissertation will be dealt with. This program will then be used to evaluate various production lines.

Evaluating all possible buffer configurations is computationally not practical. Consequently, we employ an optimisation procedure to navigate the solution space more efficiently and generate useful configurations. The second part of this chapter introduces the Tabu search (TS) metaheuristic that is used as the *generative method*.

3.1 Simulation program

Simulation can be done on a computer to imitate the operations of real-world systems. The imitation, known as a *model*, can easily be altered on the computer and the effect of the changes can be evaluated. Simulation on computers can be created using either commercially available software or programming it in a general-purpose programming language. A simulation model that is programmed for a specific problem can have a shorter computational time compared to commercially available software. With most commercial software, an entirely new simulation model needs to be created each time the line configuration or size is changed. To overcome this, a more general program (framework) needs to be created, which will then be used to model a production line.

The proposed simulation program developed in this dissertation will have the basic framework needed to simulate a serial production line with any number of machines. The user of the program needs to define the number of machines, as well as the random variables for processing, failure and repair time. The program will then create a model of the line which can then test various buffer configurations and return the throughput of the line for each. This change to the model is achieved without making any changes to the simulation program's code. A second simulation program is also created for use in more complex lines, non-serial line. This program will be used to test the problem scenario in Chapter 5.

The simulation program is created in Java based on the Discrete event simulation (DES) framework. Recall that the DES works with events. These events are scheduled

in a chronological event list, and the simulation is executed by moving from one event to the next. Upon reaching an event, specific actions need to be taken by the simulation program which, in turn, can result in more events being scheduled.

The simulation program has various components, each managing a specific activity (see Figure 3.1). The components are similar to the ones discussed in the literature

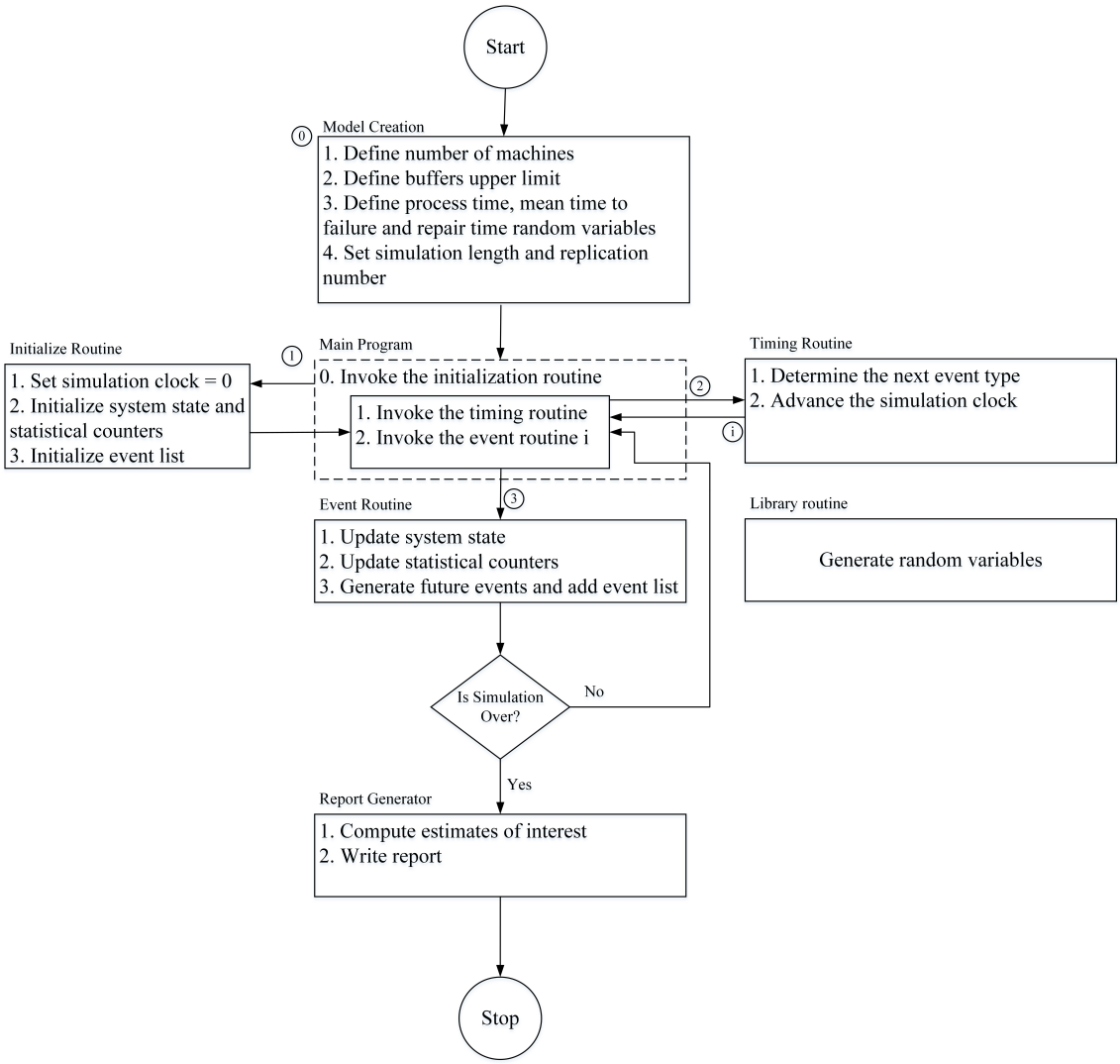


Figure 3.1: Flow control for DES with the next-event time-advance approach.

review except for the addition of a *model creation* program. Because the simulation program proposed in this dissertation is a general framework for a simulation program of a serial line, the model itself needs to be created before the *main program* can invoke the simulation.

The Stochastic simulation in java (SSJ) package [28] is used to assist with the programming. SSJ contains libraries that can manage the event list and generate random variables. The SSJ package provides the programming code for: *initialise routine*, *event routine*, *timing routine*, *library routine* and the *report generator*. For this dissertation, a unique *main program* is created to combine each of these routines. A *model creation* program is also created that can automatically generate a model of a serial production line of any size. The number of machines, buffer size and machine parameters are the only

input needed into the program to create a simulation model of a serial line of any size. The program allows for the creation of parallel lines, but the line layout and flow need to be hard-coded. This is done for the problem in Chapter 5.

The *event routine* of the SSJ has the code for activating the event, but no reaction procedure that needs to be followed. This is expanded with additional code to include the requirements of the BAP and is described in the next section.

3.1.1 Event routine

Two main events are programmed into the event routine. The first event is activated when a part leaves a machine, called a *departure event*, and a second is activated when a machine fails.

Departure event is triggered when a part is completed and leaves the station. Law [18] explains the structure for a general departure event of a simple simulation model. This structure is expanded to meet the needs of the BAP. The departure event is scheduled in the event list and represents a machine completing a part. When the event is performed by the simulation, the logic in the flow chart below is executed, see Figure 3.2.

The departure event first needs to move the completed part to the downstream station. If the downstream station is IDLE and STARVED (meaning the buffer was empty and no part was available to process) the completed part can be sent directly to the downstream station for processing. Transfer time is ignored in the simulation. In automated factories such as the Body-in-white (BIW) transfer time is constant. Transport time can be included by increasing the processing time of the station with the additional transport time. Upon part arrival at the downstream station, the station state is set to NOT IDLE and NOT STARVED. The downstream station immediately starts service on the part. The new random process time is generated and added to the current simulation time. The completion time of the event is scheduled in the event list. If the part arrived at a station that is FAILED, repair time is also generated. The departure event is delayed by the repair time. A new failure time is generated and added to the event list. The failure status is changed to NOT FAILED.

If the downstream station is NOT IDLE, it means that parts might be in the buffer before the station. It is necessary to check if the buffer has reached its capacity. If there is space left in the buffer, the part can exit the current station and join the queue while if it has reached its capacity, the part cannot exit. The current station's blocked state is set to BLOCKED and the departure event ends.

After the part has left the current station, it is ready to receive parts, the station's state is set to IDLE. If there are no parts available in the upstream buffer, the station's state is set to STARVED. The departure event ends.

If a part is available from the upstream buffer, it can be removed from the queue and enter the station, station' state is change to NOT IDLE. All parts in the queue are moved up one space. Upon arrival in the station the random processing time for the part is generated. This is based on the random distribution defined in the model creation step. The processing time of the part is added to the current simulation time to determine the time the part is planned to be completed. The next departure event is scheduled in the event list.

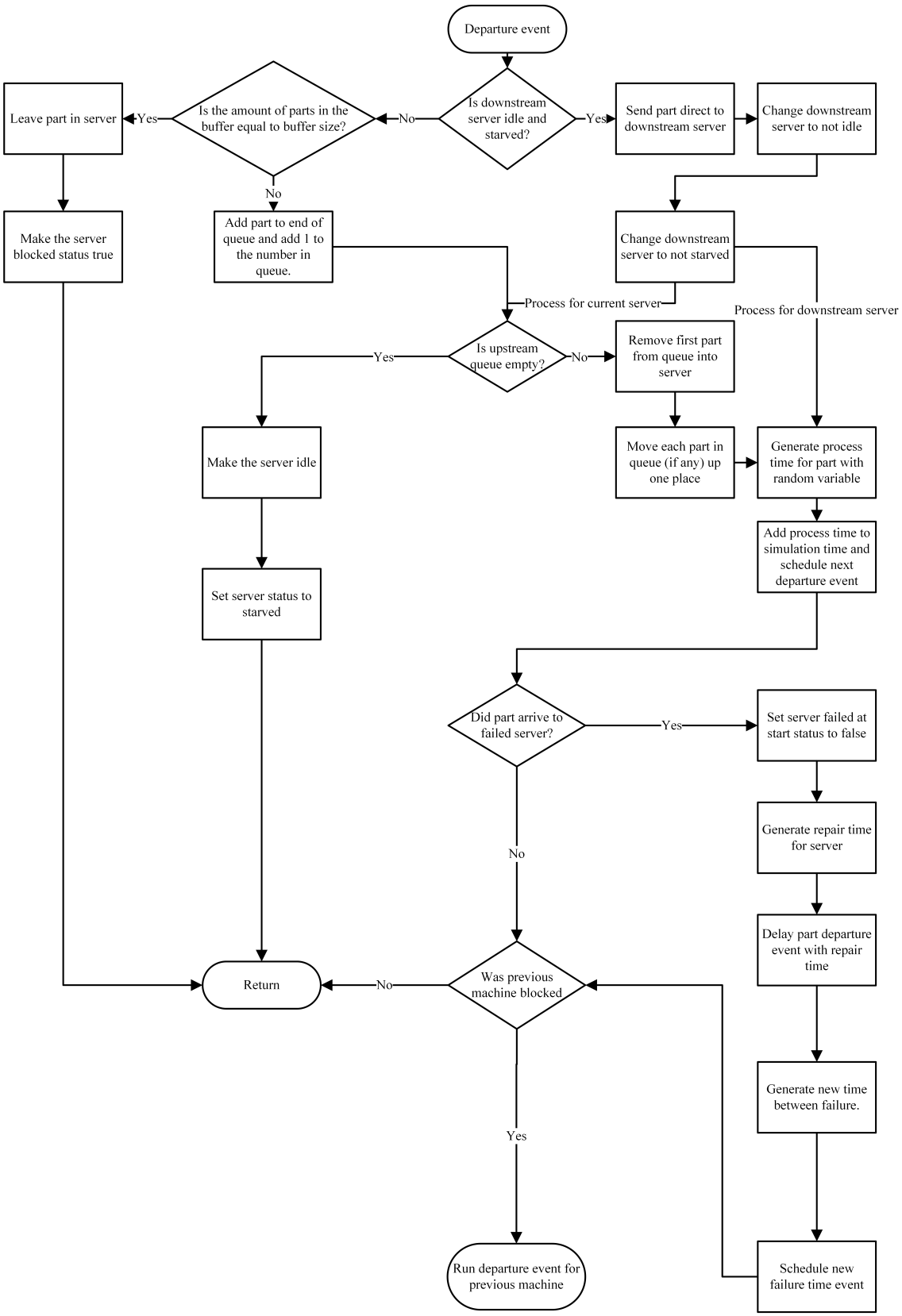


Figure 3.2: Departure event flow chart for simulation model.

The model checks if the part arrived at a station that failed while it was idle. If the machine's failed state is FAILED, repair time is generated randomly from the defined distribution. The departure event that was scheduled is now delayed by the repair time. If the machine's failed state is NOT FAILED, no adjustment to the scheduled departure event is needed.

It is possible that the upstream station was BLOCKED by a full buffer before the departure event was executed. If a part was removed from the upstream buffer, it now has space available for the upstream station to place its blocked part. If the previous station was BLOCKED, a departure event for it could be triggered. If it was NOT BLOCKED no further action is needed, and the departure event is completed.

Failure event is triggered when a station fails. The flow chart for the failure event is shown in Figure 3.3.

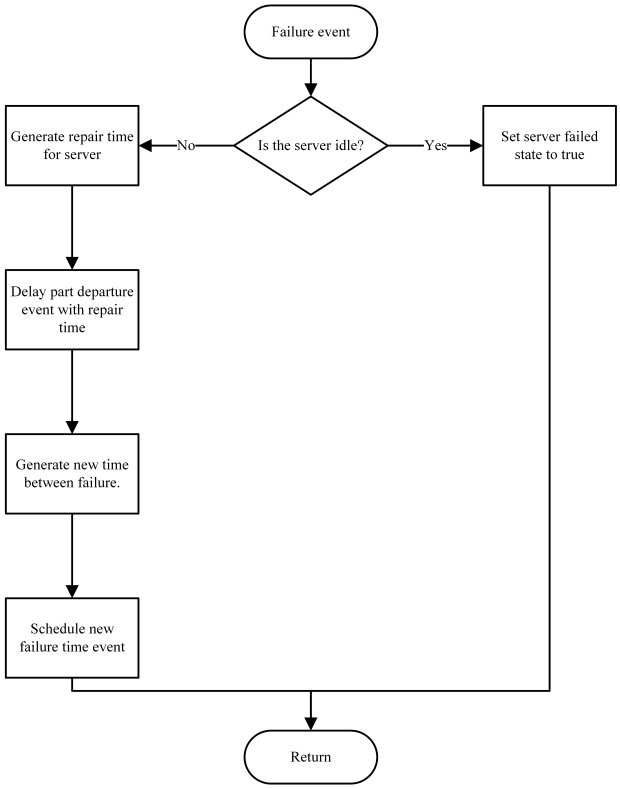


Figure 3.3: Failure event flow chart for simulation model.

When the simulation time moves to a failure event, the station stops processing and awaits repair. The machine can be in one of two states when the failure event time is reached. It can either be IDLE or NOT IDLE.

If the station is IDLE, it means no part is currently being processed by the machine. In practice, the failure will only be detected when the next part arrives at the station. Thus repair is not yet scheduled, but the station fail state is set to FAILED.

If the station is NOT IDLE the failure will stop the machine processing, and the failure is immediately detected. Repair time is generated randomly from a random distribution. The departure event of the current station is delayed by the repair time. A new failure event is scheduled by generating a random time between failure

and adding it to the current simulation time. The time between failure is generated from a specified random distribution.

The inclusion of these events in the event routine concludes the design of the simulation program. The simulation program is used to model the serial production line for Chapter 4 and a tree-structured line in Chapter 5.

The throughput of the simulation model is time-dependent. The designer specifies the timeframe for which the model needs to be evaluated—called the *simulation length*—and can be defined from seconds to years. The main advantage of using computers to execute simulations is that we can evaluate a multi-year simulation within a few hours.

This program is executed in a similar way to other simulation programs by scheduling and executing events until the simulation end time is reached. This program can change from a 10 machine line simulation to a 1 000 with a simple change of the number of machines in the program's parameter.

3.2 Generative method

The generative method moves through the solution space, guided by the objective function, and provides different scenarios for the simulation model to test. The problem investigated in this dissertation attempts to solve the BAP by finding the minimum number of buffer size needed to reach a specific throughput.

The generative method proposed in this dissertation uses two TS metaheuristics: an *outer loop* focusing on reducing the total buffer size of the line, N, and an *inner loop* for calculating the best allocation $\boldsymbol{B}$ of the given buffer size, Figure 3.4.

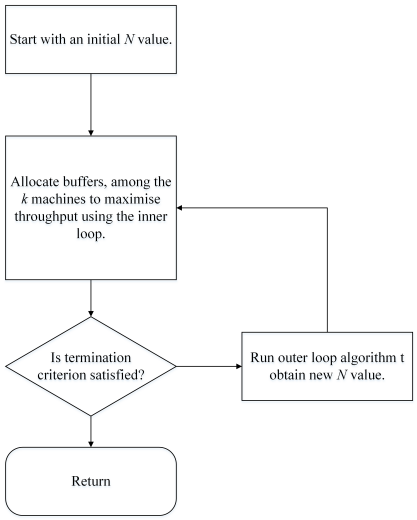


Figure 3.4: Two heuristic approaches [11].

The method starts with an initial total buffer size N. The total buffer size is then distributed among the buffers to obtain a buffer vector $\boldsymbol{B}$. The inner loop TS algorithm is then executed to determine the maximum throughput that can be obtained with a total buffer size N. Various permutations of the buffer vector $\boldsymbol{B}$ are generated. $\boldsymbol{B}$ is used in the simulation model to determine the throughput, $f(\boldsymbol{B})$, for each permutation. With each buffer permutation, the simulation model is executed until it reach a predefined simulation length and replication number and terminates. The solution is then tested against the

termination criteria. If the criteria are met, the algorithm is terminated. If not, the outer loop is run to obtain a new total buffer size, N, for the inner loop to evaluate.

The following section describes the inner and outer loop in more detail.

3.2.1 Inner tabu algorithm

The inner loop aims to find the maximum throughput that can be reached for a specific total buffer size, N. This is achieved by testing various buffer permutations that have a total buffer size of N, and finding the permutation that results in the highest throughput.

The inner Adaptive tabu search (ATS) based on our interpretation of Demir et al. [9], is described in this section. Consider a serial production line with five machines. Four buffer locations are available. The inner loop receives a total buffer size number N from the outer loop. The total buffer is then distributed among the various buffers. Recall that the *ratio of failure to repair rate* is a good approach for determining the initial distribution of a buffer. The generative method will first calculate this ratio for each machine, after which it is summed, and a percentage for each machine is calculated. Based on these percentages each machine then receives a rounded value of buffer which totals to N. If the total allocated buffer exceeds or is less than N, a random buffer's size will be either increased or decreased to meet the total line buffer size of N. The throughput of the line is then determined using simulation. Figure 3.5 is a graphical representation of the line. The line has a total buffer size of $N = 15$, for B_1 the total buffer size will be $15 \times 0.04 = 0.6$, the allocated buffer space is rounded up. Due to the rounding values, the last machines buffer size is not determined by the ratio, 0.7 in this case, but rather the remainder of the buffer size after each allocated buffers size has been deducted. The initial buffer vector is $\boldsymbol{B} = \{1, 4, 1, 9\}$. The average throughput for this configuration is 621.69 units.

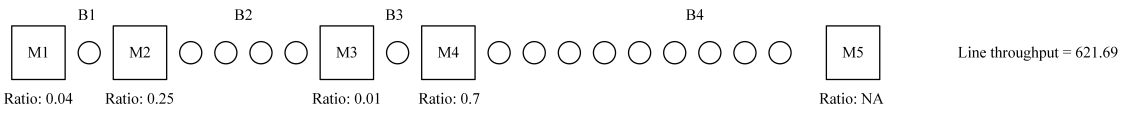


Figure 3.5: Simple production line with buffer allocated according to repair/failure ratio.

Two variables are used to keep track of the *best* buffer vector. The first can be seen as the best of the best. This contains the buffer vector that achieved the highest throughput from all tested scenarios. The above buffer vector and its throughput are stored in this variable, which is called *global best*.

The ATS algorithm then determines all possible permutations of the current buffer configuration by increasing a buffer with the *buffer change size* and decreasing another by the same number to ensure the N of the line stays consistent. For lines with less than or equal to 20 machines, the *buffer change size* is set to 1 buffer space. This allows for small changes in the buffer configuration to be tested. If the line has more than 20 machines, the total number of buffer is usually large. Changing the size of a buffer by only 1 space will generate a large number of scenarios. To prevent this the buffer size is changed by more than 1, its set to $N \times 0.01$, rounded up.

This is done for each possible permutation that can be reached by a single move. Each move is represented by the move vector (i, j), where i is the location where the buffer is added and j the location where the buffer is removed. Figure 3.6 shows four possible permutations of the initial buffer configuration. Each permutation is evaluated using the simulation model.

The best solution out of these permutations is then stored in the second variable. This

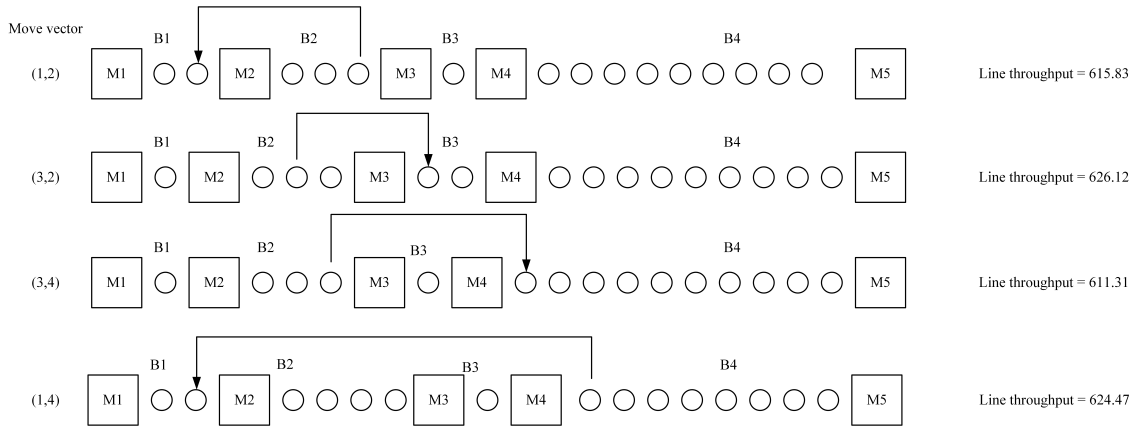


Figure 3.6: Simple production line showing various buffer permutations.

variable keeps track of the best permutation at each iteration, referred to as the *local best*. The ATS then compares the local best solution with the items in the tabu list. If the local best solution is in the tabu list, it is not considered unless it is better than the global best solution found so far. If so, the global best variable is updated with this buffer configuration. If it is not better than the global best, the second-best solution out of the permutations is checked against the tabu list. This is repeated until a usable solution is found. This solution is then the new local best and is added to the tabu list.

The actions discussed above constitute one iteration. The process of creating new permutations for the following iterations, which can be reached within one move of the new local best, is evaluated to obtain a new local best and, if possible, a new global best.

A solution remains tabu for a number of subsequent search iterations, the Tabu tenure (TT). TT is tuned adaptively based on the quality of the current solution and frequency of the moves. If an iteration does not improve the global best, the TT is increased. The TT is decreased when a new global best is found.

An intensification strategy is used to explore promising search space more thoroughly. This is done for large sized problems. If the solution found stays the best for a certain number of iterations ($0.25 \times N$), the *buffer change size* is set to 1.

Diversification guides the search to unexplored areas to avoid local optimality. Two diversification methods are used:

Restart diversification implements several random restarts during the optimisation. After $12.5 \times N$ iterations, a new buffer configuration is generated randomly. This allows for unsearched areas to be considered.

Continuous diversification forms part of the regular search process. A counter is used to track the number of times a specific move is performed. When the counter reaches a predefined value ($12.5 \times N$), the move is penalised and made less attractive. The penalty is determined as 10^{-4}, but can be adjusted.

The algorithm will continue until one of two stopping criteria is reached. Either after $50 \times N$ iterations or after no improvement is made to the *global best* for $25 \times N$ iterations. Algorithm 1 shows the pseudocode for the inner TS.

The termination criteria, intensification strategy and diversification is based on a factor of the total buffer size N. These factors are from the work of Demir et al. [9].

Algorithm 1: Pseudo code TS heuristic inner tabu loop.

input : Total buffer size N to be allocated amongst buffers. Simulation parameters such as simulation length S_{length}, the number of replications $S_{replication}$ and the number of machines in the line K. Machine random variables, *failure, repair and processing time distributions*

output: Best configuration B as a buffer vector and best throughput of the line.

if *ProblemSize* $\geq$ *20* **then**
  bufferChangeSize $= N \times 0.01$
else
  bufferChangeSize $= 1$
end
Determine the initial buffer configuration B_0 using the failure/repair ratio of each machine.;
Run simulation with $S_{replication}$, S_{length}, N and B_0; Store simulation throughput in the *global best* x_{best} and B_{best} as B_0.;
Also store the throughput in *local best* as x_{local} and B_{local} as B_0.;
for *Each j in all possible neighbours of* B_{local} **do**
  for *Each i in number of possible buffer* $K - 1$ **do**
    $B_j = B_{local}$;
    for *Each k in number of possible buffer* $K - 1$ **do**
      if B_k - *bufferChangeSize* $\geq$ *0* **then**
        $B_{ji} = B_{local,i}$ + *bufferChangeSize*;
        $B_{jk} = B_{local,k}$ - *bufferChangeSize*;
        neighbours[0] $= i$;
        neighbours[1] $= k$;
      end
    end
  end
end
Evaluate each of the permutations using the simulation with $S_{replication}$, S_{length}, N and B_j;
if *If throughput of* B_j *is* $\geq x_{global}$ **then**
  x_{global} = best throughput, X_{local} = best throughput, B_{best} and B_{local} is the corresponding buffer configuration. Set *no improvement counter = 0, decrease tabu tenure*
else
  Get best non tabu buffer vector, B_{local} is the coresponsing buffer configuration, x_{local} is the throughput, increase tabu tenure, save move in tabu list
end
Increase *number of iterations* and *non improvement counter* by 1;
if Non improvement calculator $\geq 0.25 \times N$ *and* number of machines $\geq$ *20* **then**
  Set bufferChangeSize $= 1$
end
if Number of iterations $\geq 12.5 \times N$ **then**
  Generate random B_{local}
end
Repeat step 7-9 until termination criteria is met, *Number of iterations* $> 50 \times N$ or *Non improvement calculator* $> 25 \times N$;
Return x_{best} and B_{best}.

The neighbour generating method used by Demir et al. [9] is used to evaluate all possible neighbours, (lines 9–21 in Algorithm 1). To explain this in an example, consider an initial buffer configuration $\{2,3,4,5\}$ for $n = K-1 = 4$ buffers production line. The possible neighbours that can be reached within one step are twelve, $n(n-1) = 4\times 3$, these are:

1. Swap Index: (2,1) - $\{1,4,4,5\}$
2. Swap Index: (3,1) - $\{1,3,5,5\}$
3. Swap Index: (4,1) - $\{1,3,4,6\}$
4. Swap Index: (1,2) - $\{3,2,4,5\}$
5. Swap Index: (3,2) - $\{2,2,5,5\}$
6. Swap Index: (4,2) - $\{2,2,4,6\}$
7. Swap Index: (1,3) - $\{3,3,3,5\}$
8. Swap Index: (2,3) - $\{2,4,3,5\}$
9. Swap Index: (4,3) - $\{2,3,3,6\}$
10. Swap Index: (1,4) - $\{3,3,4,4\}$
11. Swap Index: (2,4) - $\{2,4,4,4\}$
12. Swap Index: (3,4) - $\{2,3,5,4\}$

Each of these neighbours are evaluated with the evaluation method. For each iteration of the inner TS all the one step neighbours are generated and need to be evaluated to determine the best performing neighbour, *local best*. With an increase in number of machines, the number of possible neighbours increase.

3.2.2 Outer tabu algorithm

The outer loop is based on the work of Demir et al. [9]. It reduces the total size of N to obtain the smallest N possible while still meeting the required throughput. To explain how this works, consider the line in Figure 3.7.

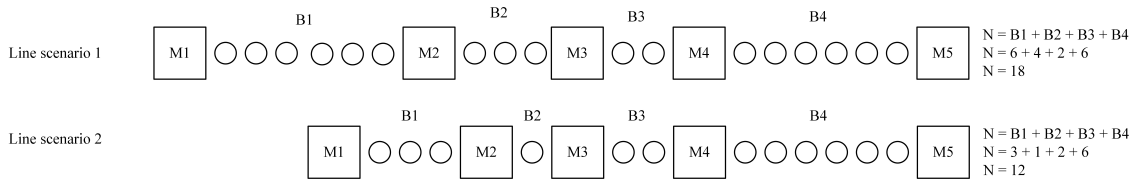


Figure 3.7: Simple production line showing various N sizes.

The algorithm starts with an initial buffer size of N_0. Upon initialisation N should be a sufficiently large number, lets assume this is $N_0 = 18$ as in line 1. The size N is then given to the inner tabu algorithm. This determines the best buffer configuration and returns it as well as the total throughput. The buffer configuration for line 1 is $\boldsymbol{B} = \{6,4,2,6\}$ for a total buffer size $N = 18$.

The outer tabu loop is then used to generate a new total buffer size, assume the result of this is $N = 12$ as in line 2. Again the inner loop evaluates this and returns the best buffer configuration and throughput. Line 2 is $B = \{3,1,2,6\}$ for a total buffer size $N = 12$. This is repeated until either of two stopping criteria is reached:

- no better solution has been found within a certain number of iterations,
- the maximum number of iterations is performed.

The pseudo-code for the outer loop is shown in Algorithm 2:

Algorithm 2: Pseudo code TS heuristic for outer loop.

input : Initial buffer size N_o to start evaluation. Number of buffers $K-1$ between machines K. Number of machines in the line K. Desired throughput f^*. Solution divisor x

output: Buffer vector $\boldsymbol{B}$ resulting in minimum total buffer to achieve desired throughput and throughput of the line $f_{(\boldsymbol{B})}$ and corresponding N.

1 Set total buffer size $N_{current}$ equal to initial buffer size N_0, set iteration counter to i = 0;

2 Run the inner tabu algorithm to obtain the best throughput value, set $f^{best} = f(N_{current})$, and $B^{Best} = B^{N_{current}}$. Place N_0 into tabu list.;

3 Generate 3 possible neighbours of $N_{current}$, called $N_s, s \in \{1,2,3\}$. Neighbour 1 $N_1 = (1/2)(N/x)$ Neighbour 2 $N_2 = (1/4)(N/x)$ Neighbour 3 $N_3 = (1/8)(N/x)$; Evaluate each neighbour with the inner tabu algorithm. ;

4 **if** *All neighbors throughput* $\geq$ *desired throughupt* f^* **then**

5 | **if** *All moves are tabu* **then**

6 | | Select the solution s that has a lower N than the current best, set $f_{best} = f(N_s)$, and $B_{Best} = B^{N_s}$.

7 | **else**

8 | | Move to the solution s having the minimum non-tabu buffer size, set $f_{best} = f(N_s)$, and B_{Best} to B^{N_s} and add all neigbours to the tabu list.

9 | **end**

10 **else**

11 | Move to the solution having the minimum total buffer size that has a throughput $\geq f^*$ and add to the tabu list.

12 **end**

13 Increase the iteration by 1 and go to step 1 until one of the termination criteria is satisfied.;

14 Termination criteria:

- No better solution found within a certain number of iterations.
- Number of iterations reaches the maximum limit.

3.3 Solving the BAP using Simulation-based optimisation (SBO)

Recall that the problem being investigated in this dissertation is to find the smallest total buffer size required to achieve the desired throughput as well as the optimal buffer configuration needed to achieve this. The solution approach described above uses SBO as the method to achieve this.

The proposed SBO method has three main components, shown in Figure 3.8. Each component is evaluated in Chapter 4 for solution quality and execution time.

The outer tabu loop requires accurate maximum throughput results from the inner tabu loop to compare the various total buffer size *N*. The Theory of constraints tabu search (TOCT) method proposed in this dissertation is tested in Chapter 4 and compared to the ATS in Demir et al. [9]. The inner tabu loop needs to evaluate various buffer permutations to find the maximum throughput possible with the provided total buffer size *N*. Each permutation is evaluated by the proposed simulation program based on DES; the program is verified in Chapter 4.

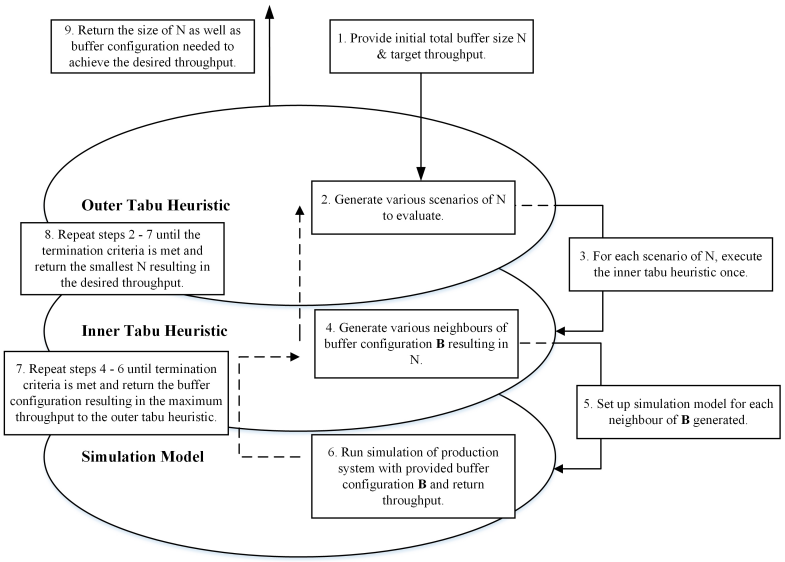


Figure 3.8: Interaction between elements of the SBO method.

A specific solution approach for the BAP has been developed in this Chapter. The use of DES to design a simulation in Java specifically for the BAP problem addresses the computational burden of generic simulation models. Two events are created for the model, a departure event and failure events. These events contain the logic followed by the simulation when a part is completed or a station fails. The works of Demir et al. [9, 11] establishes a solid foundation for the use of tabu search in a two loop manner to solve the BAP. Experiments on the inner tabu loop with the simulation model in Chapter 4 will show another opportunity for improvement by exploiting theory of constraint. Serial lines will first be used in Chapter 4 for proof of concept. In Chapter 5 the proposed solution approach will be used in a more complex tree-structured production line.

Chapter 4

Computational experiments

The proposed Simulation-based optimisation (SBO) method for solving the Buffer allocation problem (BAP) is tested in this chapter. Each of the SBO components, *outer tabu loop*, *inner tabu loop* and *simulation program* is tested to demonstrate the effectiveness of the proposed solution. To achieve this, the method will be tested on serial production lines, using different line sizes, buffer configurations and line parameters.

Demir et al. [9] used four line sizes with the number of machines denoted by the set $K = \{5, 10, 20, 40\}$. For each line, the total buffer size N is calculated by multiplying the number of machines by a factor of 5, 10, and 20. Thus 12 different serial line scenarios are used, representing lines ranging from small to large as shown in Table 4.1. Lines with

Table 4.1: Serial line scenarios for SBO experiments [9].

Number of machines in line, (K)	**Total buffer size, (N)**
5	25, 50, 100
10	50, 100 ,200
20	100, 200, 400
40	200, 400, 800

five machines are considered small, while those with 10 machines are considered medium in size. Lines with 20 and 40 machines are regarded as large lines.

Each scenario will be referred to by its number of machines (K) and total buffer size (N). For example, 5.25 will refer to a line with five machines and the total buffer size of 25. As per Demir et al. [9], the range of line scenarios allows the effectiveness of the proposed method to be tested across various line sizes and therefore, will also be used in this dissertation.

The allocation of buffers is done to decouple stations, preventing the negative effect of different machine speeds, and machine failures on the downstream system. The proposed solution should be able to solve lines that have short or long processing times, lines with a short or long time between failures, as well as lines that are either quick to repair or that take long to repair. For each line scenario, (e.g. a line with five machines and total buffer size of 25 buffers) the various machine characteristics stated above are tested. Again the work of Demir et al. [9] is referred to. The following section explains how Demir et al. [9] defined the random process time, time between failure and repair time for each machine.

Process time: the time it takes to complete a part. The processing time is randomly sampled from a uniform distribution, $U(a, b)$. Each machine's process time will first

be described with parameters, $a = 5, b = 15$. Then experiments will be done on the line with parameters, $a = 5, b = 45$. The distribution for each machine is the same.

Time between failure: each machine is subject to failure. The time between failure is randomly sampled from a geometric distribution $G(p_K)$. Unlike the processing time, each machine in the line will have a unique distribution, some machines are more reliable than some. Each machine has a unique parameter p for the geometric distribution. This parameter is randomly sampled from a uniform distribution, $p_K = \frac{1}{U(a,b)} \forall K$ to give each machine a unique distribution. Two different parameters are considered for the uniform distribution. First the parameters are $a = 1, b = 200$ then $a = 1, b = 2000$.

Repair time: each machine has a repair time once it fails. The repair time is randomly sampled from a geometric distribution $G(p_K)$. Similar to the failure time, each machine in the line will have a unique parameter for the geometric distribution. Some machines are easier to repair than others. This parameter is randomly sampled from a uniform distribution $p_K = \frac{1}{U(a,b)} \forall K$. Four different parameters are considered for the uniform distribution. For lines that have failure parameters $U(1, 200)$ repair parameters are $a = 1$ and $b = 10$, then $a = 1$ and $b = 40$. Lines that have failure parameters $U(1, 2000)$ repair parameters are $a = 1$ and $b = 100$, then $a = 1$ and $b = 400$.

Table 4.2 shows the eight different line parameters used by Demir et al. [9]. This dissertation will also use these parameters. 96 unique line scenarios are considered in the

Table 4.2: Serial line machine parameters for SBO experiments [9].

Instance	Processing time	Time between failure	Repair time
1	U(5,15)	G(1/U[1,200])	G(1/U[1,10])
2	U(5,15)	G(1/U[1,200])	G(1/U[1,40])
3	U(5,15)	G(1/U[1,2000])	G(1/U[1,100])
4	U(5,15)	G(1/U[1,2000])	G(1/U[1,400])
5	U(5,45)	G(1/U[1,200])	G(1/U[1,10])
6	U(5,45)	G(1/U[1,200])	G(1/U[1,40])
7	U(5,45)	G(1/U[1,2000])	G(1/U[1,100])
8	U(5,45)	G(1/U[1,2000])	G(1/U[1,400])

experiments. Each of the 12 line scenarios in Table 4.1 are tested with the eight machine characteristics given in Table 4.2.

The SBO approach proposed in this dissertation has three elements; *outer tabu loop, inner tabu loop* and *simulation program*. Each of these elements is tested in this chapter. The outer and inner loop uses the throughput of the line to measure their objective functions. Thus an accurate throughput for a given buffer configuration is needed. The simulation program is tested in the next section. The execution of the experiments is done on a computer having a 3.80Ghz Intel Core i5-7600K processor and 8 GB of RAM.

4.1 Simulation program verification

Recall from Chapter 3 that a simulation program based on Discrete event simulation (DES) is used in this SBO approach. This is used instead of commercially available simulation

software to reduce the computational time for running the simulation.

Before any model and its results can be used to solve the BAP, the simulation program needs to be tested to establish if the program is executing the simulation as expected. This dissertation's program is compared to a commercially available program to test similarity of results.

Only 10 of the 12 line scenarios in Table 4.1 are modelled in the commercial program due to licensing constraints. Scenarios 40.400, 40.800 are not considered. For each line scenario, three different values for each machine's failure and repair time are generated based on the parameters in Table 4.2. Thus each of the 80 scenarios is tested three times for a total of 240 tests. For these tests, a random buffer configuration $\boldsymbol{B}$ is generated for each total buffer size N.

Simio$^{\text{TM}}$ is a widely used commercial simulation software. Both DES and *Simio*$^{\text{TM}}$ is run for a duration of 300 000 time units to ensure a steady state is reached. The simulation is replicated 1 000 times to get 1 000 different throughput results. At the end of the simulation run, the throughput is recorded, and the average results across the 1 000 replications used to compare the DES program's results to those of *Simio*$^{\text{TM}}$. Secondly, the computational time of the programs model using DES is compared against that of *Simio*$^{\text{TM}}$ to see if a simulation in Java can outperform a commercial program.

The % throughput error between the *Simio*$^{\text{TM}}$ line and DES is compared using the calculation in equation (4.1), while the time factor is calculated using equation (4.2):

$$\frac{\textit{Simio}^{\text{TM}}\text{ throughput} - \text{DES throughput}}{\textit{Simio}^{\text{TM}}\text{ throughput}} \times 100 \tag{4.1}$$

$$\frac{\textit{Simio}^{\text{TM}}\text{ computational time}}{\text{DES computational time}} \tag{4.2}$$

Table 4.3 shows the results of the test. The first column shows the problem scenario, $(K.N)$. The second column shows the average throughput difference between the DES

Table 4.3: Simulation results from Java tool compared to *Simio*$^{\text{TM}}$ program.

Problem scenario	Throughput diff. (%)	Time factor
5.25	0.60	74
5.50	0.27	94
5.100	-0.48	89
10.50	1.03	67
10.100	0.54	79
10.200	0.21	65
20.100	2.30	53
20.200	0.35	60
20.400	0.01	59
40.200	2.60	51

tool and the *Simio*$^{\text{TM}}$ program calculated with equation (4.1). The third column shows the time difference between the two programs calculated with equation (4.2). The DES simulation program created in Java has on average a 0.84% difference in total throughput compared to *Simio*$^{\text{TM}}$. This difference is small and is due to the stochastic nature of simulation. Figure 4.1 supports this. The figure shows the density plot of the throughput

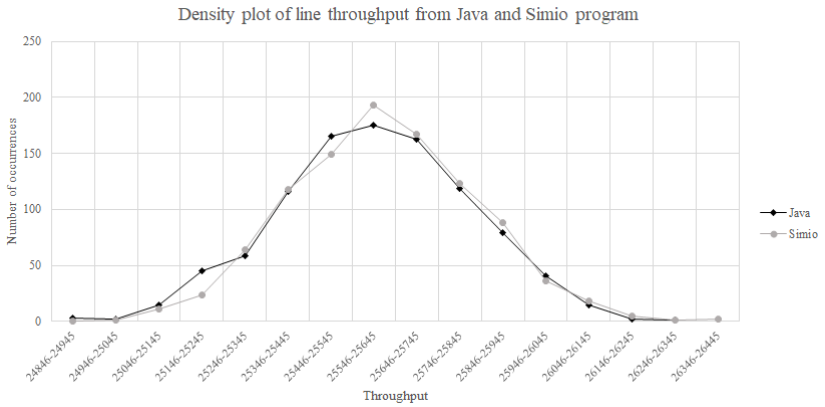


Figure 4.1: Simulation throughput comparison between Java and *Simio*™ program.

obtained from the 1 000 replications for line scenario 10.200. The two programs result in a density plot with similar shapes.

DES had a significant speed advantage compared to *Simio*™. On average, *Simio*™ takes 71 times longer than the proposed program based on DES. For the small-sized problems, *Simio*™ took 23 minutes to complete the simulation run and replications and 75 minutes for large-sized problems. In comparison, the DES only took 19 seconds for small problems and 1.5 minutes for large problems.

The experiment above verified that the simulation program using DES can obtain a throughput similar to that of commercial software. DES is also faster than commercial software. From this point, all simulations used in experiments are done using the Java simulation program created for this dissertation using DES. The simulation tool will be used by the inner tabu loop to determine the throughput of a line for a given buffer configuration. The inner loop will generate thousands of permutations that need to be tested. Thus it is crucial that the simulation achieves an accurate throughput in the shortest amount of time.

Two factors influence both the simulation's computational time and the quality of the solution it provides: the *simulation length* is the amount of time the simulation model uses to test the line, and the *simulation replications*, which is the number of times the simulation is repeated for each problem instance.

The simulation model in Java is quite fast. Still, if it takes 1.5 minutes for every evaluation during the generative method, the time required to solve the BAP can be excessively long. As the solution quality is time-dependent, a balance is needed between solution quality, simulation length and the number of replications.

4.1.1 Simulation length at steady state

Simulation *steady state* is a state at which the value of the throughput has acceptable small fluctuations over time. At the start of the simulation, throughput can be relatively fast because all the buffers are empty and all the machines are operational. As time progresses, buffers start filling up, and machines begin to fail. This can lead to a different throughput compared to the one achieved at the start of the simulation. Thus solution quality is dependent on the simulation length.

To find this point for the simulation model, all the possible scenarios in Table 4.2 are tested across all the line sizes in Table 4.1. Each scenario is evaluated for a simulation length of 1 000 000 time units. At each time unit, the throughput of the line is compared

to the average value the line can achieve at time 1 000 000. Figure 4.2 shows the results for one of the lines. The bottom graph shows it for timeline 0–1 000 000 time units, and the top graph is an enlargement of the portion at 0–10 000 time units.

At time 1 000 000 the line completed 59 198 units, achieving an average throughput of $\frac{1\,000\,000}{59\,198} = 16.89$ time units per unit. The average time unit per unit is calculated every time a unit is completed and compared to the average of 16.89 time units per unit; this is plotted against the simulation length on Figure 4.2. It is clear that at the start of the simulation, the throughput has a significant variation from the 16.89 time units per unit. During this stage it takes more time to produce units. This variation decreases as the simulation length is increased, with a simulation length of 40 000 the simulation has 0% deviation from the average of 16.89 time units per unit.

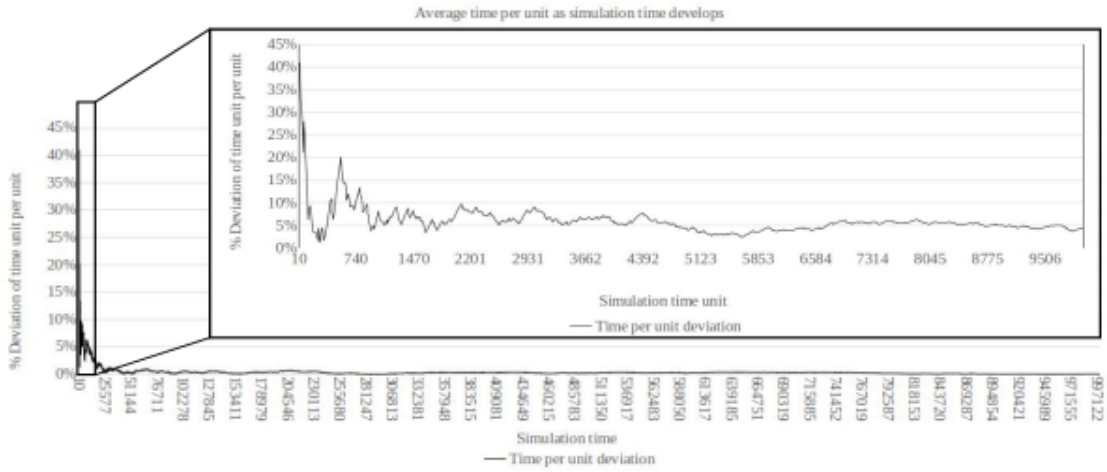


Figure 4.2: Throughput stability development over simulation length.

The simulation computational time is dependent on the simulation length. To reduce the computational time, a 5% throughput deviation will be considered as *acceptable*. In the above example at 10 000 time units, the line remains under the 5% deviation target. At this simulation length, the average time units per unit is 17.61 time units.

4.1.2 Simulation replication

The second factor contributing to simulation run time and solution quality is the number of replications.

Take a simulation of a line with five machines and a total buffer size of 25. A simulation is done with a simulation length of 10 000 time units. At the end of the simulation run, 193 units are produced, or an average of 52 time units per unit. The same simulation is repeated, this time, a total of 355 units are produced, or an average of 28 time units per unit. Figure 4.3 shows that both simulations reached a steady state at two different average throughput values per time unit.

With each replication the initial system state can vary due to the random parameters used. To obtain a stable throughput from the simulation, multiple replications of the same simulation, each with a unique random seed, are done, and the throughput of each simulation is recorded. An average throughput across all simulations is then used to reduce this variation. This can be done because each replication is independent. The number of replications performed is directly correlated to the accuracy of the average statistic. Again, a large number of replications comes at the expense of execution time.

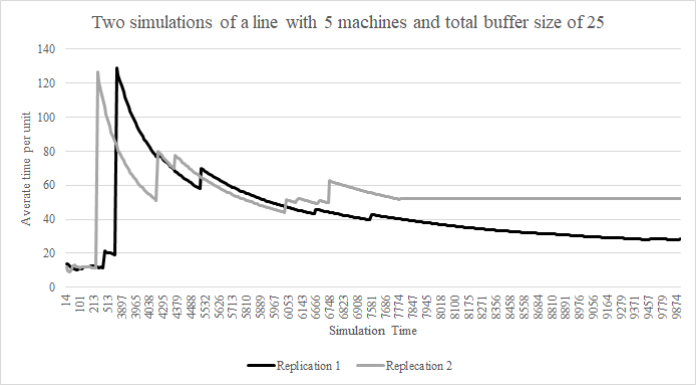
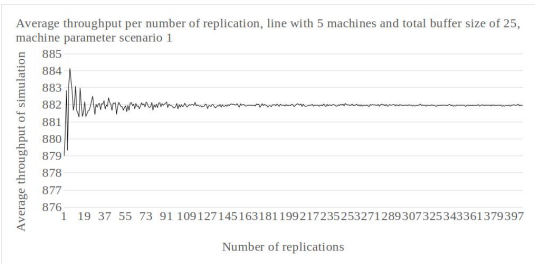
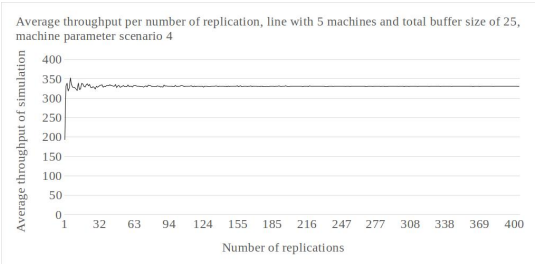


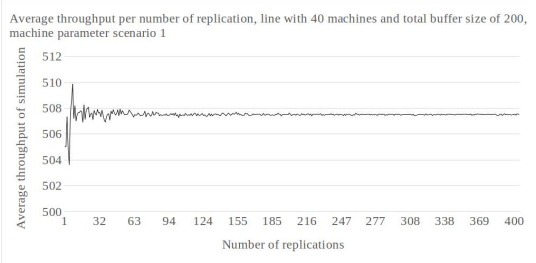
Figure 4.3: Throughput stability of two separate simulation runs of the same line.



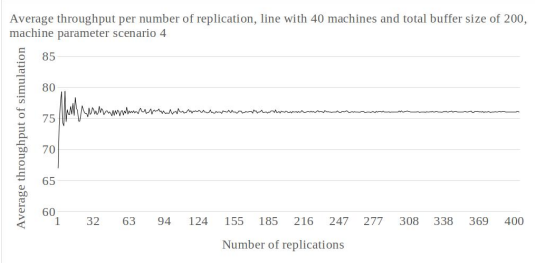
(a) Replication experiment, 5.25 parameter scenario 1.



(b) Replication experiment, 5.25 parameter scenario 4.



(c) Replication experiment, 40.200 parameter scenario 1.



(d) Replication experiment, 40.200 parameter scenario 4.

Figure 4.4: Simulation replication experiments.

To find a *good* number of replications, simulations of the problem scenarios were done, each replicated 1000 times. Figure 4.4 shows the development of the average throughput as the number of replications increases. Four various line scenarios are shown in Figure 4.4. Figure 4.4a is a simulation of a line with 5 machines and the total buffer size of 25. Machine parameters of scenario 1 were used. Figure 4.4b is also a 5.25 line scenario but with parameter scenario 4. Figure 4.4c is a line with 40 machines and total buffer size of 200. Machine parameter scenario 1 was used. Figure 4.4d is also a 40.200 line scenario but with parameter scenario 4.

Initially, the average throughput varies significantly from 1 to 100 replications and then decreases to a small variation concerning the total throughput. Although more replications do result in less variation in the average throughput, at 200 replications, the variation is already minimal compared to the total throughput achieved by the line. For the line shown in Figure 4.4a the throughput at 200 replications is only 0.01% different from the throughput at a 1000 replications.

The experiments above showed that the accuracy of the line's throughput is dependent on both simulation length and simulation replications. The longer the simulation length and the higher the number of replications, the more accurate the throughput obtained is. This is crucial for the inner loop. When two buffer scenarios are compared, it should be able to accurately say that a change in the throughput was due to the change in the buffer configuration and not due to natural throughput variation within the simulation model. Unfortunately, the increase in simulation length and number of replications come

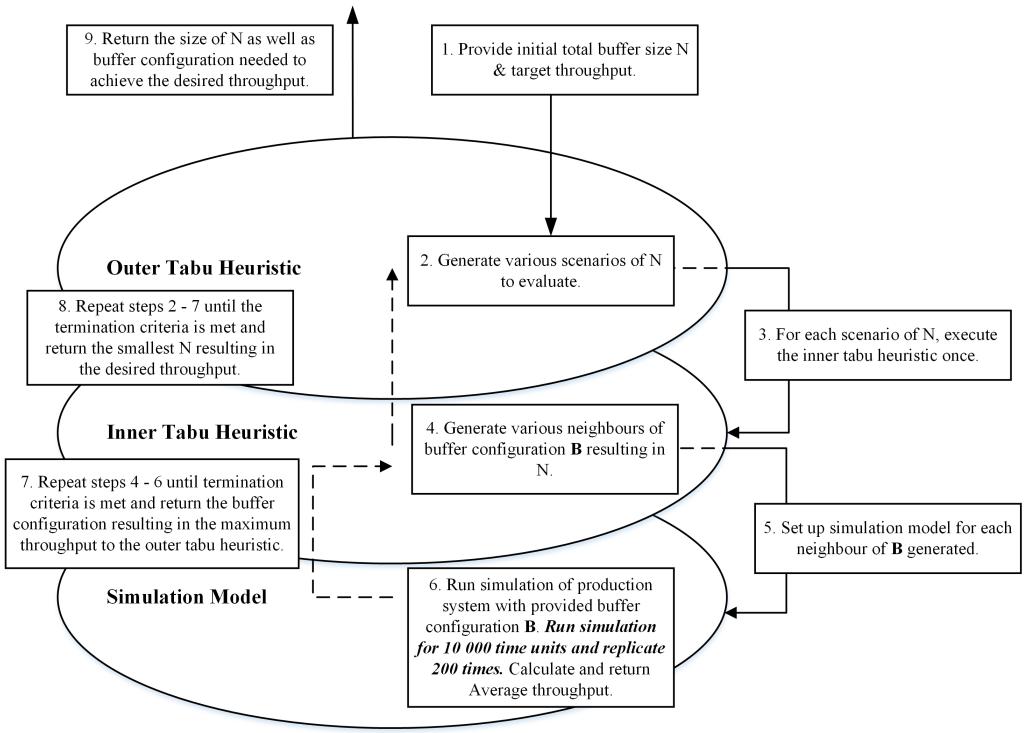


Figure 4.5: Interaction between elements of the SBO method with simulation program run length and replication info.

at the cost of computational time. To minimise the simulation computation time, for all experiments, a simulation duration of 10 000 and 200 replications will be used. This will result in a simulation steady-state error of below 5% and a small fluctuation in average throughput across the number of replications.

Considering the results from the simulation program experiments, the SBO approach shown in Figure 3.8 can be updated. Step 6 is amended in Figure 4.5. The inner tabu tests various buffer neighbours. To compare the neighbours, the throughput of the line is determined using the simulation program. The simulation will be done for a length of 10 000 time units and replicated 200 times. The average throughput across the 200 replications is returned to the inner tabu loop as the average throughput for the specific buffer configuration.

4.2 Inner tabu algorithm

The previous section showed that the proposed method for creating and simulating serial production lines using the program designed in Java is comparable to a commercial program and is faster. Recall that once the throughput of the production system can be evaluated, different scenarios of buffer configurations can be used to determine the *best* configuration.

An inner and outer tabu search algorithm is needed to generate the various buffer configurations. For the inner tabu search, the work of Demir et al. [9] is considered in this dissertation, known as the Adaptive tabu search (ATS). In this section, the efficiency of the algorithm to find the optimal buffer configuration is evaluated. The evaluation method will be the DES program created in Java with the simulation length and number of replications as determined in the previous section.

The inner tabu algorithm searches through the solution space until the termination criteria are met and returns the buffer configuration $\boldsymbol{B}$ that resulted in the maximum throughput. Recall that the termination criteria are either after $50 \times N$ iterations or $25 \times N$ iterations without an improved solution. The inner tabu is only executed once for every N provided by the outer tabu. Thus the inner tabu must be *repeatable*: various runs of the same experiment should not result in (too) different optimal solutions.

Once again the eight line scenarios in Table 4.1 and the 12 machine parameters in Table 4.2 are used. A total of 96 experimental combinations are generated.

Because the inner tabu loop must be repeatable, each of the 96 experiments will be repeated ten times. This is done to determine whether the inner loop was able to achieve the same result for an experiment across all ten replications.

The 12 problem sets are grouped into three classes: small (K = 5), medium (K = 10) and large (K = 20, 40). For the small-sized problem, the efficiency of the algorithm is compared against Complete enumeration (CE). However, for medium and large scale problems CE is not computationally feasible and won't be tested using CE.

The experiments were done on the *Lengau* cluster of the Centre for high performance computing (CHPC), South Africa, parallelising the tasks over 261 threads.

4.2.1 Results from small-sized problems

Table 4.4 shows the results from the experiments on small problems. The first column shows the problem set, shown as machine and buffer size, $(K.N)$. The second column shows the experimental scenario from Table 4.2. The third column shows the % error of the average optimal (maximum) throughput achieved, compared to the solution found using CE. The following formula is used to calculate the % error.

$$\text{Deviation for ATS} = \left(\frac{f(CE) - f(ATS)}{f(CE)} \right) \times 100 \tag{4.3}$$

For each of the ten replications the absolute % error is calculated and averaged for each experiment scenario. Column four show the average computational time of the ATS algorithm in minutes. This is the time it took one iteration of the inner loop to complete. Column five show the average number of iterations it took the algorithm to find the best throughput it returned. All entries in the table are the averages across the ten replications of each scenario.

Experimentation on small lines shows that the ATS method is capable of returning a throughput similar to the global optimum achieved using CE. The absolute error for ATS in caparison to complete enumeration was 0.30%. This small fluctuation can be attributed

Table 4.4: Inner tabu computational results for small sized problems ATS vs. CE.

Problem set	Scenario	Avg. % error from optimal solution	Avg. computation time of algorithm (min.)	Total iterations until optimal solution
5.25	1	0.06	8.28	145
	2	0.44	8.11	158
	3	0.63	7.31	145
	4	0.71	6.14	134
	5	0.20	7.08	198
	6	0.62	5.30	178
	7	0.69	4.51	129
	8	1.41	4.03	163
5.50	1	0.06	18.30	461
	2	0.10	18.84	263
	3	0.11	16.22	374
	4	0.18	15.72	291
	5	0.08	11.36	328
	6	0.10	10.70	722
	7	0.20	9.90	259
	8	0.21	8.08	267
5.100	1	0.04	27.62	713
	2	0.18	23.60	1274
	3	0.10	26.09	807
	4	0.72	18.29	486
	5	0.16	21.24	1010
	6	0.09	19.31	1356
	7	0.07	16.61	1035
	8	0.14	16.25	435

to the inherent stochasticity of the simulation, and in some instances, the method resulted in a better throughput than the CE.

The computational time to do all the experiments (24 scenarios tested 10 times) for the small-sized problems using ATS took 55 hours, excluding the time for the CE tests. Initial tests on medium-sized problems show that the time for a single experiment increases by a factor of ten. Although the quality of the solution from the proposed solution approach looks promising, the computational time makes the current approach impractical.

Recall that the ATS generates and evaluates all possible neighbours of the initial buffer configuration $\boldsymbol{B}$ during each iteration of the tabu search. Although the approach is very thorough, it does increase the number of times the simulation program needs to run. Due to the time required to evaluate a scenario using simulation, the initial tests proved that the computation times for medium and large-sized problems became unreasonable and impractical. The tests also showed that very bad scenarios are tested in complete neighbour generation. Buffers are removed from stations that are blocked and added to stations that are never blocked, resulting in a worse throughput and a waste of simulation time.

To overcome this an alteration to the work of Demir et al. [9] is proposed for neighbour generation. A smaller set of *good* neighbours are generated. This is achieved using the *theory of constraints*. It states that the performance of a line is limited by a specific station, the slowest one, known as the *bottleneck*. If the overall performance of the line is to be improved, this single station needs to be improved first. This bottleneck can be identified with reports from the simulation.

The simulation model provides statistics on how long each station was blocked during the simulation. It is thus possible to allocate weight to each buffer indicating how much time it blocked its upstream station. The longer the station is blocked, the higher the weight allocated to the buffer. The swap index is then randomly generated, where the buffer that has the highest block weight has a higher probability of being picked, and the buffer with the lowest block weight has the lowest probability of being picked. This station's buffer is increased. The buffer that will decrease in size is also chosen randomly, where the buffer that has the smallest block weight has the highest probability of being chosen and the buffer with the highest block weight the lowest probability. Only $0.5\times$(Number of machines K) neighbours are generated per iteration using the proposed method. This dramatically decreases the simulation burden. The factor 0.5 is an arbitrary value.

Another alteration to the heuristic is the inclusion of a list, storing the throughout for each evaluated buffer configuration. After generating new neighbours, the generated neighbours are compared to the items in this list to see if it has been evaluated previously. If it has been evaluated, the previously achieved throughput can be used, and the scenario does not have to be re-evaluated. The rest of the algorithm is the same as the work of Demir et al. [9]. The proposed algorithm is referred to as the Theory of constraints tabu search (TOCT) inner loop, whereas the model by Demir et al. [9] is referred to as ATS.

The experiments on small-sized problems are repeated, using the new proposed TOCT method. The following formula is used to calculate the % error.

$$\text{Deviation for TOCT} = \left(\frac{f(CE) - f(TOCT)}{f(CE)}\right) \times 100 \tag{4.4}$$

Similar to the ATS method, the TOCT is capable of achieving throughput similar to the global optimum achieved using CE. The absolute error for TOCT in comparison to CE was 0.88% versus the 0.30% for ATS.

The complete and detailed throughput results for each of the ten replications for each scenario is shown in Appendix A. Suffice to conclude that the results are tightly grouped (between maximum throughput achieved and line scenario), indicating that a single run of the inner tabu is sufficient. Multiple inner tabu runs would have achieved the same as the multiple replications, albeit slower.

The significant difference between TOCT and ATS is computation time. The ATS method took, on average, 18 times longer to execute.

The third performance comparison is the number of iterations the program took to reach the solution it returned. For these small sized problems, $K = 5$, various total buffer sizes are tested, $N = 25$, $N = 50$, $N = 100$. The inner tabu is terminated if one of the two termination criteria are met. Either after $N \times 50$ iterations, termination criteria 1, or after $N \times 25$ iterations without an improvement on the optimal solution, termination criteria 2. Table 4.6 shows the stipulated number of iterations required to meet the termination criteria for the different scenarios. The actual number of iterations at which point the best throughput found does not improve for the ATS method is shown in column five in Table 4.4. Column five in Table 4.5 shows the number of iterations at which point the best throughput found does not change for the TOCT method.

Table 4.5: Inner tabu computational results for small sized problems TOCT vs. CE.

Problem set	Scenario	Avg. % error from optimal solution	Avg. computation time of algorithm (min.)	Total iterations until optimal solution
5.25	1	0.02	0.51	502
	2	1.64	0.36	350
	3	0.26	0.29	415
	4	0.34	0.32	329
	5	0.52	0.23	433
	6	1.11	0.20	315
	7	0.33	0.28	378
	8	0.98	0.17	450
5.50	1	0.33	0.50	858
	2	2.60	0.80	1630
	3	0.61	1.42	1425
	4	1.52	1.86	858
	5	0.04	0.46	933
	6	0.14	0.53	1031
	7	0.11	0.85	879
	8	3.75	0.60	879
5.100	1	0.09	0.72	1119
	2	0.79	0.71	1335
	3	0.21	2.34	1635
	4	3.36	2.18	1064
	5	0.52	0.78	1645
	6	0.43	0.48	1262
	7	0.37	0.44	1949
	8	1.11	0.90	1538

Both the ATS and TOCT found, on average, the optimal solution in fewer iterations than the maximum allowed iterations. After that point, the algorithm kept running until the termination criteria were met but could not find a better solution. The ATS needed fewer iterations compared to TOCT because it can find a better solution per iteration due to complete neighbourhood generation, but it takes much longer to do a single iteration. The termination criteria can thus be revised if the same results are found for medium and large-sized problems.

Table 4.6: Inner tabu termination criteria for small-sized problems.

Problem set	Termination criteria 1	Termination criteria 2
	$(N \times 50)$	$(N \times 25)$
5.25	1250	625
5.50	2500	1250
5.100	5000	2500

4.2.2 Results from medium-sized problems

Complete enumeration is not possible for medium and large-sized problems. For the medium sized problems, the throughput achieved using ATS and TOCT is compared using equation (4.5), which shows the calculation used to determine the throughput deviation between the two methods.

$$\% \text{ throughput deviation between ATS and TOCT} = \left(\frac{|f(TOCT) - f(ATS)|}{f(TOCT)} \right) \times 100. \tag{4.5}$$

Table 4.7 shows experimental results for the medium sized problems. The first column shows the problem set for the medium sized line, and the scenario in the second column. For each of the ten replications, the % throughput deviation is calculated using equation (4.5); the average result is shown in the third column. Columns 4–5 are the average computational time for the two methods and columns 6–7 the average number of iterations it took the tabu algorithm to reach its optimal solution.

For experiments on lines with 10 machines and total buffer size 200, only two of the eight scenarios are tested using the ATS method due to long execution times.

The average error between the ATS method and TOCT method is 1.68%. The ATS method takes 5.5 times longer to complete compared to TOCT.

Similar to small-sized problems, the throughput achieved across the ten replications are tightly grouped. The complete and detailed throughput results for each of the ten replications for each scenario are shown in Appendix A. The third performance comparison is the number of iterations the program took to reach the solution it returned. The same termination criteria apply. The inner tabu is terminated if one of the two termination criteria are met. Either after $N \times 50$ iterations, termination criteria 1, or after $N \times 25$ iterations without an improvement on the optimal solution, termination criteria 2. Both ATS and TOCT found, on average, the optimal solution at fewer iterations. After that point, the algorithm kept running until the termination criteria is met but could not find a better solution.

Table 4.7: Inner tabu computational results for medium sized problems ATS vs. TOCT.

Problem set	Scenario	% Deviation between ATS & TOCT	Computation time of algorithm (min.)		Total iterations until optimal solution	
			ATS	TOCT	ATS	TOCT
10.50	1	-1.11	112.20	11.40	419	639
	2	-0.71	56.53	3.43	367	763
	3	-4.59	76.13	3.84	443	1135
	4	-6.44	36.65	2.38	352	1046
	5	-0.33	76.69	11.09	314	680
	6	-0.74	77.23	12.05	382	506
	7	-0.53	66.17	9.09	426	1144
	8	-1.80	57.90	8.48	285	744
10.100	1	-0.20	270.18	39.20	433	883
	2	-1.33	249.75	57.43	871	2078
	3	-1.20	259.67	50.49	853	2245
	4	-6.62	199.22	31.07	534	1424
	5	-0.46	191.02	26.36	869	919
	6	-0.62	161.65	26.04	696	1737
	7	-0.91	155.50	19.01	602	1045
	8	-2.27	103.49	13.70	685	1758
10.200	1	-0.21	629.54	22.57	2315	2152
	2	-0.07	382.31	56.39	1376	2342
	3	–	–	82.03	–	3714
	4	–	–	45.34	–	2861
	5	–	–	47.52	–	2948
	6	–	–	45.57	–	4508
	7	–	–	53.63	–	3499
	8	–	–	29.44	–	1883

4.2.3 Results from large-sized problems

Large sized problems can only be evaluated using the proposed TOCT algorithm. Due to the size of the problem and the increase in computational time required as the total buffer size and number of machines increase, ATS becomes computationally expensive to test comparatively. Table 4.8 shows the results for the large sized problem with 20 machines as well as 40 machines. Note that due to long computational time, experiments on lines with 40 machines and 400 N as well as 800 N could not be performed.

The detailed throughput results can be seen in Appendix A. For the large sized problems, more outliers are present compared to medium and small-sized problems. Similar to the previous experiments, maximum throughput is achieved within fewer iterations than required by the termination criteria.

The inner tabu loop experimentation showed that the proposed TOCT method could satisfy the BAP's first objective: *finding the buffer configuration* $\boldsymbol{B}$ *that results in the maximum throughput.* For small problems, the TOCT method achieved similar results to CE and ATS within a much shorter time. The reduction in execution time makes it

possible to solve medium sized problems using SBO. Large sized problems with large number of buffer $400N$ as well as $800N$ still require large computational time to solve even with the proposed method.

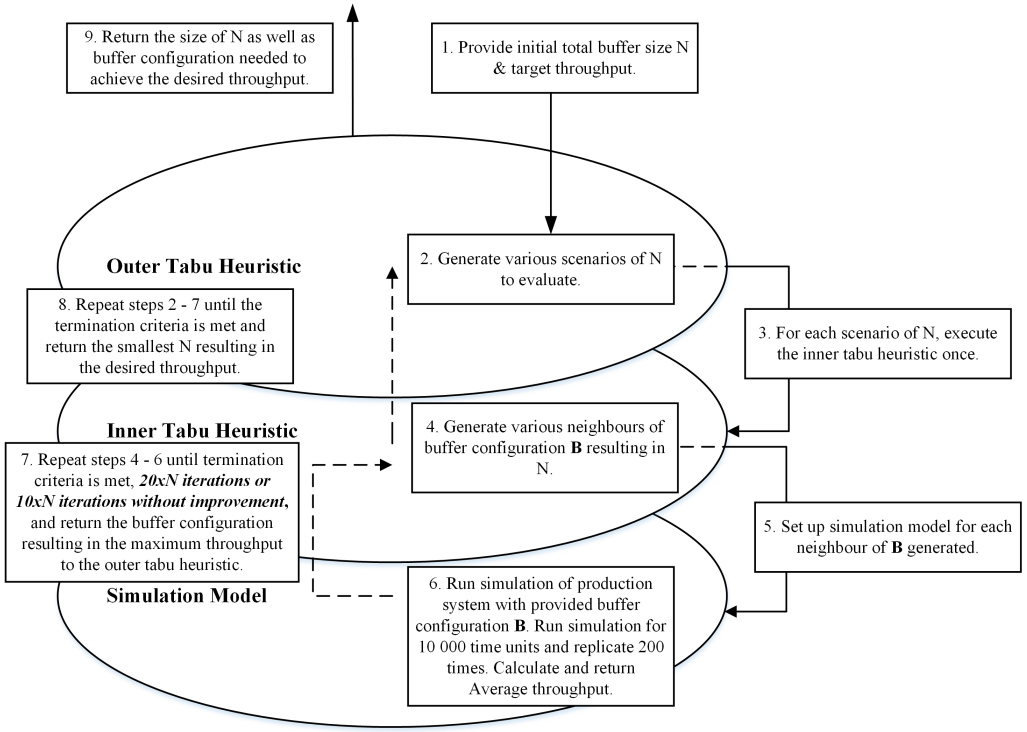


Figure 4.6: Interaction between elements of the SBO method with inner loop termination criteria.

The experiments indicate that in the majority of cases, the maximum throughput was achieved with fewer iterations than currently set up in the termination criteria. For future experiments, the termination criteria for the inner tabu loop is changed to either $20 \times N$ or $10 \times N$ iterations without an improved solution. This can significantly reduce the execution time required to run the inner tabu algorithm and makes it more practical for the use in the outer tabu loop. Replicating each experiment 10 times indicated that the inner loop returned sufficiently similar incumbent (near-optimal) solutions. It is therefore argued that it is safe to run the inner loop once for each N provided by the outer loop as its results are similar to multiple replications. Figure 4.6 shows the flow of the proposed SBO method including the addition of the inner loop termination criteria for step 7.

Table 4.8: Inner tabu computational results for large sized problems, 20 & 40 machines ATS vs. TOCT.

Problem set	Scenario	Throughput	Time (min.)	Iteration
20.100	1	575.33	66.04	1205
	2	119.66	33.20	3309
	3	623.69	89.58	1120
	4	57.39	15.41	3380
	5	295.38	63.19	1148
	6	76.22	20.19	2077
	7	19.12	11.39	2343
	8	101.30	31.31	1219
20.200	1	602.44	99.30	1932
	2	552.28	186.81	1136
	3	470.88	154.52	5558
	4	170.09	82.93	1852
	5	262.47	178.05	2897
	6	199.80	129.33	2453
	7	233.57	121.80	3172
	8	89.13	49.56	3372
20.400	1	670.06	642.68	5215
	2	570.03	621.26	5458
	3	603.30	495.67	1208
	4	333.46	270.36	298
	5	274.24	464.34	3861
	6	191.54	376.49	4280
	7	205.63	286.36	5457
	8	122.19	237.02	5624
40.200	1	373.83	376.54	9004
	2	15.28	57.06	2843
	3	18.26	65.55	4060
	4	14.73	47.92	7257
	5	227.09	462.19	4702
	6	28.47	78.03	3320
	7	21.37	64.60	4315
	8	16.01	51.61	4681

4.3 Solving the BAP using SBO

The final piece of the SBO method is the outer loop. Recall that the main objective of the SBO is to find a buffer configuration, $\boldsymbol{B}$, that results in the smallest total buffer size N while achieving a required throughput. The experiments thus require a target throughput. Similar to the experiments on the inner loop the line scenarios presented in Table 4.1 and Table 4.2 are used. To determine the target throughput, each experiment is done with half the initial N. For example, a line with $K = 5$ machines and $N = 25$ total buffer size,

the inner tabu loop is executed using $\lceil \frac{N}{2} \rceil$, $\lceil \frac{25}{2} \rceil = 12$. For a line with $K = 5$ machines and $N = 12$ total buffer size the inner loop achieved a maximum throughput of 918. This then becomes the target for the outer loop. It will start with the initial total buffer size $N = 25$ and search for the smallest N that results in a throughput of at least 918. It can be that the outer loop only return $N = 12$, or if possible, it could achieve a throughput of 918 with even a smaller total buffer size.

The execution of the experiments is done on a computer having 3.80Ghz Intel Core i5-7600K processor and 8 GB of RAM. For small-sized problems, the experiment was repeated five times. For medium and large-sized problems, the experiment was only replicated once. The CHPC is not used to run these experiments but a desktop. Do to computational resources, only limited number of replications are tested. All entries in the tables are the averages of the number of replications for each problem instance. The results are shown in Table 4.9. The first column is the problem set from Table 4.1. The second column is the problem scenario from Table 4.2. The throughput target the algorithm needs to achieve is shown in the third column. The actual throughput and the corresponding optimal buffer size to achieve this is shown in the fourth and five column respectively. Lastly, the time to solve one instance of the SBO in minutes is shown in the sixth column. For small sized problems the final throughput achieved by the model is very similar to the objective. In ten of the scenarios the model achieved the throughput at $\lceil \frac{N}{2} \rceil$ of the starting N. In 14 of the scenarios, the model achieved the required throughput with less than $\lceil \frac{N}{2} \rceil$ of the starting N.

Table 4.10 show the results for medium-sized problems. Similar to small-sized problems, the model was able to achieve a required throughput close to the target throughput. Again in all scenarios, the required buffer to do this was either $\lceil \frac{N}{2} \rceil$, or less than $\lceil \frac{N}{2} \rceil$ of the starting N. 20 scenarios was less than $\lceil \frac{N}{2} \rceil$. Table 4.11 show the results for large sized problems. 21 scenarios was less than $\lceil \frac{N}{2} \rceil$ of the starting N. With the large sized problems, 3 scenarios 20.100.8, 20.200.2 and 20.200.8 was not successfully solved. A higher throughput is achieved with a total buffer size required bigger than $\lceil \frac{N}{2} \rceil$ of the starting N.

In this section its shown that proposed method for creating and simulating serial production lines using the program designed in Java is comparable to a commercial program and is faster. DES model is proven to be quicker than a commercial program. For small-sized problems *Simio*™ took 23 minutes to complete the simulation run and replications and 75 minutes for large-sized problems. In comparison, the DES only took 19 seconds for small problems and 1.5 minutes for large problems. Experiments with the simulation model show that simulation length and replication has an impact on both solution quality and execution time. To minimise the simulation computation time, for all experiments, a simulation duration of 10 000 and 200 replications will be used. This will result in a simulation steady-state error of below 5% and a small fluctuation in average throughput across the number of replications. Initial tests using DES with the ATS inner loop proved that SBO is capable of solving the BAP. However, long execution time makes it unusable for medium and large-sized problems. Two alterations are considered for the ATS and a new inner tabu loop is proposed based on theory of constraints, TOCT as well as a list that saves previously tested buffer scenarios and their throughput so that same scenarios are not re-evaluated with the simulation model. The proposed method is 18 times faster than ATS for small-sized problems and 5.5 times for medium-sized problems. The proposed method is used to solve the BAP for objective two, showing that SBO is an effective model.

Table 4.9: Outer tabu experimental results small sized problems.

Problem set	Scenario	Target throughput	Avg. throughput achieved	Avg. total buffer size	CPU time (min.)
5.25	1	918	918	12	10
	2	626	629	12	8
	3	681	683	12	5
	4	423	425	12	5
	5	294	295	12	6
	6	238	240	12	3
	7	277	278	12	3
	8	190	192	12	2
5.50	1	650	651	12	8
	2	710	714	20	29
	3	843	844	24	36
	4	755	757	24	36
	5	360	360	25	18
	6	270	270	20	13
	7	345	345	25	16
	8	198	199	24	9
5.100	1	940	940	37	54
	2	435	436	40	32
	3	785	786	49	103
	4	345	348	49	36
	5	317	317	43	42
	6	288	288	33	14
	7	325	326	40	15
	8	299	301	38	19

Table 4.10: Outer tabu experimental results medium sized problems.

Problem set	Scenario	Target throughput	Avg. throughput achieved	Avg. total buffer size	CPU time (min.)
10.50	1	552	608	9	35
	2	60	60	11	7
	3	392	392	24	44
	4	31	31	18	9
	5	255	288	9	37
	6	212	224	9	36
	7	280	281	25	81
	8	164	164	24	64
10.100	1	837	837	42	738
	2	684	684	50	842
	3	729	730	50	754
	4	391	392	48	541
	5	276	276	45	466
	6	255	256	47	390
	7	283	283	47	305
	8	100	101	50	121
10.200	1	503	503	50	458
	2	614	615	66	998
	3	650	655	98	1826
	4	368	370	98	922
	5	283	283	43	433
	6	216	217	63	415
	7	214	215	67	473
	8	131	132	96	396

Table 4.11: Outer tabu experimental results large sized problems.

Problem set	Scenario	Target throughput	Avg. throughput achieved	Avg. total buffer size	CPU time (min.)
20.100	1	13	14	47	141
	2	8	5	45	59
	3	9	9	20	241
	4	9	12	16	58
	5	16	16	46	137
	6	12	15	39	30
	7	9	10	16	11
	8	23	98	88	169
20.200	1	601	601	82	618
	2	112	550	124	988
	3	15	15	70	109
	4	124	125	97	1507
	5	211	261	80	710
	6	19	196	80	413
	7	14	216	44	137
	8	32	103	163	219
20.400	1	669	669	63	2885
	2	563	563	133	2615
	3	237	564	157	2217
	4	223	256	182	1437
	5	274	274	161	8171
	6	189	190	99	1006
	7	203	203	188	3652
	8	112	112	175	1811

Chapter 5

Case study on BIW production facility

Body-in-white (BIW) is a production phase in which the automobile's metal body, called the *body in white*, is assembled using preformed pieces of metal. Some of the parts processed during this phase (originally shown in Figure 1.1) is repeated here in Figure 5.1 for reference.

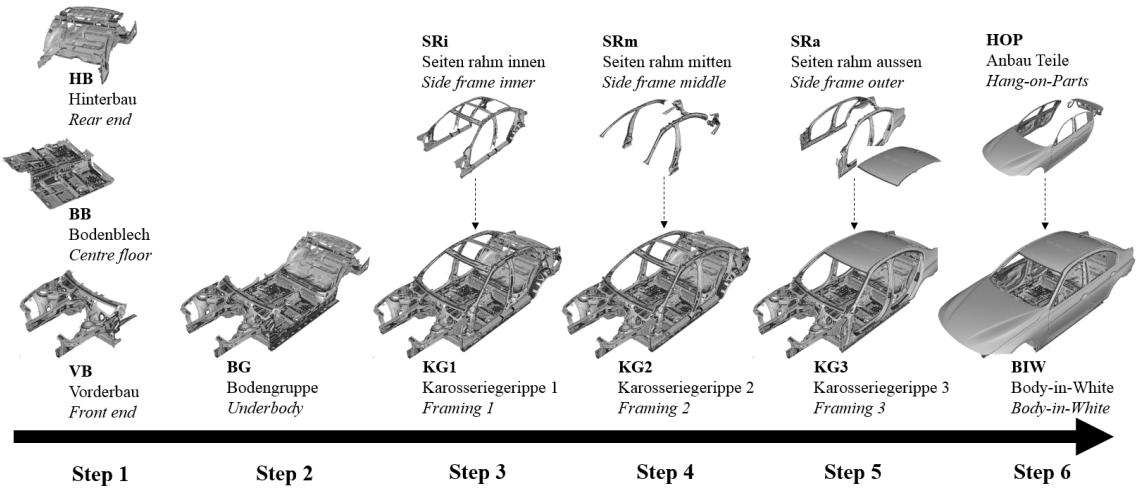


Figure 5.1: Components assembled during BIW process.

Such a line can consists of hundreds of welding robots, the production line this case study is based on has 292 robots. The structure of the body changes with each new model. Consequently, so does the technology used to join the preformed pieces of metal. As a result, a new body shop needs to be designed for each new model, or a major model update.

The body shop manufactures multiple model variants. The lines are subject to failure and are partially unbalanced. Throughput of these manufacturing lines are generally very high and is affected by factors such as variation in processing times and reliability. The impact of these factors on the throughput of the line is reduced by introducing buffers that aims to mitigate idle time due to starving (no input available) or blocking (no space for output). However, allocating buffers into a production line requires additional capital, material handling and is limited by available floor space. Buffers also increase the work-in-process inventory. Due to these factors the strategic placement of buffers on the production

line is an important manufacturing design problem.

The BIW production system under investigation is a BMW production facility, Figure 5.2 and Figure 5.3 illustrates the layout of this production system. The figures show the stations, buffers and material flow for the BIW factory that produces the components in 1.1. Differences should be noticed between the line topology in figures 5.2 and 5.3 compared to a serial line. The topology is a tree structure, various machines have more than one station feeding parts into them. Figure 5.2 is the first part of the factory. The three main sub-assemblies are produced in separate areas, the *Front End*, *Centre Floor* and *Rear End*. The *Front End* has seventeen production cells with unique station codes. To conform with the BAP notation, each of these production cells will be simulated and referred to as a single *machine K*. Each of these machines are separated with a small buffer. The *Centre Floor* has nine machines while the *Rear End* has twelve. Each machine has a unique processing time, which can be simulated with a uniform distribution. Each machine also has unique failure and repair times. Not only do the distributions of the various machines' failure and repair rate differ, but also its parameters. Some are modelled using Gamma, some using Beta distributions. The topology of the line is also not serial but tree structured.

Figure 5.3 shows the second part of the production facility. The three main sub-assemblies are constructed into one piece and reinforced with additional sub-assemblies as it moves down the main line. Together the total facility has 63 machines, 59 buffer locations and a total buffer size N of 318. Referring to our experiments in Chapter 4, the BIW problem is considered a large-sized problem. The SBO method developed in this dissertation can be used to solve the BAP for complex lines such as this production system. A simulation is created of the line, using its tree topology. Each machine has unique random distributions that represent the processing time, failure rate and repair time, respectively.

The simulation has a simulation length of three days (72 hours). The simulation starts with all buffers empty and a single part in every station. The throughput for the first two days are not recorded as it allows the simulation to reach a steady state. The throughput for the third day is then stored and used for analysis. The simulation is replicated 10 times and the average of the throughput across the ten replications is returned as the performance result for the line.

The BMW plant is measured on daily output performance. Stability is crucial and the line needs to achieve its targets daily. The current buffer allocation and line parameters results in an average daily throughput of 239.7 units per day.

Using the TOCT inner tabu tested in Chapter 4 the optimal buffer configuration resulted in a daily throughput of 282.7 units per day. Reconfiguring the allocation of buffer increased the performance of the line by 43 units from the initial state.

The objective function of the BAP for this dissertation is achieving a specific throughput with the smallest total buffer size. The SBO tested in Chapter 4 is used on the BMW line. A target throughput of 230 units per day needs to be achieved by the line. The SBO method obtained a buffer configuration that resulted in 242.9 units per day with a total buffer size of 126. That is a reduction of 192 units in the total buffer size. The algorithm required 3 hours and 55 minutes to finish. Table 5.1 and Table 5.2 show the buffer allocation for the initial case study versus the optimal buffer allocation. None of the buffers where reduced to 0 thus, the optimal buffer configuration is to have more small buffers among the line than fewer large buffers. Some buffer locations *(B47,B50, B52, B56)* increased in size.

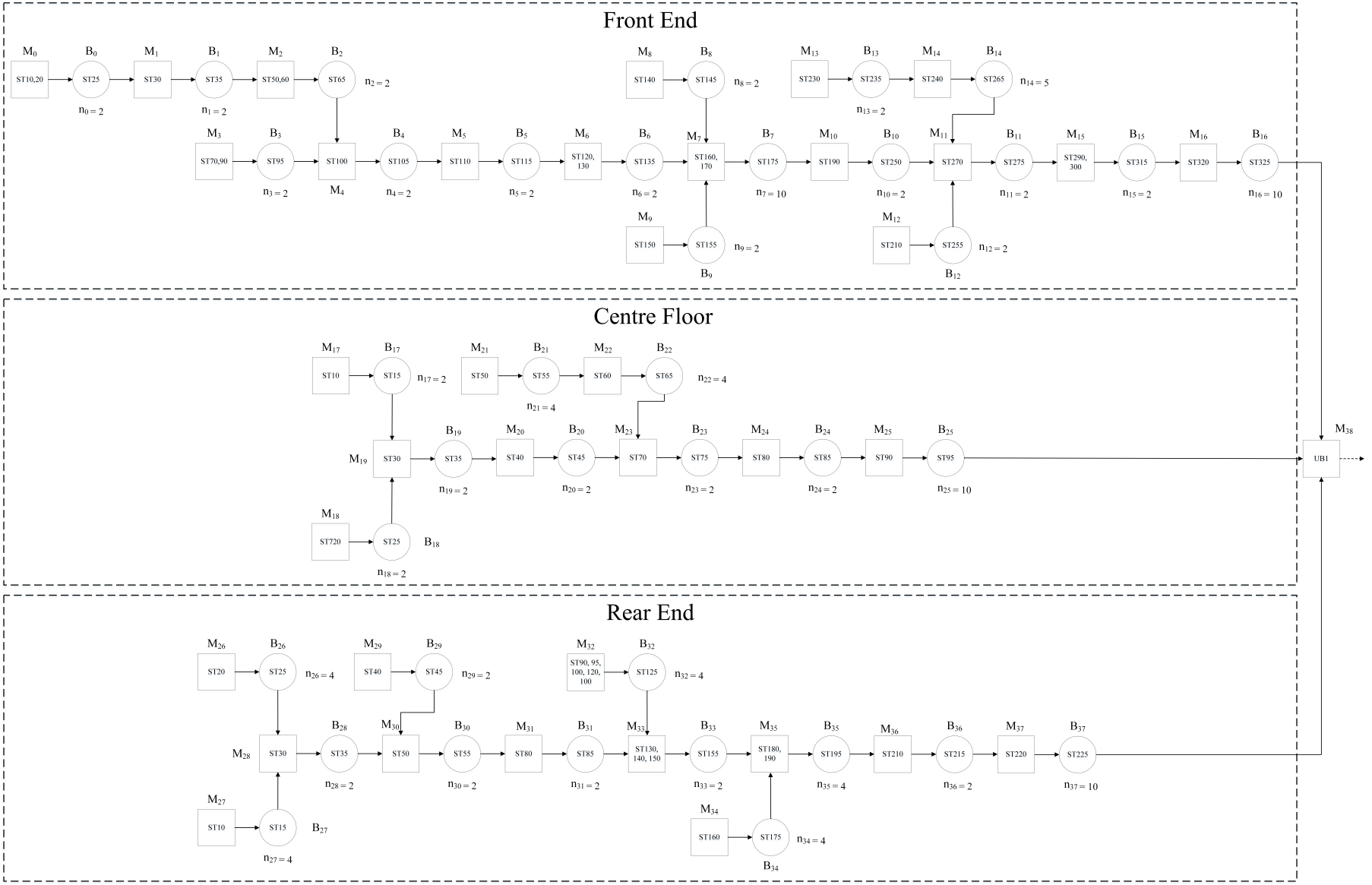


Figure 5.2: BMW production line phase 1.

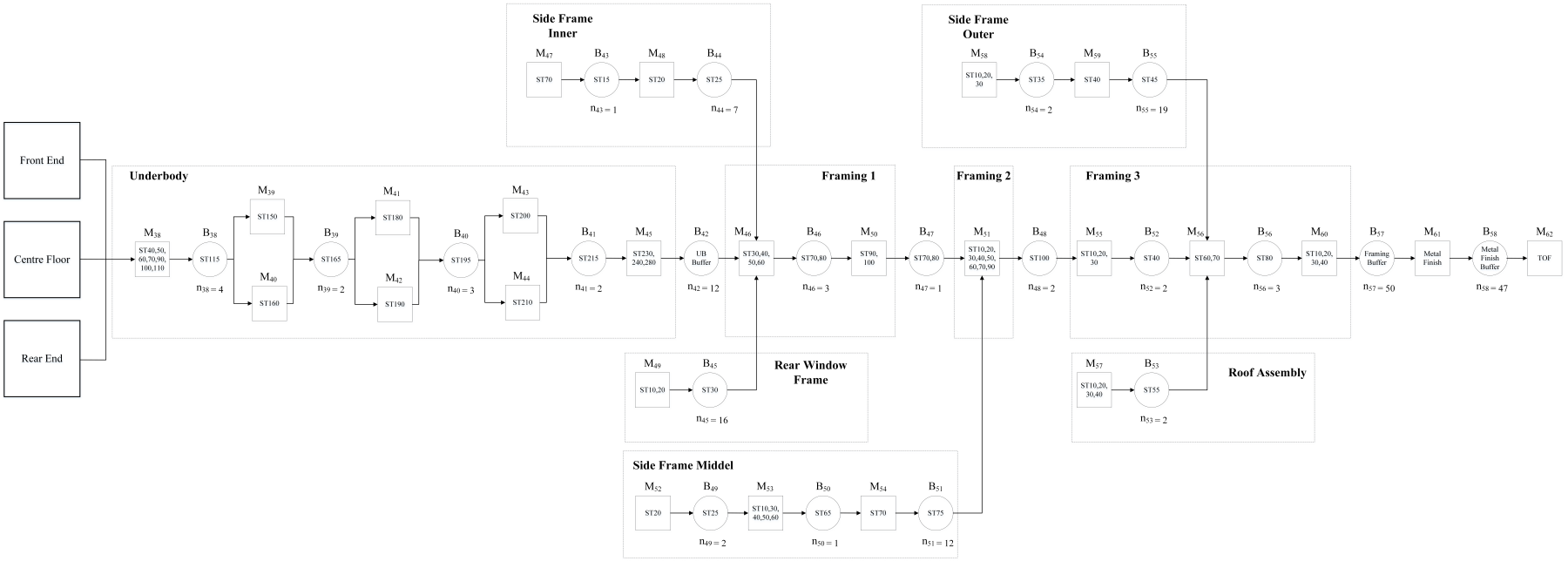


Figure 5.3: BMW production line phase 2.

The case study shows that the proposed SBO method is able to solve the BAP for complex lines such as the BMW BIW production line.

Table 5.1: Buffer allocation results production phase 1.

Front End			Centre Floor			Rear End		
Buffer index	Initial ***Bi*** size	Optimal ***Bi*** size	Buffer index	Initial ***Bi*** size	Optimal ***Bi*** size	Buffer index	Initial ***Bi*** size	Optimal ***Bi*** size
B0	2	2	B17	2	1	B26	4	1
B1	2	1	B18	2	1	B27	4	1
B2	2	1	B19	2	1	B28	2	1
B3	2	1	B20	2	1	B29	2	1
B4	2	1	B21	4	2	B30	2	2
B5	2	2	B22	4	2	B31	2	1
B6	2	1	B23	2	1	B32	4	2
B7	10	1	B24	2	1	B33	2	1
B8	2	1	B25	10	1	B34	4	1
B9	2	1				B35	4	1
B10	2	1				B36	2	1
B11	2	1				B37	10	1
B12	2	1						
B13	2	1						
B14	5	1						
B15	2	1						
B16	10	1						

Table 5.2: Buffer allocation results production phase 2.

Underbody			Framing		
Buffer index	Initial $\boldsymbol{Bi}$ size	Optimal $\boldsymbol{Bi}$ size	Buffer index	Initial $\boldsymbol{Bi}$ size	Optimal $\boldsymbol{Bi}$ size
B38	4	1	B45	16	1
B39	2	1	B46	3	3
B40	3	2	B47	1	3
B41	2	2	B48	2	2
B42	12	2	B49	2	1
B43	1	1	B50	1	2
B44	7	1	B51	12	2
			B52	2	3
			B53	2	1
			B54	2	1
			B55	19	1
			B56	3	4
			B57	50	1
			B58	47	47

Chapter 6

Conclusion

In this dissertation a Simulation-based optimisation (SBO) approach is proposed to solve the Buffer allocation problem (BAP) for unreliable production lines with complex line structures. The objective was to achieve a desired throughput with the smallest total buffer size. Various methods were investigated to model unreliable production systems, and simulation was found to provide the required flexibility to model complex systems accurately.

A new Discrete event simulation (DES) tool was created that can simulate varying sizes of serial lines by adjusting the line size parameters. The DES's performance was compared to that of commercial simulation software. The results of the experiments showed that the proposed simulation model can achieve similar throughput results as the commercial software, but in a much shorter amount of time.

This dissertation built on the foundation laid by Demir et al. [9] whose integrated approach use two tabu loops. To improve its efficiency, this dissertation introduced a neighbourhood generation mechanism, based on the Theory of Constraints. This reduced the amount of neighbours evaluated by the tabu loop for each iteration by generating neighbours that increase the buffer after stations that are blocked and reducing buffer in front of stations that are least blocked. That is, a mechanism that explicitly aims to alleviate the production bottleneck. The performance of the proposed inner loop was then compared to that of Demir et al. [9]. The proposed method using Theory of Constraints had a significant time advantage.

The results of the experiments showed that for small problems the two methods resulted in similar throughout. The results were also compared to that of complete enumeration, demonstrating that both methods are able to achieve global optimal solutions. For medium and large problems this dissertation's implementation of the Demir et al. [9] algorithm could not produce results within a reasonable time.

The outer tabu loop of Demir et al. [9] was used with the Theory of Constraints inner loop and discrete event simulation model to solve the BAP. Experiments showed that the proposed method is able to solve the BAP for unreliable serial lines.

The same tool was then applied to simulate complex tree-structured production lines such as the BMW production system in Rosslyn, South Africa. More specifically, the complex production system of the Body in White facility. The method could accurately simulate this complex line. Results showed an alternative buffer configuration and an increase in daily output. The target throughput could be achieved with a reduction in total buffer size.

6.1 Future opportunities

Future research can be performed on the multi-threading logic used in SBO. Being able to run various instances of the simulation across multiple threads does decrease the required execution time and further improvements on this has great potential. Inclusion of cost into the buffer allocation program will make the model more applicable to industry. Certain areas in the production line can be cheaper to add buffer than other. In the BMW Body-in-white (BIW) case study, it is less capital intensive to store large amounts of the sub-assemblies that is still small in size than having large buffers later in the production line where a complete vehicle needs to be stored in the buffer.

Appendix A

Appendix A

Tables A.1 to A.6 show the maximum throughput achieved by the inner tabu loop. The first column us the problem set, machine size and buffer size. The second column refers to the problem scenario, referring to processing time, machine failure and repair rate. The third column shows the specific inner loop method used, either Complete enumeration (CE), Adaptive tabu search (ATS) or Theory of constraints tabu search (TOCT). The result achieved for each of the 10 replications are shown in fourth to thirteenth column.

Table A.1: Maximum throughput achieved per replication for each scenario and method.

Problem set	Scenario	Method	Replication Number									
			1	2	3	4	5	6	7	8	9	10
5.25	1	ATS	938.44	938.47	938.42	938.79	938.35	938.63	938.55	938.31	938.51	938.34
		TOCT	937.59	937.83	937.89	937.81	937.77	937.39	937.23	937.67	937.72	937.83
		CE	937.89	-	-	-	-	-	-	-	-	-
5.25	2	ATS	697.58	698.40	697.94	697.08	697.73	698.53	697.04	697.05	698.05	696.89
		TOCT	670.03	686.81	682.24	688.50	692.17	677.71	676.51	686.48	678.24	693.05
		CE	694.58	-	-	-	-	-	-	-	-	-
5.25	3	ATS	738.68	739.53	739.7	739.27	741.05	741.23	738.97	738.44	739.7	738.92
		TOCT	734.50	734.49	735.33	738.43	738.83	736.70	735.57	738.30	732.16	733.39
		CE	734.89	-	-	-	-	-	-	-	-	-
5.25	4	ATS	492.12	489.09	490.88	487.48	490.22	487.12	488.08	489.99	487.86	487.26
		TOCT	485.01	485.13	486.71	489.09	487.02	483.73	487.55	488.87	486.45	484.07
		CE	485.56	-	-	-	-	-	-	-	-	-
5.25	5	ATS	302.69	302.17	302.20	302.17	302.33	302.26	302.48	302.15	302.08	302.31
		TOCT	289.96	300.34	301.19	300.86	299.24	300.11	300.01	300.12	300.19	299.85
		CE	301.67	-	-	-	-	-	-	-	-	-
5.25	6	ATS	255.87	256.11	256.10	256.28	255.855	257.78	256.12	256.07	256.39	256.56
		TOCT	253.58	251.74	251.51	251.02	205.09	252.43	251.34	253.74	250.03	253.6
		CE	254.73	-	-	-	-	-	-	-	-	-
5.25	7	ATS	303.86	303.37	303.29	303.74	303.04	303.01	303.72	303.24	303.34	303.91
		TOCT	301.45	300.78	300.01	302.19	298.48	300.76	301.52	301.1	301.38	298.15
		CE	301.38	-	-	-	-	-	-	-	-	-
5.25	8	ATS	221.32	222.43	223.20	222.61	221.72	222.65	222.04	222.07	222.39	221.83
		TOCT	215.88	215.84	218.26	217.66	219.78	220.21	215.46	217.43	217.36	222.90
		CE	219.14	-	-	-	-	-	-	-	-	-

Table A.2: Maximum throughput achieved per replication for each scenario and method.

Problem set	Scenario	Method	Replication Number									
			1	2	3	4	5	6	7	8	9	10
5.50	1	ATS	653.5	653.62	653.96	654.02	654.69	654.39	653.46	653.85	654.01	654.15
		TOCT	652.67	650.77	650.78	652.74	652.44	651.60	652.02	651.85	652.90	652.58
		CE	654.19	-	-	-	-	-	-	-	-	-
5.50	2	ATS	762.35	763.25	763.15	762.30	762.38	761.78	761.83	763.82	762.96	762.61
		TOCT	745.43	744.38	733.06	734.48	740.84	751.00	745.22	742.47	744.61	739.63
		CE	761.92	-	-	-	-	-	-	-	-	-
5.50	3	ATS	863.77	862.58	864.75	865.69	864.10	863.03	862.24	862.43	862.80	863.20
		TOCT	853.13	860.06	857.47	857.33	858.64	861.05	859.59	858.67	857.60	860.41
		CE	863.70	-	-	-	-	-	-	-	-	-
5.50	4	ATS	803.46	802.05	806.57	801.15	801.09	801.46	802.16	803.94	803.98	802.33
		TOCT	795.55	785.21	783.04	795.77	794.66	793.09	788.86	793.10	794.23	788.11
		CE	803.40	-	-	-	-	-	-	-	-	-
5.50	5	ATS	363.14	362.91	362.92	362.78	362.95	363.07	362.95	363.37	363	362.74
		TOCT	362.8	362.54	362.63	362.51	362.39	362.57	362.28	362.58	362.5	362.7
		CE	362.69	-	-	-	-	-	-	-	-	-
5.50	6	ATS	271.84	272.16	271.8	271.83	272.28	272.12	272.09	272.29	272.46	271.84
		TOCT	271.71	271.29	271.5	271.12	271.53	271.31	271.03	271.81	271.01	271.79
		CE	271.80	-	-	-	-	-	-	-	-	-
5.50	7	ATS	352.71	352.84	353.01	352.61	352.87	352.75	353.02	352.65	353.46	352.62
		TOCT	351.87	351.81	351.59	351.28	351.81	351.99	352.23	351.52	351.52	352.21
		CE	352.16	-	-	-	-	-	-	-	-	-
5.50	8	ATS	229.88	230.06	230.37	229.92	229.16	230.26	229.85	229.77	230.48	230.31
		TOCT	215.94	221.87	222.12	222.7	219.15	226.57	217.11	222.33	220.47	221.78
		CE	229.63	-	-	-	-	-	-	-	-	-

Table A.3: Maximum throughput achieved per replication for each scenario and method.

Problem set	Scenario	Method	Replication Number									
			1	2	3	4	5	6	7	8	9	10
5.100	1	ATS	941.45	941.4	941.47	941.79	941.25	941.16	941.63	941.38	941.07	941.52
		TOCT	941.26	940.94	940.76	940.91	940.87	940.9	941.13	940.76	941.05	940.87
		CE	941.76	-	-	-	-	-	-	-	-	-
5.100	2	ATS	438.73	439.42	438.94	439.83	438.97	440.37	439.59	438.8	439.76	439.95
		TOCT	435.63	436.49	435.37	437.87	435.92	436.8	437.08	436.57	438.4	436.92
		CE	440.17	-	-	-	-	-	-	-	-	-
5.100	3	ATS	806.11	806.6	804.91	805.25	806.29	806.77	807.55	807.68	806.83	804.61
		TOCT	805.38	804.53	805.09	804.94	805.03	805.26	804.67	804.25	804.28	806.51
		CE	806.70	-	-	-	-	-	-	-	-	-
5.100	4	ATS	438.01	438.67	438.84	440.4	436.8	436.11	436.14	437.94	436.46	439.7
		TOCT	426.45	436.25	427.03	427.54	429.91	430.01	424.34	421.85	423.1	416.4
		CE	441.10	-	-	-	-	-	-	-	-	-
5.100	5	ATS	319.03	318.94	318.78	318.82	318.55	318.75	318.81	318.6	318.69	319.09
		TOCT	317.29	317.76	317.52	317.67	318.23	317.69	317.98	317.98	317.18	317.18
		CE	319.30	-	-	-	-	-	-	-	-	-
5.100	6	ATS	291.85	291.68	291.66	291.59	292.17	291.44	291.9	291.58	291.26	291.43
		TOCT	290.6	290.83	290.46	290.44	290.07	290.64	290.71	290.47	290.63	290.79
		CE	291.81	-	-	-	-	-	-	-	-	-
5.100	7	ATS	328.13	328.06	328.44	328.14	327.85	328.2	327.88	327.87	328.23	328.12
		TOCT	327.65	327.14	326.78	327.37	326.78	327.26	326.81	326.9	327.14	326.88
		CE	328.29	-	-	-	-	-	-	-	-	-
5.100	8	ATS	308.6	307.96	308.82	308.29	308.1	307.68	307.06	308.28	307.22	308.17
		TOCT	305.46	304.81	304.18	304.58	305.54	303.92	304.45	304.47	305.05	305.33
		CE	308.20	-	-	-	-	-	-	-	-	-

Table A.4: Maximum throughput achieved per replication for each scenario and method.

Problem set	Scenario	Method	Replication Number									
			1	2	3	4	5	6	7	8	9	10
10.50	1	ATS	707.58	708.2	707.57	707.38	707.93	708.37	707.58	707.36	707.59	707.43
		TOCT	698.93	701.89	701.14	700.92	699.57	700.04	699.03	699.9	698.72	698.93
10.50	2	ATS	61.04	61.07	61.29	61.17	60.95	61.11	61.26	61.21	61.15	61.07
		TOCT	60.79	60.64	60.84	60.69	60.84	60.5	60.71	60.72	60.75	60.51
10.50	3	ATS	450.82	450.96	450.3	450.76	450.09	450.02	450.41	450.53	451.05	450.24
		TOCT	438.89	432.83	430.09	435.98	424.23	419.44	431.26	427.3	432.19	435.22
10.50	4	ATS	34	33.94	34.4	34.09	34.1	34.05	34.1	34.12	34.15	34.23
		TOCT	32.32	32.17	32.19	31.77	32.56	31.75	31.89	31.77	32.16	31.95
10.50	5	ATS	325.87	325.96	325.83	325.85	326.12	325.74	325.95	325.91	325.66	325.83
		TOCT	325.29	324.87	324.82	324.73	324.72	324.68	324.91	324.67	324.87	324.56
10.50	6	ATS	277.1	277.04	277.11	277.11	277.6	276.76	276.87	276.61	276.84	276.85
		TOCT	275.22	274.28	274.79	275.02	274.88	275.05	274.91	275.07	275.38	274.84
10.50	7	ATS	299.46	299.86	300.03	300.34	299.5	300.17	300.04	299.72	299.6	299.98
		TOCT	298.04	298.09	298.55	298.84	298.32	298.06	298.87	297.6	298.33	298.12
10.50	8	ATS	201.04	200.36	199.46	200.5	203.31	199.17	201.3	201.2	200.78	199.92
		TOCT	197.02	196.48	196.8	197.2	198.36	197.44	197.26	197.77	196.46	196.8

Table A.5: Maximum throughput achieved per replication for each scenario and method.

Problem set	Scenario	Method	Replication Number									
			1	2	3	4	5	6	7	8	9	10
10.100	1	ATS	840.95	840.9	841.02	840.97	840.9	840.89	840.99	840.98	840.89	841.05
		TOCT	839.23	839.3	839.31	839.39	839.22	839.36	839.23	839.37	839.42	839.16
10.100	2	ATS	709.98	710.24	710.12	710.17	710.45	709.82	710.49	710.15	709.73	709.82
		TOCT	698.88	700.96	700.53	703.95	700.2	699.68	699.8	700.12	701.06	702.41
10.100	3	ATS	770.75	772.04	770.03	770.95	771.26	771.31	770.56	770.21	770.28	769.94
		TOCT	762.1	764.14	763.8	759.69	761.35	761.09	762.44	760.78	761.07	759.15
10.100	4	ATS	490.24	490.3	490.39	491.66	490.83	491.97	490.25	492.31	490.14	489.57
		TOCT	456.83	474.11	460.77	438.61	449.47	464.19	450.35	468.16	463.42	476.82
10.100	5	ATS	278.69	278.56	278.77	278.59	278.69	278.59	278.88	278.88	278.71	278.64
		TOCT	277.71	277.19	277.24	277.63	277.3	277.23	277.6	277.46	277.2	277.59
10.100	6	ATS	260.73	261.08	260.73	261.26	260.82	261.04	260.9	260.69	261.01	261.02
		TOCT	259.46	259.21	259.27	259.76	259.14	259.22	259.26	258.94	259.01	259.88
10.100	7	ATS	293.44	293.8	293.51	294.25	294.38	293.89	293.51	293.65	293.55	293.54
		TOCT	291.04	291.09	291.33	291.01	291.03	291.01	292.05	290.25	290.56	291.7
10.100	8	ATS	128.76	127.2	127.51	126.93	128.58	127.63	127.8	127.98	127.68	127.77
		TOCT	123.6	124.84	125.26	124.64	125	125.81	124.8	123.95	124.86	126.67

Table A.6: Maximum throughput achieved per replication for each scenario and method.

Problem set	Scenario	Method	Replication Number									
			1	2	3	4	5	6	7	8	9	10
10.200	1	ATS	504.62	504.9	504.77	505.38	504.82	504.65	504.73	504.72	504.75	504.75
		TOCT	503.67	503.63	503.51	504.02	503.52	504.04	503.53	504.17	503.81	503.58
10.200	2	ATS	616.13	616.34	616.3	616.44						
		TOCT	615.94	616	615.89	615.79	616.25	615.92	615.79	615.63	615.53	616.46
10.200	3	TOCT	667.24	668.15	664.1	663.36	660.19	674.64	669.06	670.52	675.85	667.98
10.200	4	TOCT	406.46	412.6	420.99	409.06	406.29	419.16	430.83	405.74	419.01	409.57
10.200	5	TOCT	284.32	284.94	284.28	284.37	284.74	284.37	284.55	284.13	284.57	284.4
10.200	6	TOCT	218.51	218.45	219.17	219.82	218.57	219.15	218.71	218.8	218.86	219.25
10.200	7	TOCT	218.93	217.56	218.97	218.09	218.18	218.58	217.81	218.52	219.11	218.46
10.200	8	TOCT	142.76	142.07	142.78	142.94	142.52	143.58	142.23	141.86	141.97	141.17

Bibliography

[1] Altiparmak, F., Dengiz, B., and Bulgak, A. A. (2007). Buffer allocation and performance modeling in asynchronous assembly system operations: An artificial neural network metamodeling approach. *Applied Soft Computing Journal*, 7(3):946–956.

[2] Amiri, M. and Mahtashami, A. (2012). Buffer allocation in unreliable production lines based on design of experiments, simulation, and genetic algorithm. *International Journal of Advanced Manufacturing Technology*, pages 371–383.

[3] Battini, D., Persona, A., and Regattieri, A. (2009). Buffer size design linked to reliability performance: A simulative study. *Computers and Industrial Engineering*, 56(4):1633–1641.

[4] Bulgak, A. A. (2006). Analysis and design of split and merge unpaced assembly systems by metamodelling and stochastic search. *International Journal of Production Research*, 44(18/19):4067–4080.

[5] Burman, M. H. (1995). *New results in flow line analysis.* PhD thesis, Massachusetts Institute of Technology.

[6] Can, B. and Heavey, C. (2011). Comparison of experimental designs for simulation-based symbolic regression of manufacturing systems. *Computers \& Industrial Engineering*, 61(3):447–462.

[7] Can, B. and Heavey, C. (2012). A comparison of genetic programming and artificial neural networks in metamodeling of discrete-event simulation models. *Computers \& Operations Research*, 39(2):424–436.

[8] Chan, F. T. S. and Ng, E. Y. H. (2002). Comparative evaluation of buffer allocation strategies in a serial production line. *The International Journal of Advanced Manufacturing Technology*, 19:789–800.

[9] Demir, L., Tunal, S., and Eliiyi, D. T. (2012). An adaptive tabu search approach for buffer allocation problem in unreliable non-homogenous production lines. *Computers and Operations Research*, 39(7):1477–1486.

[10] Demir, L., Tunali, S., and Eliiyi, D. T. (2014). The state of the art on buffer allocation problem: A comprehensive survey. *Journal of Intelligent Manufacturing*, 25(3):371–392.

[11] Demir, L., Tunali, S., Eliiyi, D. T., and Løkketangen, A. (2013). Two approaches for solving the buffer allocation problem in unreliable production lines. *Computers and Operations Research*, 40(10):2556–2563.

[12] Gershwin, S. B. (1987). An efficient decomposition method with finite storage space and blocking. *Operations Research*, 35(2).

[13] Gosavi, A. (2015). *Simulation-Based Optimization*. Springer, ISSN 1387-666X, second edition.

[14] Grobler, W. (2017). *Investigating net ideal cycle time estimation and efficient buffer allocation for Body-in-White production lines*. PhD thesis, University of Pretoria.

[15] Jeong, K. C. and Kim, Y. D. (2000). Heuristics for selecting machines and determining buffer capacities in assembly systems. *Computers and Industrial Engineering*, 38(3):341–360.

[16] Kolb, O. and Göttlich, S. (2015). A continuous buffer allocation model using stochastic processes. *European Journal of Operational Research*, 242(3):865–874.

[17] Kose Simge, Demir Leyla, T. S. and Deniz, E. T. (2014). Capacity improvement using simulation optimization approaches: A case study in the thermotechbology industry. *Engineering Optimization*, 47(2):149–164.

[18] Law, A. M. (2015). *Simulation Modeling and Analysis*. McGraw-Hill Education, fifth edition.

[19] Lee, S. (2000). Buffer sizing in complex cellular manufacturing systems. *International Journal of Systems Science*.

[20] Manson, N. (2006). Is operations research really research? *ORiON*, 22(2):155–180.

[21] Nahas, N., Ait-Kadi, D., and Nourelfath, M. (2006). A new approach for buffer allocation in unreliable production lines. *International Journal of Production Economics*, 103(2):873–881.

[22] OICA (2018). International organization of motor vehicle manufacturers. Available at http://www.oica.net/category/economic-contributions/.

[23] Sabuncuoglu, I., Erel, E., and Gocgun, Y. (2006). Analysis of serial production lines: characterisation study and a new heuristic procedure for optimal buffer allocation. *International Journal of Production Research*, 44(13):2499–2523.

[24] Spieckermann, S., Gutenschwager, K., Heinzel, H., and Voß, S. (2000). Simulation-based optimization in the automotive industry – A case study on body shop design. *Simulation*, 75(5):276–286.

[25] Talbi, E. (2009). *Metaheuristics: from design to implementation*. Johan Wiley & Sons, Hoboken, N.J.

[26] Tiacci, L. (2012). Event and object oriented simulation to fast evaluate operational objectives of mixed model assembly lines problems. *Simulation Modelling Practice and Theory*, 24:35–48.

[27] Vitanov, I. V., Vitanov, V. I., and Harrison, D. K. (2009). Buffer capacity allocation using ant colony optimisation algorithm. *Proceedings of the 2009 Winter Simulation Conference*, pages 3158–3168.

[28] Yücesan, E., Chen, C.-H., Snowdon, J. L., and Charnes, J. M., editors (2002). *SSJ: A Framework for Stochastic Simulation in Java*. IEEE Press. Available at http://simul.iro.umontreal.ca/ssj/indexe.html.

[29] Zandieh, M., Joreir-Ahmadi, M. N., and Fadaei-Rafsanjani, A. (2017). Buffer allocation problem and preventive maintenance planning in non-homogenous unreliable production lines. *International Journal of Advanced Manufacturing Technology*, pages 1–13.