Volume 36 (2), pp. 111–139
http://orion.journals.ac.za

ORiON
ISSN 0259–191X (print)
ISSN 2224–0004 (online)
©2020

Solving the buffer allocation problem using simulation-based optimisation

JW Joubert[*] DJ Kotze[†]

Received: 3 August 2020; Revised: 26 October 2020; Accepted: 28 October 2020

Abstract

In production lines, buffers function as a means to decouple stations, which reduce the effect that station failures and varying process times have on the complete line's throughput. However, adding larger buffers can be costly, for example, in the automotive industry where it results in increased working capital. This manuscript addresses the buffer allocation problem (BAP), seeking the smallest total buffer size while meeting a prescribed throughput by employing a simulation-based optimisation approach. A Tabu Search algorithm searches the solution space for the optimal buffer configuration while a discrete event simulation model evaluates each configuration, accounting for the machine (un)reliability. Since the multiple simulations add a sizeable computational burden, our approach introduces a novel neighbourhood search mechanism, which borrows from the Theory of Constrains. Solving test sets available in the literature suggest that this approach is 18 times faster than prior Adaptive Tabu Search approaches for small problems, and more than five times faster for medium-sized problems.

Key words: Buffer allocation problem, discrete event simulation, simulation-based optimisation, tabu search, theory of constraints.

1 Introduction

Factors such as uneven processing time and station failure create instability in the produc- tion line, which has a negative effect on throughput. These factors can cause other stations to become either blocked or starved. Buffers are locations in which semi-completed parts, also called work-in-progress (WIP), are stored within a 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. The bigger the buffer, the longer the production line can run independently.

[*]Corresponding author: Centre for Transport Development, Industrial & Systems Engineering, University of Pretoria, South Africa, email: johan.joubert@up.ac.za

[†]Department of Industrial and Systems Engineering, BMW South Africa, Rosslyn, Pretoria, South Africa, email: dirk.j.kotze12@gmail.com

http://dx.doi.org/10.5784/36-2-684

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 problem of allocating buffers (location and size) optimally is known as the BAP. In this study the BAP is solved for a serial and non-serial heterogeneous process time, unreliable production line.

The BAP is frequently solved using a two-step approach [11]. The *evaluation* step calculates the throughput of a production line for a given buffer configuration. Various versions of evaluation have been employed in the literature. Analytical methods have been used for the BAP. Exact analytical methods are only applicable to small-sized problems. 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. The literature found on the BAP using the decomposition method is limited to serial production lines. 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, or digital twin, 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. But the disadvantage of simulation is the time it takes to evaluate a scenario. This can be reduced by creating a discrete event simulation (DES) model specifically for the application that does not have the additional general tools available in commercial simulation software.

Simulation on its own, as well as analytical methods, are not optimisation techniques per se. Hence, the second *generative* step moves through the solution space and considers the various buffer configurations to be evaluated [11]. The generative step 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 mechanisms to the way it moves through the solution space and have the main advantage that they can escape local optimum.

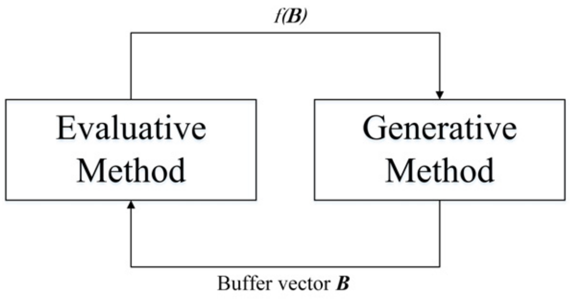

Figure 1: *Buffer allocation problem solution approach.*

The methods start with an initial buffer configuration $\boldsymbol{B}$. With the given buffer configura- tion, the evaluative method can determine the throughput of the line, $f(\boldsymbol{B})$. 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.

In this study, simulation-based optimisation (SBO) is used to solve the BAP. A DES model created in Java is used as evaluative method. An adaptive tabu search with theory of constraints neighbour generation is proposed to solve the BAP for both a serial and non-serial heterogeneous unreliable production line. The performance of the algorithm is investigated for small-size as well as medium and large-sized problems. Finally the scalability of the proposed solution is tested on a non-serial large sized problem from industry.

The rest of the article is organised as follows. The assumptions of the model and the BAP is described in the next section based on a comprehensive literature review on the BAP. The method of using an evaluative and generative element iteratively to solve the BAP is studied. In §3 the SBO approach is developed. A simulation program is created in a general programming language, Java. This program is a newly designed blueprint using the library of the stochastic simulation in Java (SSJ). The program simulates 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. In §3.2 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.* [10]. The second method is a proposed improvement on the first to improve the evaluation time. In §4 this artefact is 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.

2 Literature review

The BAP deals with finding the optimal buffer configuration to incorporate in the produc- tion line to achieve a specific objective [11]. Total buffer size N is allocated among the $K-1$ intermediate buffer locations in the production line with K machines (Figure 2).

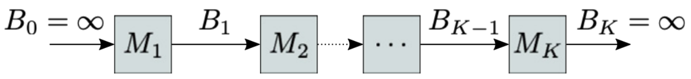

Figure 2: *A typical series of machines $M_1, \ldots, M_K$ with finite buffers $B_1, \ldots, B_{K-1}$ in a queuing network.*

Associated with each machine $M_k, k \in \boldsymbol{K}$ 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 $N = \sum_{k=1}^{K-1} B_k$ where B_k denotes the number of buffer units at location k.

Due to the complexity of the BAP, numerous publications are available in the literature. For a comprehensive study of the available literature on the BAP see the works of Demir *et al.* [11]. The BAP can be expressed in three main forms depending on the objective function.

Objective 1 Maximise the throughput of the line for a given fixed number of buffer N [10, 19, 20]. 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 [12, 15, 21]. 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 article. Production plant designers are faced with the problem of achieving a balance between throughput and WIP in buffers.

The BAP solution approach 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, if the process times are different, it is known as a *heterogeneous* line [25]. 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* [11]. BAP has been proven to be an NP-Hard combinatorial optimisation problem [11]. 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. 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. A generative and evaluative method is executed in an iterative manner. The generative method generates various permutation of buffer allocation from the solution space for the evaluative method to evaluate and determine the throughput of the line.

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. Thus, 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. 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, knowledge-based methods as well as degrading ceiling local search heuristics have been used [14, 15, 20]. 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. Metaheuristics have been widely applied in the buffer allocation problem. Typical solution methods in this area are genetic algorithm, tabu search, simulated annealing and ant-colony.

Genetic algorithm is based on the theory of evolution that favour reproduction of individuals with specific traits. In Dolgui [13] two different genotype generation mechanisms are compared for the genetic algorithm. The first increases or decreases an individual buffer by one, the second approach continues increasing or decreasing the gene as long as an improvement in the goal is achieved. Cruz [9] employed a special version multi-objective genetic algorithm to solve the BAP for objective two.

Demir *et al.* [10] 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.* [10] proposed a new ATS and compared it to the simple Tabu Search (TS) 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 The ATS also employs an intensification strategy. If a solution found to be the best does not change for a certain number of iterations, $(0.25 \times N)$, the increment (decrement) size is reduced to 1 for large-sized problems.

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.

Demir *et al.* [10] 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. 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.

Demir *et al.* [12] expanded on the work of Demir *et al.* [10] to solve the BAP for objective two. The inner loop includes an 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 cannot 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.

The main advantage of metaheuristics are that they can escape local optimum. Their main disadvantage is that they are not problem specific and thus, they have to be adjusted to produce solutions for a specific problem.

Artificial neural networks [1, 4, 6, 7], as well as genetic programming [6, 7] are successfully employed to solve the BAP.

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 [11]. For larger production systems, approximation methods are used. Two approximation methods studied in the BAP literature are the decomposition method [9, 10, 12, 19] and aggregation method.

The general idea of the decomposition method it to break the line up into L smaller lines, while the aggregation method combines the stations. These methods are very similar in implementation. The decomposition method is time-efficient in the calculation of the line's throughput and is based on the theory of queueing networks. 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. Burman [5] improved the method to be able to model heterogeneous lines. The main advantage of the decomposition method is its computational efficiency. However, the disadvantage is that it can be applicable only under the assumptions that processing rates are either deterministic or exponentially distributed and failure and repair rates are either geometric or exponentially distributed random variables.

An alternative method used as an evaluative method for the BAP is a simulation model. 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 adequate 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 [17].

Simulation is applied in various research contributions [2, 3, 4, 7, 8, 16, 18, 21, 22, 23]. 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. 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 [18] developed an approach to find the buffer configuration that leads to the lowest cost in terms of investment and running cost. Amiri & Mahtashami [2] proposed a multiobjective formulation to solve the BAP for an unreliable line. A detailed DES 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 & Mahtashami [2] was a polynomial regression model. Can & Heavey [6] investigated the robustness and accuracy of genetic programming to create meta-models. In a subsequent paper, Can & 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 & 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. 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. 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. 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* [17]. 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 various components of a DES is presented by Law [17]. 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. When a simulation model is created the 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. Upon initialisation, the simulation program schedules the events. The time of each event occurrence, t_k, is randomly generated from the distributions. This is added to the current time to determine the time of occurrence. 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. 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.

Assembly line simulator (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 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. Kose *et al.* [16] 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*. Spieckermann *et al.* [21] conducted a comprehensive case study at BMW AG. 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.

3 Material and methods

Recall that solving the 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 article, the *best* buffer configuration will be the one that achieves the desired throughput with the minimum total number of buffers, objective two. The lines under investigation are subject to machine failure and different processing times, and these are stochastic in nature. It cannot be assumed that the distributions for these process rates are exponentially distributed or that the failure and repair rates are either geometric or exponentially distributed. These assumptions are required for approximation methods such as the decomposition method. The throughput of these lines can be evaluated using simulation as the *evaluative method*, with any distributions. 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. Demir *et al.* [12] solved the BAP for objective two using tabu search. The method was extensively tested on various line sizes and machine parameters, forming a solid base for generative method. The combined use of simulation and metaheuristics is known as SBO.

3.1 Discrete event simulation

DES require various components to function. The SSJ package, [24] 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 article, a unique *main program* is created to combine each of these routines. Two specific event routines are created to meet the requirements of BAP.

The various components of DES and the interaction between them are shown in Figure 3. A *model creation* program is used to create the model of the line itself. The number of machines, total buffer limit and machine parameters are defined. 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. 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.

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 *failure event*.

Departure event is triggered when a part is completed and leaves the station. Law [17] 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 flow chart Figure 4, is executed. 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. It is assumed that 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.

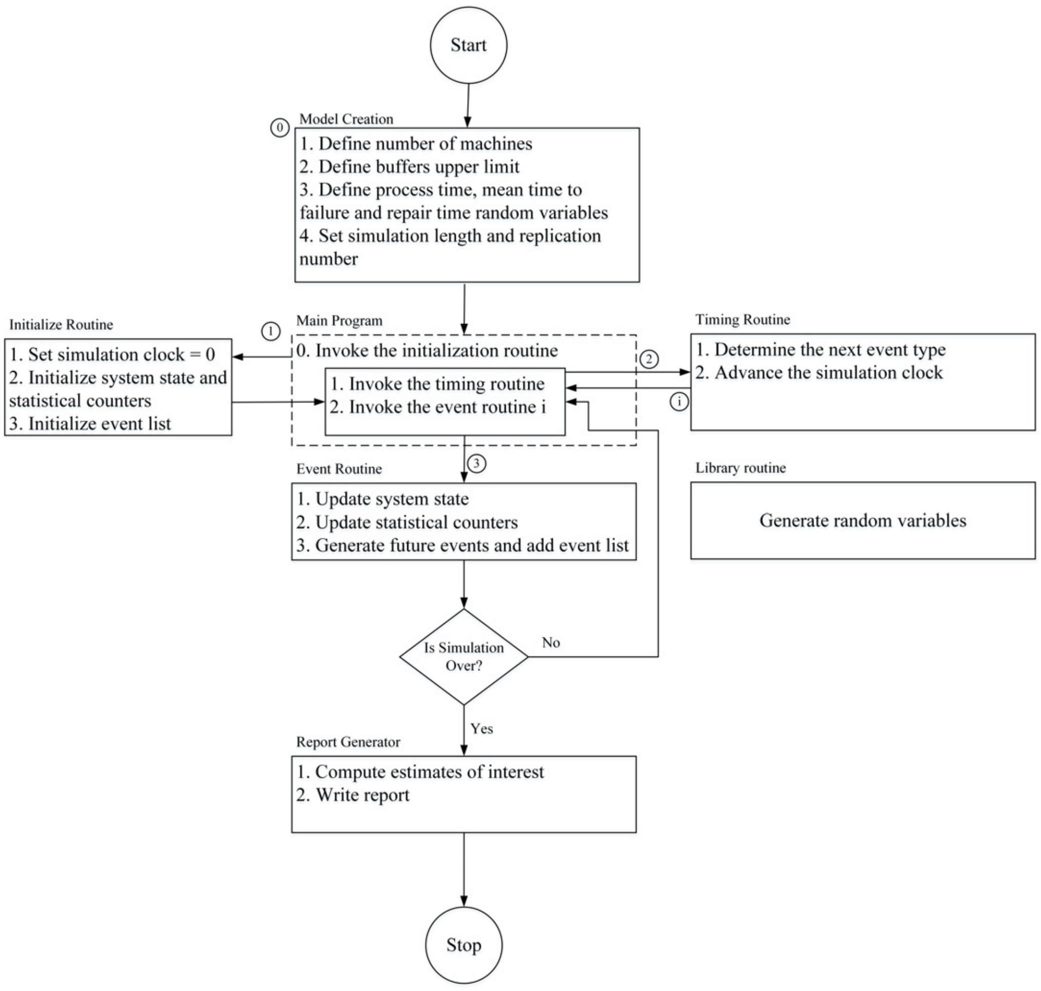
Start
0
Model Creation
1. Define number of machines
2. Define buffers upper limit
3. Define process time, mean time to failure and repair time random variables
4. Set simulation length and replication number
Initialize Routine
1. Set simulation clock = 0
2. Initialize system state and statistical counters
3. Initialize event list
1
Main Program
0. Invoke the initialization routine
1. Invoke the timing routine
2. Invoke the event routine i
2
Timing Routine
1. Determine the next event type
2. Advance the simulation clock
i
3
Event Routine
1. Update system state
2. Update statistical counters
3. Generate future events and add event list
Library routine
Generate random variables
Is Simulation Over?
No
Yes
Report Generator
1. Compute estimates of interest
2. Write report
Stop

Figure 3: *Flow control for DES with the next-event time-advance approach.*

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.

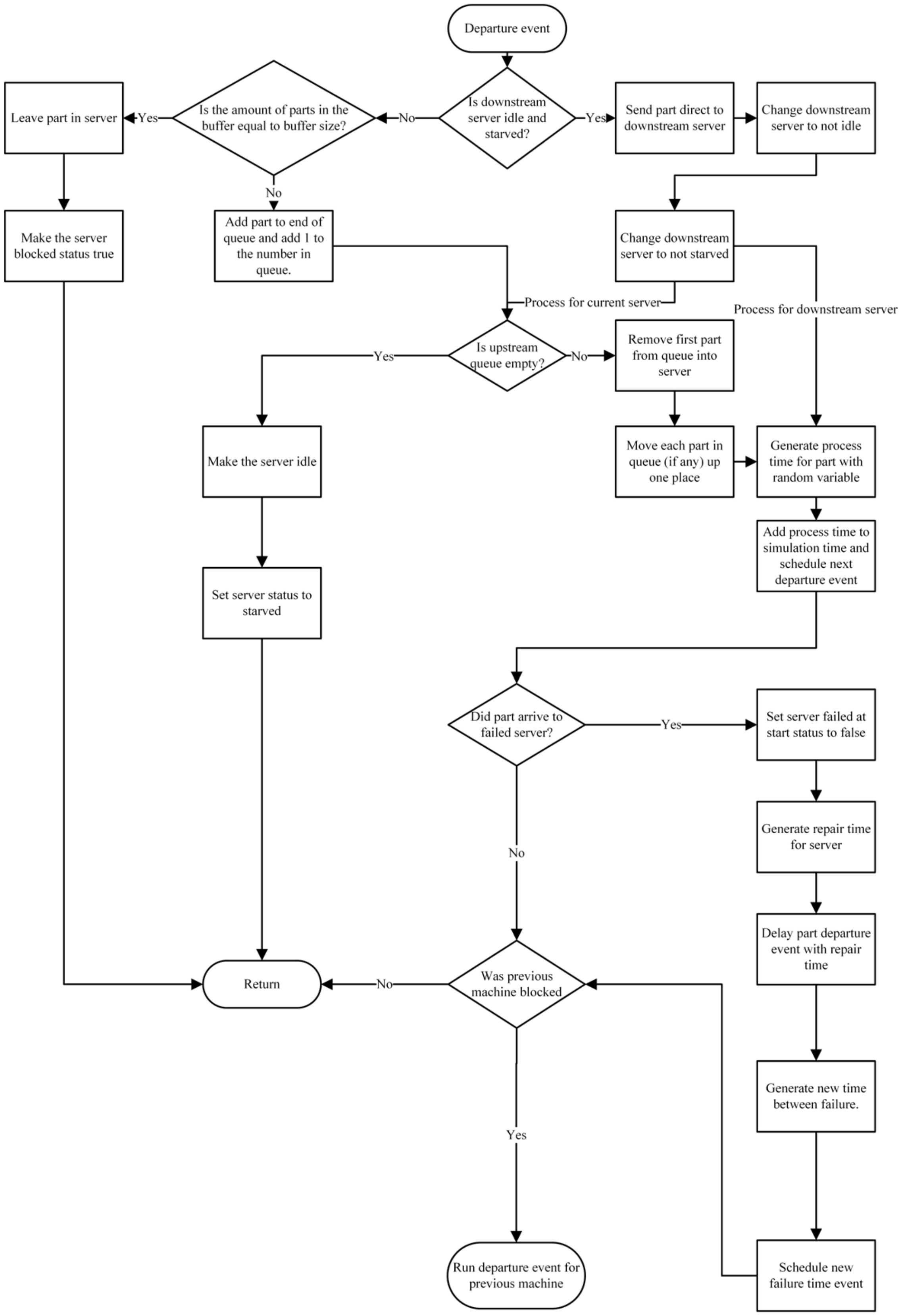
Departure event
Is downstream server idle and starved?
Yes
Send part direct to downstream server
Change downstream server to not idle
No
Is the amount of parts in the buffer equal to buffer size?
Yes
Leave part in server
Make the server blocked status true
No
Add part to end of queue and add 1 to the number in queue.
Change downstream server to not starved
Process for current server
Process for downstream server
Is upstream queue empty?
Yes
No
Remove first part from queue into server
Move each part in queue (if any) up one place
Generate process time for part with random variable
Add process time to simulation time and schedule next departure event
Make the server idle
Set server status to starved
Did part arrive to failed server?
Yes
Set server failed at start status to false
Generate repair time for server
Delay part departure event with repair time
Generate new time between failure.
Schedule new failure time event
No
Was previous machine blocked
No
Return
Yes
Run departure event for previous machine

Figure 4: *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 5.

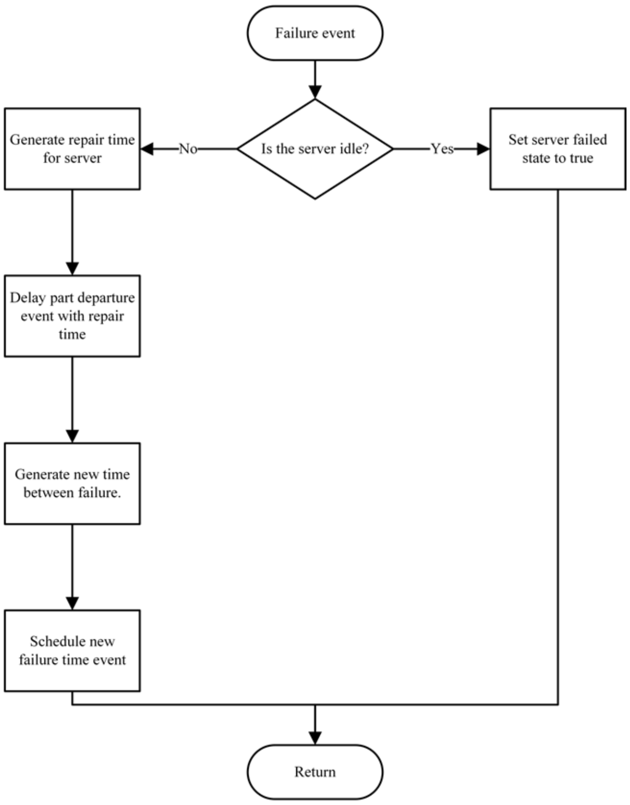
Failure event
Is the server idle?
No
Yes
Generate repair time for server
Set server failed state to true
Delay part departure event with repair time
Generate new time between failure.
Schedule new failure time event
Return

Figure 5: *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 DES is implemented in Java and used to evaluate the throughput of the production line scenario generated by the generative method. The simulation returns the throughput as well as statistics on the amount of time a station was either blocked or starved.

3.2 Tabu Search

Demir *et al.* [11] proposed two TS metaheuristics working in an iterative manor to solve the BAP for objective two. The inner loop using the ATS determines the maximum throughput a line can achieve for a given buffer size N. The ATS neighbour generation evaluates all the scenarios that can be reached by either increasing or decreasing each individual buffer size. Some of these neighbours actually decreases the buffer after stations that are highly blocked and increase for stations that are rarely blocked, generating *'bad'* neighbours. To overcome this an alteration to the work of Demir *et al.* [10] is proposed for neighbour generation, this method is called the Theory of Constraints Tabu Search (TOCT).

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. It is crucial that this station is never blocked nor starved. This bottleneck can be identified with reports from the DES.

The DES 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 number of neighbours that need to be evaluated per iteration of the inner tabu. The factor 0.5 is quite an arbitrary choice in this paper but can be subjected to sensitivity analysis in future work.

Another alteration to the ATS 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 ATS in the work of Demir *et al.* [10].

The outer loop is based on the work of Demir *et al.* [12]. It reduces the total size of N to obtain the smallest N possible while still meeting the required throughput. The algorithm starts with an initial buffer size of N_0. Upon initialisation N should be a sufficiently large number. 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 outer tabu loop is then used to generate a new total buffer size. Again the inner loop evaluates this and returns the best buffer configuration and throughput. 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.

4 Computational study

Experiments on small-sized, medium and large-sized problems with varying machine para- meters are done to evaluate the efficiency of the proposed solution method. At first serial production lines are considered.

The DES execution time is compared to commercial simulation software. Experiments are done to determine the simulation length and replication that achieves a balance between execution time and solution quality. The DES is then used to test the proposed TOCT against the ATS inner loop.

The BAP is solved for objective two by using the DES with the TOCT inner loop and outer loop from Demir *et al.* [12]. Finally the proposed solution approach's scalability is tested on a real world non-serial production line.

4.1 Data sets

Data sets used in Demir *et al.* [10] will also be used in this article. Four line sizes with the number of machines denoted by the set $K = \{5, 10, 20, 40\}$ are evaluated. 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 1.

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

Table 1: *Serial line scenarios for SBO experiments [10].*

Lines with 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). As per Demir *et al.* [10], the range of line scenarios allows the effectiveness of the proposed method to be tested across various line sizes. For each line scenario the various machine characteristics are tested. Again the work of Demir *et al.* [10] is referred to. The following section explain how the random process time, time between failure and repair time for each machine are defined. Table 2 shows the eight different line parameters.

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 other. 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 $b = 400$.

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])

Table 2: *Serial line machine parameters for SBO experiments [10].*

4.2 DES experiment and results

The DES is tested in this 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.

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. The DES is compared to a commercially available program to test similarity of results.

Only 10 of the 12 line scenarios in Table 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 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*$^{\text{TM}}$ were 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 (1)

$$\frac{\textit{Simio}^{\text{TM}} \text{ throughput} - \text{DES throughput}}{\textit{Simio}^{\text{TM}} \text{ throughput}} \times 100, \tag{1}$$

while the time factor is calculated using equation (2)

$$\frac{\textit{Simio}^{\text{TM}} \text{ computational time}}{\text{DES computational time}}. \tag{2}$$

Table 3 shows the results of the test. The first column shows the problem scenario, $(K.N)$.

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

Table 3: *Simulation results from Java tool compared to* Simio^{TM} *program.*

The second column shows the average throughput difference between the DES tool and the *Simio*$^{\text{TM}}$ program calculated with equation (1). The third column shows the time difference between the two programs calculated with equation (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 6 supports this. The figure shows the density plot of the throughput obtained from the 1 000 replications for line scenario 10.200. The two programs result in a density plot with similar shapes.

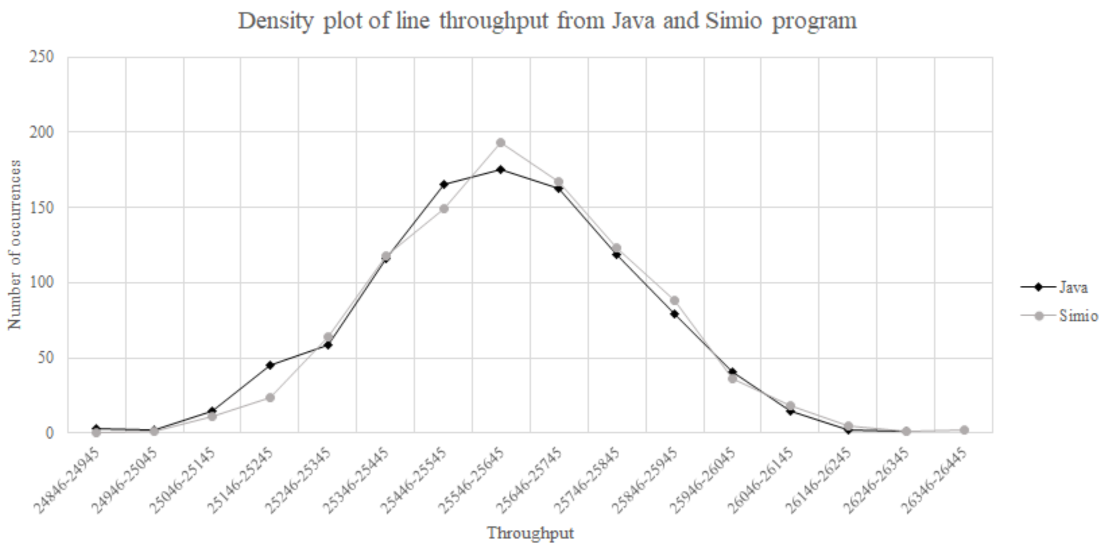

Figure 6: *Simulation throughput comparison between Java and* Simio^{TM} *program.*

DES had a significant speed advantage compared to *Simio*$^{\text{TM}}$. On average, *Simio*$^{\text{TM}}$ takes 71 times longer than the proposed program based on DES. For the small-sized problems, *Simio*$^{\text{TM}}$ 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 manuscript 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.

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 2 are tested across all the line sizes in Table 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.

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. Experiments of the simulation show, the throughput has a significant variation from the 16.89 time units per unit at the start of the simulation. 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.

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.

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.

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.

To find a *good* number of replications, simulations of the problem scenarios were done, each replicated 1000 times. 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. Experiments showed 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 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.

4.3 TOCT experiments and results

Once again the eight line scenarios in Table 1 and the 12 machine parameters in Table 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 ATS and TOCT algorithm are compared against complete enumeration (CE). However, for medium and large scale problems CE is not computationally feasible and will not be tested using CE but will be compared against the ATS.

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

Table 4 shows the results from the experiments on small problems using ATS. First the ATS is compared with the CE.

The first column shows the problem set, shown as machine and buffer size, $(K.N)$. The second column shows the experimental scenario from Table 2. The third column shows the % error of the average optimal (maximum) throughput achieved, compared to the solution found using CE. To calculate the % error we use

$$\text{Deviation for ATS} = \left(\frac{f(CE) - f(ATS)}{f(CE)}\right) \times 100. \tag{3}$$

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 comparison to complete enumeration was 0.30%. This small fluctuation can be attributed to the inherent stochasticity of the simulation, and in some instances, the method resulted in a better throughput than the 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
5.10	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

Table 4: *Inner tabu computational results for small sized problems ATS vs CE.*

The experiments on small-sized problems are repeated, using the new proposed TOCT method. To calculate the % error we use

$$\text{Deviation for TOCT} = \left(\frac{f(CE) - f(TOCT)}{f(CE)} \right) \times 100. \tag{4}$$

Table 5 shows the results from the experiments on small problems using TOCT.

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 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 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. Column five in Table 5 shows the number of iterations at which point the best throughput found does not change for the TOCT method.

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

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

Table 5: *Inner tabu computational results for small sized problems TOCT vs CE.*

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.

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 (5), which shows the calculation used to determine the throughput deviation between the two methods.

$$\begin{array}{c}\text{\% throughput deviation}\\ \text{between ATS and TOCT}\end{array} = \left(\frac{|f(TOCT) - f(ATS)|}{f(TOCT)}\right) \times 100 \tag{5}$$

Table 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 (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

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

Table 6: *Inner tabu termination criteria for small-sized problems.*

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

Table 7: *Inner tabu computational results for medium sized problems ATS vs TOCT.*

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. Once more the number of iterations the program took to reach the solution it returned is considered. The same termination criteria apply. 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.

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. Note due to long computational time, experiments on lines with 40 machines and 400 N as well as 800 N could not be performed using TOCT.

Table 8 shows the results for the large sized problem with 20 machines as well as 40 machines.

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

Table 8: *Inner tabu computational results for large sized problems, 20 & 40 machines ATS vs TOCT.*

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 **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.

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.

4.4 Solving the BAP using SBO, experiments and results

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 1 and Table 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 for small-sized problems are shown in Table 9.

The first column is the problem set from Table 1. The second column is the problem scenario from Table 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 10 show the results for medium and large-sized problems. With the large sized problems, 3 scenarios 20.100.8, 20.200.2 and 20.200.8 was not successfully solved, for medium-sized problems and remaining large-sized problems the model was able to achieve a required throughput close to the target throughput.

45 scenarios was less than or equal to $\lceil \frac{N}{2} \rceil$, of the starting N. Only three 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.

Experimentation showed that DES can effectively be used to solve BAPs. In conjunction with the proposed inner loop, using TOCT and the outer loop based on the works of [10] the BAP is solved for various line sizes and line parameters.

4.5 Solving BAP a case study

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.

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 9: *Outer tabu experimental results small sized problems.*

Such a line can consists of hundreds of welding robots, the production line this case study is based on has 292 robots. 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 topology is a tree structure, various machines have more than one station feeding parts into them. The three main sub-assemblies are produced in separate areas, the *Front End*, *Centre Floor* and *Rear End*. The *Front End* has seventeen machines. 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. These stations are followed by the main line, which produces the *underbody* and $sideframes$. 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.

Together the total facility has 63 machines, 59 buffer locations and a total buffer size N of 318. The BIW problem is considered a large-sized problem. The SBO method developed in this manuscript 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

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
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

Table 10: *Outer tabu experimental results medium sized problems.*

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 10 replications is returned as the performance result for the line.

The local automotive 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 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.

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 11: *Buffer allocation results production phase 1.*

The objective function of the BAP for this article is achieving a specific throughput with the smallest total buffer size. The proposed SBO method is used on a production line of a local automotive manufacturer of luxury vehicles. 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 11 and Table 12 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. The case study shows that the proposed SBO method is able to solve the BAP for complex, realistic lines.

5 Conclusion

In this study, we propose a SBO approach to solve the BAP for unreliable, heterogeneous serial and non serial production lines. The objective is to achieve a desired throughput with the smallest buffer size. The proposed approach uses DES designed specifically for the BAP as evaluative

Underbody			Framing		
Buffer index	**Initial *Bi* size**	**Optimal *Bi* size**	**Buffer index**	**Initial *Bi* size**	**Optimal *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

Table 12: *Buffer allocation results production phase 2.*

method and integrated TOCT as inner loop and TS outer loop as generative method. The TOCT uses an alternative neighbourhood generation mechanism based on theory of constraints. The proposed methods elements are tested on serial lines and finally applied to a case study.

The results on the experiments on the DES model show it can achieve similar throughput to that of commercial software, with 0.84% difference in total throughput compared to *Simio*$^{\text{TM}}$. DES has a significant time advantage compared to *Simio*$^{\text{TM}}$, for small-sized problems, *Simio*$^{\text{TM}}$ took 23 minutes compared to 19 seconds for DES, and 75 minutes for large problems compared to 1,5 minutes for DES. Initial experiments on simulation length and replication show that a balance between solution quality and execution time for this problem can be achieved. Simulation length of 10 000 and 200 replications are used. It must be noted that these values can be very problem specific.

Experiments on the TOCT show that for small-sized problems the absolute error for TOCT in comparison to CE was 0.88% versus the 0.30% for ATS. The significant difference between TOCT and ATS is computation time. The ATS method took, on average, 18 times longer to execute. Experiments also show that the metaheuristics solution stops improving long before the termination criteria are met. For medium-sized problems 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. Large sized problems the ATS method was computationally impossible to run, as well as experiments on line with 40 machines and 400, 800 N could not be done with TOCT.

The proposed SBO method is capable of solving the BAP for objective two. 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. With the large sized problems, 3 scenarios 20.100.8, 20.200.2 and 20.200.8 was not successfully solved, for medium-sized problems and remaining large-sized problems the model was able to achieve a required throughput close to the target throughput. 45 scenarios was less than or equal to $\lceil \frac{N}{2} \rceil$, of the starting N. Only three 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.

The model can be adapted to non-serial lines. The automotive manufacturing plant is measured on daily output performance. Solving the case study for objective one show that the throughput of the line can be increased from 239.7 unites per day to 282.7 units with the same total buffer size. Solving it for objective two with a throughput target of 230 units per day, the throughout was achieved with 242.9 units per day with total buffer size of 126, a reduction of the total buffer size with 126. The algorithm required 3 hours and 55 minutes to finish.

The use of parallel implementation can have a significant improvement on the computational time of the model and is another direction for future research.

References

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

[2] Amiri M & Mahtashami A, 2012, *Buffer allocation in unreliable production lines based on design of experiments, simulation, and genetic algorithm*, International Journal of Advanced Manufacturing Technology, **62(1–4)**, pp. 371–383.

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

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

[5] Burman MH, 1995, *New results in flow line analysis,* PhD Thesis, Massachusetts Institute of Technology.

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

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

[8] Chan FTS & Ng EYH, 2002, *Comparative evaluation of buffer allocation strategies in a serial production line*, The International Journal of Advanced Manufacturing Technology, **19(11)**, pp. 789–800.

[9] Cruz FR, Van Woensel T & Smith JM, 2010, *Buffer and throughput trade-offs in m/g/1/k queueing networks: A bi-criteria approach,* International Journal of Production Economics, **25(1)**.

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

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

[12] Demir L, Tunali S, Eliiyi DT & Løkketangen A, 2013, *Two approaches for solving the buffer allocation problem in unreliable production lines*, Computers & Operations Research, **40(10)**, pp. 2556–2563.

[13] Dolgui A, Eremeev A, Kolokolov A & Sigaev A, 2002, *A genetic algorithm for the allocation of buffer storage capacities in a production line with unreliable machines,* Journal of Mathematical Modelling and Algorithms, **1(2)**, pp. 89–104.

[14] Jeong KC & Kim YD, 2000, *Heuristics for selecting machines and determining buffer capacities in assembly systems*, Computers & Industrial Engineering, **38(3)**, pp. 341–360.

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

[16] Köse SY, Demir L, Tunal S & Eliiyi DT, 2015, *Capacity improvement using simulation optimization approaches: A case study in the thermotechnology industry*, Engineering Optimization, **47(2)**, pp. 149–164.

[17] Law AM, 2015, *Simulation Modeling and Analysis*, 5th Edition, McGraw-Hill Education, New York (NY).

[18] Lee S, 2000, *Buffer sizing in complex cellular manufacturing systems,* International Journal of Systems Science, **31(8)**, pp. 937–948.

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

[20] Sabuncuoglu I, Erel E & 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)**, pp. 2499–2523.

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

[22] 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**, pp. 35–48.

[23] Vitanov IV, Vitanov VI & Harrison DK, 2009, *Buffer capacity allocation using ant colony optimisation algorithm*, Proceedings of the 2009 Winter Simulation Conference, pp. 3158–3168.

[24] Yücesan E, Chen CH, Snowdon JL & Charnes JM (Eds), 2002, *SSJ: A Framework for Stochastic Simulation in Java*, IEEE Press, Available from http://simul.iro.umontreal.ca/ssj/indexe.html.

[25] Zandieh M, Joreir-Ahmadi MN & Fadaei-Rafsanjani A, 2017, *Buffer allocation problem and preventive maintenance planning in non-homogenous unreliable production lines*, International Journal of Advanced Manufacturing Technology, **91(5–8)**, pp. 2581–2593.