International Journal of Production Research,
Vol. 44, No. 13, 1 July 2006, 2499–2523

Taylor & Francis
Taylor & Francis Group

Analysis of serial production lines: characterisation study and a new heuristic procedure for optimal buffer allocation

I. SABUNCUOGLU*†, E. EREL‡ and Y. GOCGUN†

†Department of Industrial Engineering, Bilkent University, Ankara 06800, Turkey
‡Department of Management, Bilkent University, Ankara 06800, Turkey

(*Revision received September 2005*)

Buffer allocation in serial production lines is one of the important design issues, and hence it has been studied extensively in the literature. In this paper, we analyse the problem to characterise the optimal buffer allocation; specifically, we study the cases with single and multiple bottleneck stations under various experimental conditions. In addition, we develop an efficient heuristic procedure to allocate buffers in serial production lines to maximise throughput. The results of the computational experiments indicate that the proposed algorithm is very efficient in terms of both solution quality and CPU time requirements. Moreover, the characterisation study yields interesting findings that may lead to important practical implications. A comprehensive bibliography is also provided in the paper.

Keywords: Serial production lines; Buffer allocation; Bottleneck

1. Introduction

In this paper, we study the unpaced, asynchronous serial production lines with reliable stations. Specifically, our objective is two-fold:

(i) To characterise the optimal buffer allocation and
(ii) to develop a new heuristic for general production lines with both reliable and unreliable stations to allocate buffers between stations to maximise throughput.

A serial line consists of serially ordered stations (machines, cells, etc) with buffers between them and units (part, component, sub-assembly, or product) move through these stations sequentially. These systems are extensively studied in the literature for various design (long-term) and operational (short-term) issues, since even a small change in the design parameters of such high-volume production lines leads to significant savings or losses in system performances. Although majority of the previous work has concentrated on the throughput measure (see the review article of Dallery and Gershwin 1992), performance measures other than throughput (i.e. the interdeparture time variability and average WIP inventory) have also been considered in recent studies (Miltenburg 1987, Martin and Lau 1990,

*Corresponding author. Email: sabun@bilkent.edu.tr

International Journal of Production Research
ISSN 0020–7543 print/ISSN 1366–588X online © 2006 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/00207540500465535

Hendricks and McClain 1993, So 1997, Papadopoulos and Vidalis 2001a, Kim and Lee 2001, Erel *et al*. 2002).

Previous studies indicate that the design parameters such as line-length, buffer size, buffer allocation, location of bottleneck stations, reliability of stations, variability of station processing times all have significant effects on system performances. As indicated in Conway *et al*. (1988), however, buffer size and allocation appear to be the most prominent factors for the throughput measure (production rate). In the relevant literature, there are also several studies to determine optimal buffer allocation and some rules-of-thumb (guidelines) recommended for practitioners.

In this study, we develop a new buffer allocation scheme and test its performance on various benchmark problems. The results indicate that the proposed algorithm is very efficient in terms of quality of solution and computational requirements. Moreover, we study the characterisation of optimal buffer allocation in serial production lines and draw conclusions leading to important managerial implications (i.e. recommending buffer allocation policies based on promising designs for a given system configuration). We employ an efficient enumeration technique for small size problems and a heuristic for large size problems.

The rest of the paper is organised as follows. In section 2, we present a literature review on buffer allocation. Brief explanation of the production model is given in section 3. The main structure of the proposed algorithm is discussed in section 4. The results of computational experiments that highlight the relative performance of the proposed algorithm with respect to the others reported in the literature are given in section 5; a characterisation of good buffer allocations are also given in this section. Finally, concluding remarks and further research directions are given in section 6.

2. Literature review

We present the literature review in two parts:

(i) characterisation of buffer allocation in production lines with reliable stations, and
(ii) overview of buffer allocation algorithms.

2.1 *Characterisation of buffer allocation*

The most prominent study in this area is due to Conway *et al*. (1988) who analyse serial production lines with reliable stations having uniform and exponential processing times. The authors draw several useful generalisations from their extensive simulation experiments. Some of the important results from this seminal work are:

1. Buffers improve throughput at a decreasing rate.
2. Buffers should be placed symmetrically throughout the line with more emphasis at the centre.
3. When a single buffer is to be assigned, its best location in the line is the same place where the effect of an infinite buffer is maximum.

In another study a three-station, asynchronous, unbalanced line with different mean processing times and variances is analysed (Powell 1994). His results indicate that the buffers should be assigned to bottleneck stations by favouring the sides that face towards the centre of the line. The author also reports that optimal buffer allocation is more sensitive to mean processing times than to the variance of the processing times. In a follow up study, Powell and Pyke (1996) consider the same asynchronous production line above and examine the effects of various design factors on optimal buffer allocation such as the coefficient of variation, line length, and multiple-bottlenecks. Their main results are:

1. Significant imbalances in mean processing times of stations (by introducing bottleneck stations) are needed to shift the optimal buffer allocation of the balanced line.
2. The best design for highly unbalanced lines is to place almost all buffers around the bottleneck station.
3. Bottleneck stations close to the centre of the line attract more buffers easily.
4. Line length has a little impact on general behaviour of unbalanced lines.

Hillier and So (1991a) examine the effect of the coefficient of variation of the processing times on the optimal buffer allocation in balanced lines with processing times having two-stage Coxian-type distributions. Their results based on Markovian analysis indicate that the centre stations should be supported with more buffers in lines with higher variability.

Hillier *et al.* (1993) study balanced lines with identical exponential processing times. The authors state that characterising the optimal buffer allocation is extremely difficult due to the discrete nature of decision variables. The main findings of this study are:

1. Total number of buffer spaces to be allocated has more impact on throughput than the pattern of buffer allocation.
2. The optimal buffer allocation typically follows an inverted bowl pattern (assigning more buffers to centre stations).

Harris and Powell (1999) analyse buffer allocation patterns for both balanced and unbalanced serial production lines with reliable stations having log-normal processing times. They examine several designs such as lines with a single-bottleneck and multiple bottlenecks, and lines with imbalances in mean and variances. The results confirm the conjectures made by Powell and Pyke (1996) about the effects of multiple bottlenecks on buffer allocation.

Balanced and unbalanced lines with exponential and Erlang processing times have been studied (Hillier 2000). The results indicate that processing time variability has a small impact on the pattern of buffer allocation. An inverted bowl pattern for buffer allocation is generally optimal, but the bowl shape becomes more definite with larger numbers of buffer spaces. For the unbalanced lines, the buffer allocation pattern deviates from the bowl phenomenon in response to favouring bottleneck stations.

Despite the work mentioned above, we can conclude that the characterisation of optimal buffer allocation is far from complete, since most of the work regarding characterisation consists of small-size systems (i.e. limited number of buffer slots and short lines), and as the problem size increases, finding optimal or near optimal solution becomes extremely difficult.

2.2 *Buffer allocation algorithms*

The algorithms developed for buffer allocation problem can be classified as evaluative (descriptive) algorithms and search (generative) algorithms (Papadopoulos and Spinellis 2000a). Evaluative algorithms are descriptive in nature that estimate the system performance such as throughput, and average WIP; simulation (Conway *et al*. 1988), traditional Markovian state models (Heavey *et al*. 1993) and decomposition methods (Sevastyanov 1962, Gershwin 1987, Dallery and Frein 1993) are among the examples in this category. Search algorithms focus on the optimisation of the system performance using decision variables such as buffer capacities of the buffer locations along the line, and workload allocations to the stations. Classical search methods such as the Hooke-Jeeves method, knowledge-based methods, dynamic programming-based methods, and various heuristic procedures fall into this category (see Altiok and Stidham 1983, Jafari and Shantikumar 1989, Hillier and So 1991a, Hillier and So 1991b, Yamashita and Altiok 1998, Diamantidis and Papadopoulos 2004). The algorithm proposed in this study also falls into this category.

The algorithms used to solve the optimal buffer allocation problem with the objective of maximising throughput are generally applied to specific types of production lines such as lines with reliable/unreliable stations, or balanced/unbalanced lines (see Ho *et al*. 1979, Chow 1987, Papadopoulos and Vouros 1998, Papadopoulos and Vidalis 1998, Papadopoulos and Spinellis 2000a, Papadopoulos and Spinellis 2000b, Gurkan 2000, Papadopoulos and Vidalis 2001b). In this study, however, we propose an algorithm that can be applied to any production line and compare its performance against similar algorithms in the literature. These algorithms are explained below.

Seong *et al*. (1995) develop two search algorithms, SEVA (standard exchange vector algorithm) and non-SEVA (non-standard exchange vector algorithm), to solve the buffer allocation problem. Both of them are local search algorithms that evaluate a specific neighbourhood as long as better solutions are obtained. The neighbourhood is found via the concept of standard exchange vectors for SEVA and the concept of pseudo-gradient and gradient projection is used in the implementation of non-SEVA. There are also two versions of non-SEVA based on the rounding procedure of the throughput gradient function; both large and small steps are taken to approximate the gradient of the throughput function in version 1 (non-SEVA1) whereas only small steps are used in the second version (non-SEVA2).

Gershwin and Schor (2000) consider unreliable lines with deterministic operation times. They analyse two buffer allocation problems: a primal and a dual problem. The objective of the primal problem is to minimise total number of buffers subject to a throughput constraint, whereas dual problem maximises throughput with fixed numbers of buffers. They solve the dual problem by means of a gradient method. First, an initial allocation and a search direction are determined, and then a linear search is made in that direction until the maximum throughput is achieved. A new direction is chosen and the same procedure is applied until no improvement is observed (see Schor 1995 and Gershwin and Schor 2000).

Harris and Powell (1999) develop a simulation search algorithm (SSA) that employs simulation to estimate the throughput for alternative allocations. The algorithm is based on a simplex search with a search direction to the highest throughput candidate. The algorithm starts with an initial allocation and then

generates its neighbours by transferring a buffer from a selected buffer location. The authors use the Spendley-Hext reflection procedure (Spendley and Hext 1962) to find the search direction.

In another study, Selvi (2002) proposes a simulation-based heuristic procedure, called LIBA, to solve the same problem. The main feature of the procedure is to minimise the difference between the throughput values of two sub-lines created by dividing the line around each buffer location and to transfer buffers from the faster sub-line to the slower one. The idea stems from the fact that throughput of the whole line is bounded by the minimum of the throughput values of these two sub-lines. The line is first divided into two sub-lines at the centre buffer location. Then potential *givers* and *receivers* are determined to accomplish the buffer transfers. In order to determine the potential giver, the initial sub-line (faster line) is again divided into two sub-lines from the central buffer location of the initial sub-line. The process of dividing the lines continues until two sub-lines remain with at most one buffer location. Among these two final sub-lines, the one with higher throughput is labelled as the potential giver. The potential receiver is determined similarly in such a way that the sub-line with smallest throughput is labelled as the potential receiver. Buffers are transferred from the giver to the receiver until no improvement in throughput is observed. Selvi (2002) also suggests an initial solution procedure as explained below:

First, a criticality function (cr_i) for any buffer location B_i is determined by using the following equation:

$$cr_i = \frac{1}{(\rho_i + \rho_{i+1})}$$

where ρ_i is the production rate of station i in isolation (this is further discussed in section 3). Then, a relative criticality (RC_i) for any buffer location B_i is determined as:

$$RC_i = \frac{cr_i}{(cr_1 + cr_2 + \cdots + cr_{N-1})}.$$

The number of buffer slots that are initially allocated to buffer location B_i, IB_i, is calculated by multiplying RC_i by the total number of buffer slots B. If IB_is are not integer, the integer part of IB_is are assigned as the capacity of each location. The remaining buffer slots are assigned to buffer locations in the decreasing order of the decimal part of their IB_i values. If the decimal parts are equal, the buffer location with higher integer part in IB_i gets the buffer. In the case of the exact equality of IB_is, the buffer location that is closer to the centre of the line gets the buffer.

3. Production line model

In this section, we describe the production line considered in our study in more detail. The serial production line consists of N stations ($S_1, S_2, \ldots, S_N$) and $N-1$ finite-capacity buffer locations ($B_1, B_2, \ldots, B_N$) in tandem (see figure 1). The parts enter the line at the first station, visit intermediate stations, and leave the line at the last station. Stations are subject to blocking and starving: a station is considered to be blocked if its part cannot be transferred to the downstream buffer, and is said

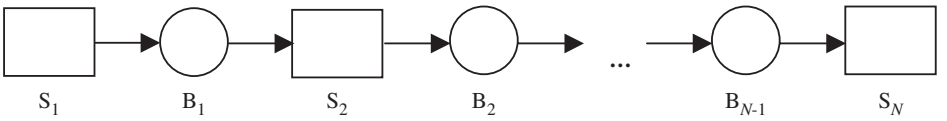

Figure 1. The N-station production line.

to be starved if it is idle due to lack of parts in the upstream buffer. Moreover, the following assumptions are used in the model:

1. There is an infinite buffer capacity at the upstream of the first station as well as at the downstream of the last station.
2. The time to transfer parts through the buffers is zero.
3. Stations do not fail when they are idle.
4. When a failure occurs, the part being processed remains on the station; it continues to be processed as soon as the station becomes available.

Prior to the description of the proposed algorithm in the next section, we give some well-known formulae to introduce some relationships for production lines:

$$\lambda_i = 1/T_i$$
$$\mu_i = 1/MTTF_i$$
$$r_i = 1/MTTR_i$$
$$e_i = MTTF_i/(MTTF_i + MTTR_i) = r_i/(\mu_i + r_i)$$
$$\rho_i = \lambda_i e_i = \lambda_i r_i/(\mu_i + r_i)$$

where T_i, average processing time of station i; λ_i, processing rate of station i; μ_i, average failure rate of station i; $MTTF_i$, mean time to failure for station i; $MTTR_i$, mean time to repair for station i; e_i, efficiency of station i in isolation; r_i, average repair rate of station i.

Note that $e_i = 1$ for reliable stations, and $e_i < 1$ for unreliable stations. Hence, the isolated production rate is equal to the processing rate for any reliable station.

4. Proposed algorithm

The proposed algorithm is based on the idea of transferring buffers from under-utilised buffer locations to fully utilised locations so that throughput is improved by means of an efficient buffer allocation. Typically buffer locations around bottleneck stations have higher potential to be receivers whereas the remaining under-utilised locations are considered as potential givers.

4.1 *Preliminaries*

In the algorithm, we use isolated production rates to distinguish bottleneck stations (this idea was used by Papadopoulos and Vidalis 2001b and to determine initial buffer allocation). Specifically, the stations that have isolation production rates below or equal to R_{avg} are considered to be bottleneck stations, where $R_{avg} = (\rho_1 + \cdots + \rho_N)/N$

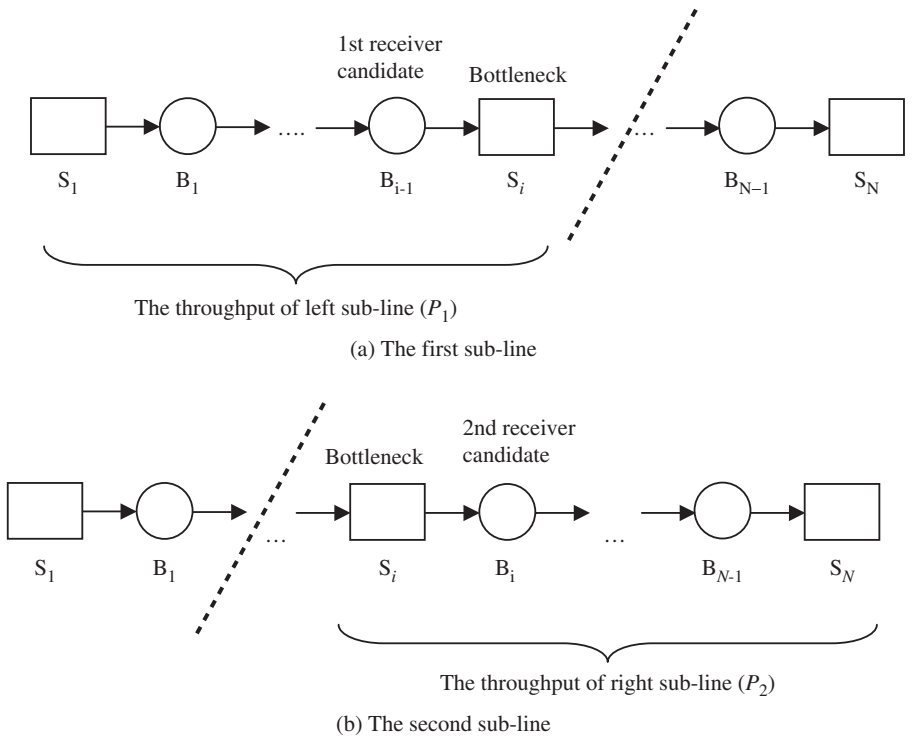

Figure 2. A schematic view of division process.

(average production rate of the N-station line). Note that this type of bottleneck is defined as mean bottleneck. It is also possible to create a different type of bottleneck by inflating the standard deviation of processing times; in other words, the stations that have standard deviations above or equal to the average standard deviation of all stations are considered to be bottlenecks. These types of bottleneck are called variance bottlenecks. In the proposed algorithm, we first consider mean bottlenecks and then consider the variance bottlenecks.

Furthermore, we use a factor called criticality measure of a buffer location proposed by Selvi (2002), $F(\rho_i, \rho_{i+1}) = 1/(\rho_i + \rho_{i+1})$, to determine the order in which buffer locations are considered. Utilising the above measure to allocate buffers leads to acceptable designs, since it conforms to the following generalisations given by Freeman (1964):

1. Avoid extreme buffer allocation, even in highly unbalanced lines.
2. Allocate more buffers to the station with the highest mean downtime.
3. Allocate more buffers between a bad and a mediocre station than a bad and a good station.

In order to determine a receiver, we first identify the most severe bottleneck station in the system (the one with the smallest ρ). Then we divide the line into two sub-lines at this bottleneck; the first sub-line consists of stations one through the bottleneck (figure 2(a)) and the second one consists of stations from the bottleneck to the last station (figure 2(b)). Note that the bottleneck station is included in both

of the sub-lines. The throughput of these two sub-lines is estimated using simulation. The buffer location of the bottleneck that lies in the sub-line with smaller throughput is labelled as the receiver. As can be intuitively expected, if the bottleneck is the first or the last station, there is no need to divide the line into smaller sub-lines since the receiver is the buffer location facing to the centre of the line. Note also that the same procedure is repeated for the other bottleneck stations in the ascending order of their isolation production rates. The rationale behind the division process is as follows:

It is known that the throughput values of the sub-lines are upper bounds for the throughput of the entire line: $P < \min(P_1, P_2)$ where P is the throughput of the whole line, P_1 and P_2 are the throughput values of the left and right sub-lines, respectively. Hence, if the minimum of P_1 and P_2 is increased, a better upper bound for P is obtained.

Let the first candidate seen in figure 2(a) be arbitrarily selected as the receiver. We keep in mind that buffer exchange is only performed if it increases P. As a result, buffer transfer from any location to that receiver will improve P_1; whereas P_2 will remain unchanged or decrease during buffer transfer. This is because of the fact that, if the giver belongs to the left sub-line, P_2 will not be affected; otherwise P_2 will decrease. As a result, if the left sub-line is slower than the right sub-line, selecting the first candidate as receiver most probably will increase the upper bound for P. If the left sub-line is faster than the right one, the second candidate must be chosen as a receiver to increase the upper bound for P.

Hence, in order to improve the upper bound for P, it is reasonable to initially select the candidate location that lies in the sub-line with smaller throughput as the receiver. Note that the throughput values of the sub-lines may be equal to each other; in such cases, the location closer to the end of the line is initially considered as the receiver.

We determine the potential givers based on their criticality measures. Buffer locations are ranked in the ascending order of their criticality values, excluding the bottleneck buffer locations; the location with the smallest value is labelled as the giver. Hence, a buffer transfer starts from the least critical location and relatively less amount of time is spent to determine the giver. Note that the location of the bottleneck that lies at opposite side of the receiver is the last giver. Moreover, if the criticality values of any two locations are equal, the one being farther away from the current bottleneck is chosen as the first giver.

4.2 *The structure of the algorithm*

The proposed algorithm consists of four main stages:

1. Determination of receivers.
2. Determination of givers.
3. Buffer transfer from givers to receivers.
4. Further improvement by considering the buffer locations which have potential to be receivers.

In the algorithm, initially a receiver (one of the buffer locations of the most severe bottleneck station) and a giver (the buffer location with the smallest criticality value) are selected. Then buffers are transferred from the giver one at a time to the receiver until no improvement in throughput is observed. This buffer transfer is repeated with

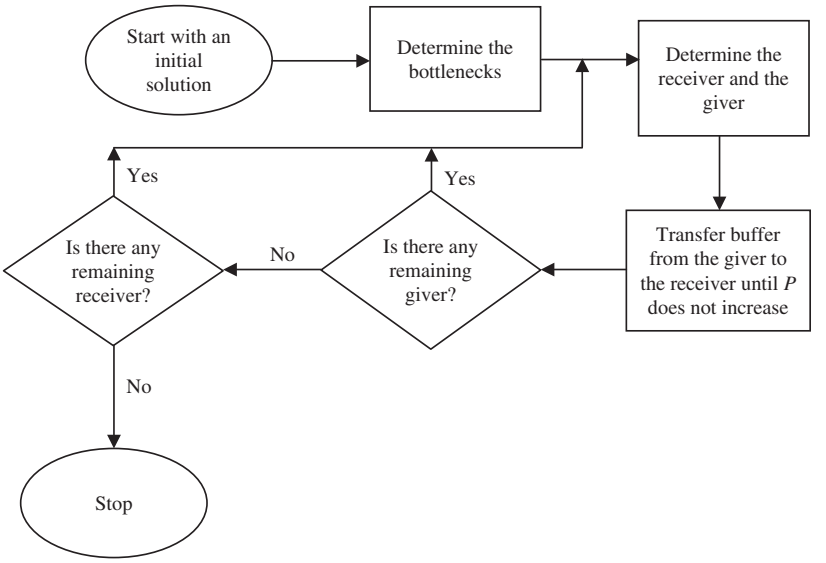

Figure 3. Flowchart of the algorithm.

other givers. Once all the givers are exhausted (i.e. all the potential givers are tested), a new receiver is identified and the same process is repeated until all receivers are considered.

In the process of identification of a receiver, two candidates are considered: the first one is the buffer location at the opposite side of the previous receiver and the second candidate is the one that is determined by the division process explained in section 4.1. These two candidates are then subjected to a marginal analysis in which the effect of assigning an extra buffer on throughput is assessed. The candidate with the highest contribution is selected as the receiver. At the final stage, the algorithm checks if throughput can be further improved by transferring buffers from the previous receivers to the locations that have potential to increase the throughput of the line. We consider $\lceil N/4 \rceil$ locations as receivers, one by one, by alternating around the centre, where $\lceil x \rceil$ is the smallest integer greater than or equal to x. Note that these locations were not considered as receivers in the previous steps. Accordingly, we not only consider the centre-of-line locations as receivers, but also try to increase throughput by buffering previous givers that may be highly unbuffered during the previous steps of the algorithm. In this step, the first receiver is the buffer location which is exactly at the centre of the line; however, if the number of stations is odd, there will be two central buffer locations. In this case, the one near the beginning of the line is considered as the first receiver. The flow diagram (figure 3) and the pseudo code of the algorithm are given next.

Notation:

j station index, $j = 1, \ldots, N$
S set of the bottleneck stations
B_j jth bottleneck location, $j = 1, \ldots, N-1$
G set of givers (all buffer locations except the current receiver)

$g_{current}$ current giver
C_i set of ith receiver candidate, $i = 1, 2$
$r_{current}$ current receiver.

Steps of the algorithm

Step 0. (INITIALISE): Start with an initial solution.

Step 1. (IDENTIFY BOTTLENECK)
Step 1.1. Calculate $R_{avg} = (\rho_1 + \cdots + \rho_N)/N$.
Step 1.2. Set $S = \{S_i : \rho_i \leq R_{avg}\}$.
Step 1.3. Set the current bottleneck to S_j where $j = \arg\min\{\rho_i : S_i \in S\}$ and update $S = S \setminus \{S_j\}$. In case of tie, select the station closest to the centre.

Step 2. (IDENTIFY RECEIVER)
Step 2.1. If $|C_2| > 0$, then set $r_{current}$ to the element of C_2, set $C_2 = \emptyset$, and go to Step 3.
Step 2.2. Divide the line into two sub-lines, L_1 and L_2, at the current bottleneck S_j and evaluate their production rates.
Step 2.3. If $|C_1| > 0$, then go to Step 2.6.
Step 2.4.1. If L_1 is slower, then set $r_{current} = B_{j-1}$ and $C_1 = C_1 \cup \{B_j\}$.
Step 2.4.2. Else, set $r_{current} = B_j$ and $C_1 = C_1 \cup \{B_{j-1}\}$.
Step 2.5. Go to Step 3.
Step 2.6. If L_1 is the slower sub-line, then set $C_2 = C_2 \cup \{B_{j-1}\}$, else set $C_2 = C_2 \cup \{B_j\}$.
Step 2.7. Compare C_1 and C_2 by marginality analysis.
Step 2.8.1. If C_1 is selected as the receiver, set $C_1 = \emptyset$, and go to Step 3.
Step 2.8.2. Else set $C_2 = \emptyset$, and replace B_j or B_{j-1} (whichever is not selected in Step 2.8.1) with the element in C_1.

Step 3. (IDENTIFY GIVER AND TRANSFER BUFFERS)
Step 3.1. Determine the set of givers G.
Step 3.2. While $|G| > 0$ do.
Step 3.2.1 Rank G in the ascending order of their criticality values with the last element being the location at the opposite side of the $r_{current}$.
Step 3.2.2. Set the location with minimum criticality measure to $g_{current}$. In case of tie, select the station farthest from the bottleneck.
Step 3.2.3. Update $G = G \setminus \{g_{current}\}$.
Step 3.2.4. Transfer one buffer from $g_{current}$ to $r_{current}$ until no improvement in throughput is observed.
Step 3.3. If $|C_2| > 0$, then return to Step 2.1.
Step 3.4.1. If $|S| > 0$, then return to Step 1.3.
Step 3.4.2.1. Else, if $|C_1| > 0$, then set $r_{current}$ to the element of C_1, set $C_1 = \emptyset$, and go to Step 3.1.
Step 3.4.2.2. Else, go to Step 4.

Step 4. (FURTHER IMPROVEMENT)
Step 4.1. Let number of receivers $m = \lceil N/4 \rceil$.
Step 4.2. Identify the receivers from the middle position to the right and left by alternating and oscillating around the centre. Label them as $R_1, \ldots, R_m$.

Step 4.3. Identify the givers from the previous receivers (identified in Steps 1–3) in the reverse generation order. Label them as $G_1, \ldots, G_k$.
Step 4.4. Transfer buffer units one at a time from G_is to each R_i until no improvement is observed.

Step 5. (STOP THE ALGORITHM)

Numerical example

We clarify the steps of our algorithm with an example problem taken from Seong *et al*. (1995). It is also the problem instance10 in our experiments given in table 1. The system consists of four stations where the processing, failure and repair times are exponentially distributed and total available buffer space is 30. The isolation production rates are 1.63, 0.65, 1.18, and 1.69, respectively. Note that $R_{avg} = 1.29$ with stations 2 and 3 being bottlenecks. The first bottleneck is station 2 (the most severe bottleneck). The right sub-line consists of stations 2, 3, and 4 whereas the left sub-line consists of stations 1 and 2. The throughputs of the left and right sub-lines are 0.6421 and 0.6379, respectively. This indicates that the location downstream of the bottleneck (the second buffer location) is more critical and it is labelled as the first receiver. The criticality values of the buffer locations are 0.4386, 0.5464, and 0.3484, respectively. The first giver is the third buffer location and it is followed by locations 1 and 2.

Table 1. Parameters for production lines reported in Seong *et al*. (1995). Failure and repair times are exponentially distributed; processing times are deterministic for the cases from 1 to 8, whereas they are exponential for the remaining cases.

Case	N	B	μ_i, r_i, λ_i of station i, $i = 1, \ldots, N$
1	3	15	(0.03, 0.09, 1) (0.03, 0.09, 1) (0.03, 0.09, 1)
2	4	30	(0.35, 0.7, 1) (0.2, 0.8, 1) (0.23, 0.7, 1) (0.05, 0.6, 1)
3	5	12	(0.2, 0.7, 1) (0.2, 0.7, 1) . . . (0.2, 0.7, 1)
4	10	47	(0.03, 0.03, 1) (0.03, 0.03, 1) . . . (0.03, 0.03, 1)
5	5	110	(0.14, 0.3, 1) (0.1, 0.4, 1) (0.2, 0.4, 1) (0.2, 0.5, 1) (0.17, 0.4, 1)
6	5	200	(0.1, 0.2, 0.1) (0.2, 0.3, 1) (0.3, 0.4, 1) (0.2, 0.4, 1) (0.1, 0.3, 1)
7	8	110	(0.001, 0.95, 1) (0.1, 0.85, 1) (0.05, 0.9, 1) (0.02, 0.95, 1) (0.15, 0.7, 1) (0.2, 0.6, 1) (0.25, 0.5, 1) (0.05, 0.9, 1)
8	9	155	(0.3, 0.7, 1) (0.2, 0.6, 1) (0.4, 0.7, 1) (0.2, 0.5, 1) (0.2, 0.6, 1) (0.1, 0.5, 1) (0.4, 0.7, 1) (0.3, 0.8, 1) (0.2, 0.5, 1)
9	4	10	(0.07, 0.17, 3.7) (0.11, 0.37, 1.5) (0.49, 0.78, 1.1) (0.19, 0.5, 3)
10	4	30	(0.38, 0.45, 3) (0.3, 0.55, 1) (0.35, 0.5, 2) (0.45, 0.4, 0.36)
11	5	10	(0.1, 0.3, 1.2) (0.3, 0.5, 1) (0.5, 0.2, 3) (0.4, 0.3, 2) (0.2, 0.1, 1.8)
12	5	15	(0.3, 0.64, 2.8) (0.4, 0.83, 1.7) (0.45, 0.75, 2.5) (0.35, 0.85, 3.4) (0.1, 0.74, 1.9)
13	5	115	(0.08, 0.4, 2.6) (0.24, 0.4, 3) (0.2, 0.6, 3.4) (0.17, 0.5, 4.7) (0.1, 0.3, 1.5)
14	6	130	(0.3, 0.2, 3) (0.5, 0.5, 1) (0.1, 0.3, 1.2) (0.2, 0.1, 1.8) (0.3, 0.2, 1.5) (0.4, 0.3, 2)
15	8	125	(0.25, 0.52, 1) (0.18, 0.48, 3.6) (0.23, 0.58, 1.7) (0.32, 0.5, 1.4) (0.19, 0.47, 2.8) (0.35, 0.46, 2.7) (0.26, 0.66, 1.6) (0.2, 0.41, 1.2)
16	9	200	(0.2, 0.7, 2.5) (0.1, 0.6, 1.5) (0.3, 0.8, 2.8) (0.2, 0.8, 3.6) (0.1, 0.7, 2.1) (0.1, 0.6, 1.9) (0.3, 0.8, 2.7) (0.2, 0.5, 3) (0.3, 0.6, 2)
17	10	310	(0.1, 0.8, 2.7) (0.15, 0.3, 1.8) (0.3, 0.5, 2.1) (0.2, 0.5, 2.3) (0.27, 0.8, 1.6) (0.3, 0.7, 2.7) (0.12, 0.6, 1.5) (0.2, 0.6, 1.5) (0.13, 0.6, 1.2) (0.3, 0.4, 2.6)
18	10	315	(0.365, 0.465, 2.4) (0.215, 0.565, 1.7) (0.305, 0.485, 2.8) (0.375, 0.455, 2.2) (0.34, 0.455, 2.1) (0.39, 0.39, 2.5) (0.265, 0.5, 1.1) (0.285, 0.49, 1.3) (0.255, 0.495, 1.6) (0.24, 0.505, 0.8)

Starting with a uniform allocation, in which 10 buffer slots are evenly allocated to each buffer location, and by transferring buffers from the givers to the first receiver, we obtain the new solution as (10, 16, 4). The second bottleneck is station 3 and the buffer locations near the bottleneck are 2 and 3. It is also known from previous step (Step 2) that location 1 is one of the receiver candidates. The other receiver candidate is location 3 since location 2 is considered as the receiver at the previous step. Thus, we conduct the marginality analysis for buffer locations 1 and 3. The results indicate that location 1 yields higher throughput with an extra buffer. After buffers are transferred from givers to the receiver, the new solution becomes (11, 15, 4). Since there remain no bottleneck stations, we continue to transfer buffers from the other givers to the last receiver (location 3 in our case). But this does not yield any improvement in the current solution. Finally, we execute Step 4 of the algorithm to see any further improvement in throughput. However, since all of the locations have already been considered as receivers, the algorithm stops with the solution of (11, 15, 4) and the resulting throughput value of 0.6385. Note that the throughput of the initial solution, 0.6337, is improved by 0.7%.

5. Computational results

This section consists of two parts:

(i) comparison of the proposed method against other heuristics, and
(ii) characterisation of the optimal buffer allocation in serial lines.

5.1 *Evaluation of the proposed method*

We compare the proposed method against six heuristics reported in the literature. These are LIBA, SEVA, two versions of non-SEVA (non-SEVA1 and non-SEVA2), simulation search algorithm (SSA), and dual algorithm (DA). We compare our method with LIBA using the same data set; however, we estimate the throughput values of the allocations of SEVA, non-SEVA, SSA and DA via simulation.

In the comparison with SEVA, non-SEVA, and LIBA, our proposed approach is implemented with two different initial solution schemes. The resulting algorithms are called heuristic-1 (H-1) and heuristic-2 (H-2). H-1 starts with a uniform allocation, whereas H-2 utilises the solution of LIBA as the initial solution. To compare the procedures, we use the 18 problem instances reported in Seong *et al.* (1995). The parameters of these problems are given in table 1. As for the comparison of SSA, we select a sample of the instances suitable for comparison reported in Harris and Powell (1999), since the majority of the instances in this set are used to characterise buffer allocation. These 12 instances belong to reliable and unbalanced lines with lognormal processing times that have a single mean bottleneck.

As for the comparison of DA, we use four instances reported in Schor (1995), which belong to unreliable and balanced/unbalanced lines. Four methods are generated for each of the first two instances. For each of these instances, the first and second algorithms start with the same initial solution, however the latter requires additional processing; similarly, the third and fourth algorithms start with the same initial solution, whereas the latter requires additional processing to improve the solution. Furthermore, in the first three cases, the stations have identical

deterministic processing times and geometrically distributed failure and repair times. The probability that the ith station fails during an operation is p_i; its probability of being repaired while it is under repair is r_i. For the last case, the stations have different processing rates and exponentially distributed failure and repair times (see table 2 for parameters of the instances).

The solutions of the problem instances generated by these algorithms are evaluated via simulation for throughput measure. Simulation is implemented via Java. Simulation run length of 200 000 entities and 25 independent replications are used in the experiments. Since we estimate steady-state performance of the system, the first 10 000 observations, corresponding to transient state, are discarded from the simulation runs. The results of statistical analysis indicate that the 95% confidence interval for throughput has a half-length less than 0.0001 according to the paired-t test.

The results of the comparison of our method with SEVA, LIBA and non-SEVA indicate that H-1 yields better solutions than the best of SEVA and non-SEVA in 10 cases and the same solution for the remaining eight problems in terms of the throughput measure (see table 3). As compared to LIBA, H-1 finds better solutions in three cases; the same solutions are obtained for the remaining 15 cases. We also give the computational requirements of the proposed algorithm in terms of the number of iterations (each iteration corresponds to an N-station throughput evaluation); the proposed algorithm is reasonably efficient and it is significantly faster than LIBA for large size problems. As can be seen in table 3, H-1 requires fewer numbers of iterations than do both SEVA and non-SEVA in nine out of 18 cases. If the algorithm starts with the initial solution of LIBA, it yields different solutions (in terms of throughput) in only two out of 18 cases. Furthermore, for Cases 1, 3, and 4 the solution of H-2 is not given because the initial solution of LIBA for these cases yields a uniform allocation.

As compared to SSA, H-1 yields better solution in six designs and the same solution in five designs (SSA performs better in only one case) in the 12-instance problem set of Harris and Powell in terms of throughput; the results are shown in table 4. As for the comparison of DA, H-1 outperforms DA in two cases, and it yields the same solution (in terms of throughput) in remaining two cases (see table 5).

In conclusion, H-1 is a computationally fast procedure that results in high-quality solutions. Hence, we use H-1 in the following section to characterise the optimal buffer allocation.

5.2 *Characterisation of buffer allocation*

We characterise the optimal buffer allocation under various experimental conditions. As seen in table 6, two buffer levels, three bottleneck locations, and two bottleneck levels are considered. Processing times are generated from lognormal distribution (Knott and Sury 1987, Buzacott and Shantikumar 1993). In the non-bottleneck case (base-line condition) the mean and the standard deviation of the processing times are 1 and 0.5 time units, respectively. Hence, the base-level for coefficient of variation (cv) is 0.5. In the bottleneck cases, it is increased to 1.2 and 2 depending on the severity level of bottleneck situation. In the simulation experiments, line lengths are set to 9 and 15, which is sufficiently long to extract necessary information from simulation experiments (our previous experiences indicated that short lines do not lead to meaningful results whereas long lines require excessive computation times).

Table 2. Parameters for the cases reported in Schor (1995). The processing rates of the stations in all cases are deterministic, and they are equal to 1 in Cases 1–3.

Case	N	B	Par.	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
1	10	346	p_i	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.007		
			r_i	0.095	0.095	0.095	0.095	0.095	0.095	0.095	0.095	0.095	0.095		
2	12	243	p_i	0.037	0.015	0.02	0.03	0.03	0.01	0.02	0.02	0.02	0.03	0.03	0.01
			r_i	0.35	0.15	0.4	0.4	0.3	0.2	0.3	0.3	0.4	0.4	0.3	0.25
3	5	31	$1/p_i$	20	167	22	22	26							
			$1/r_i$	11	19	12	7	7							
4	7	54	$1/p_i$	820	5700	870	830	970	1900	1100					
			$1/r_i$	450	760	460	270	270	650	320					
			$1/\mu_i$	40	34	39	38	37	40	43					

Table 3. Comparison of proposed algorithm with SEVA, non-SEVA and LIBA. (SEVA* refers to best of SEVA and non-SEVA in terms of throughput).

Case	Method	Allocation	P	Iter.	Percent. diff. from SEVA*
1	LIBA	(7, 8)	0.5786	3	0
	SEVA	(7, 8)	0.5786	8	
	Non-SEVA1	(7, 8)	0.5786	8	
	Non-SEVA2	(7, 8)	0.5786	8	
	H-1	(7, 8)	0.5786	4	
2	LIBA	(18, 9, 3)	0.6658	14	0
	SEVA	(20, 8, 2)	0.6658	48	
	Non-SEVA1	(19, 9, 2)	0.6658	25	
	Non-SEVA2	(20, 8, 2)	0.6658	29	
	H-1	(18, 9, 3)	0.6658	12	
	H-2	(18, 9, 3)	0.6658	11	
3	LIBA	(3, 3, 3, 3)	0.6465	11	0.1
	SEVA	(2, 4, 4, 2)	0.6461	34	
	Non-SEVA1	(2, 4, 4, 2)	0.6461	25	
	Non-SEVA2	(2, 4, 4, 2)	0.6461	30	
	H-1	(3, 3, 3, 3)	0.6465	12	
4	LIBA	(0, 5, 7, 8, 7, 9, 6, 5, 0)	0.1760	614	0.1
	SEVA	(0, 3, 8, 8, 9, 8, 8, 3, 0)	0.1758	333	
	Non-SEVA1	(0, 5, 5, 10, 10, 5, 5, 5, 2)	0.1751	41	
	Non-SEVA2	(0, 4, 7, 8, 9, 8, 6, 4, 1)	0.1758	58	
	H-1	(0, 4, 7, 8, 8, 8, 8, 4, 0)	0.1760	104	
5	LIBA	(24, 31, 33, 22)	0.6528	73	0.04
	SEVA	(22, 27, 38, 23)	0.6524	141	
	Non-SEVA1	(22, 28, 36, 24)	0.6525	36	
	Non-SEVA2	(22, 28, 36, 24)	0.6525	43	
	H-1	(27, 27, 34, 22)	0.6528	28	
	H-2	(27, 28, 33, 22)	0.6528	28	
6	LIBA	(36, 107, 45, 12)	0.5701	129	0
	SEVA	(37, 107, 45, 11)	0.5701	164	
	Non-SEVA1	(44, 100, 44, 12)	0.5701	92	
	Non-SEVA2	(37, 106, 46, 11)	0.5701	57	
	H-1	(38, 104, 45, 13)	0.5701	70	
	H-2	(38, 101, 49, 12)	0.5701	62	
7	LIBA	(2, 4, 0, 5, 16, 66, 17)	0.6666	91	0
	SEVA	(19, 0, 3, 0, 15, 60, 13)	0.6666	226	
	Non-SEVA1	(16, 16, 0, 16, 16, 32, 14)	0.6633	36	
	Non-SEVA2	(8, 8, 6, 8, 9, 51, 20)	0.6666	46	
	H-1	(0, 3, 0, 11, 16, 65, 15)	0.6666	69	
	H-2	(0, 3, 0, 8, 17, 65, 17)	0.6666	66	
8	LIBA	(10, 23, 38, 15, 12, 23, 22, 12)	0.6288	440	0.05
	SEVA	(10, 26, 36, 13, 10, 22, 27, 11)	0.6284	597	
	Non-SEVA1	(13, 24, 32, 19, 13, 18, 20, 16)	0.6285	76	
	Non-SEVA2	(11, 28, 36, 13, 11, 20, 24, 12)	0.6284	118	
	H-1	(12, 20, 36, 15, 11, 27, 21, 13)	0.6288	103	
	H-2	(12, 20, 35, 15, 12, 27, 21, 13)	0.6288	95	
9	LIBA	(0, 7, 3)	0.6258	6	0
	SEVA	(0, 7, 3)	0.6258	26	
	Non-SEVA1	(0, 6, 4)	0.6228	16	
	Non-SEVA2	(0, 7, 3)	0.6258	24	

(continued)

Table 3. Continued

Case	Method	Allocation	*P*	Iter.	Percent. diff. from SEVA*
	H-1	(0, 7, 3)	0.6258	8	
	H-2	(0, 7, 3)	0.6258	7	
10	LIBA	(11, 16, 3)	0.6384	21	0
	SEVA	(11, 16, 3)	0.6384	53	
	Non-SEVA1	(11, 16, 3)	0.6384	37	
	Non-SEVA2	(11, 16, 3)	0.6384	24	
	H-1	(11, 15, 4)	0.6384	14	
	H-2	(11, 15, 4)	0.6384	12	
11	LIBA	(1, 3, 4, 2)	0.3468	16	0.2
	SEVA	(1, 3, 4, 2)	0.3468	31	
	Non-SEVA1	(1, 3, 4, 2)	0.3468	48	
	Non-SEVA2	(1, 3, 4, 2)	0.3468	48	
	H-1	(1, 3, 3, 3)	0.3475	15	
	H-2	(1, 3, 3, 3)	0.3475	15	
12	LIBA	(4, 7, 3, 1)	0.9714	15	0.2
	SEVA	(4, 6, 3, 2)	0.9702	74	
	Non-SEVA1	(4, 6, 4, 1)	0.9696	25	
	Non-SEVA2	(4, 6, 4, 1)	0.9696	25	
	H-1	(4, 7, 3, 1)	0.9714	15	
	H-2	(4, 7, 3, 1)	0.9714	15	
13	LIBA	(17, 17, 22, 59)	1.1248	39	0
	Non-SEVA1	(29, 15, 29, 42)	1.1248	35	
	Non-SEVA2	(22, 11, 22, 60)	1.1248	27	
	H-1	(21, 27, 16, 51)	1.1248	34	
	H-2	(20, 31, 11, 53)	1.1248	35	
14	LIBA	(17, 39, 23, 39, 12)	0.4797	267	3.9
	Non-SEVA1	(26, 45, 26, 16, 17)	0.4618	58	
	Non-SEVA2	(23, 38, 47, 14, 8)	0.4453	83	
	H-1	(17, 35, 26, 40, 12)	0.4799	67	
	H-2	(17, 33, 28, 39, 13)	0.4801	55	
15	LIBA	(57, 8, 17, 17, 5, 8, 13)	0.6755	213	1.4
	Non-SEVA1	(33, 2, 18, 0, 21, 19, 32)	0.6660	85	
	Non-SEVA2	(33, 2, 18, 0, 21, 19, 32)	0.6660	92	
	H-1	(48, 13, 18, 15, 4, 10, 17)	0.6755	83	
	H-2	(54, 6, 18, 17, 5, 8, 17)	0.6755	101	
16	LIBA	(28, 47, 20, 15, 21, 20, 22, 27)	1.2780	292	0.2
	Non-SEVA1	(42, 42, 8, 25, 25, 25, 8, 25)	1.2758	50	
	Non-SEVA2	(70, 44, 15, 17, 16, 16, 6, 16)	1.2610	80	
	H-1	(25, 44, 4, 18, 18, 24, 18, 49)	1.2783	93	
	H-2	(28, 41, 12, 10, 20, 25, 18, 46)	1.2784	86	
17	LIBA	(6, 22, 15, 15, 21, 20, 41, 112, 58)	0.9860	819	0.04
	Non-SEVA1	(34, 34, 34, 2, 34, 34, 34, 34, 70)	0.9707	56	
	Non-SEVA2	(20, 15, 16, 18, 25, 15, 20, 112, 69)	0.9856	60	
	H-1	(6, 25, 14, 22, 22, 20, 39, 110, 52)	0.9860	283	
	H-2	(6, 25, 14, 20, 25, 23, 39, 106, 52)	0.9860	256	
18	LIBA	(29, 18, 26, 31, 30, 38, 49, 40, 54)	0.5417	80	0
	Non-SEVA1	(35, 35, 35, 35, 18, 52, 18, 35, 52)	0.5417	57	
	Non-SEVA2	(65, 20, 5, 5, 5, 65, 5, 5, 140)	0.5417	31	
	H-1	(35, 22, 35, 35, 35, 35, 35, 35, 48)	0.5417	48	
	H-2	(29, 24, 26, 31, 30, 38, 49, 40, 48)	0.5417	36	

Table 4. Comparison of proposed algorithm with SSA. The processing time distributions of non-bottleneck stations are log-normal (1, 0.5); mean of the bottleneck station is 1.25.

Case	N	Bottleneck location	B	Method	Allocation	P	Percent. diff. from SSA
1	4	3	3	SSA	(1, 1, 1)	0.7509	0
				H-1	(1, 1, 1)	0.7509	
2	4	3	6	SSA	(1, 3, 2)	0.7857	0
				H-1	(1, 3, 2)	0.7857	
3	4	3	9	SSA	(2, 3, 4)	0.7950	0.1
				H-1	(2, 4, 3)	0.7957	
4	6	4	5	SSA	(1, 1, 1, 1, 1)	0.7443	0
				H-1	(1, 1, 1, 1, 1)	0.7443	
5	6	4	10	SSA	(1, 2, 3, 2, 2)	0.7842	0.3
				H-1	(1, 2, 3, 3, 1)	0.7863	
6	6	4	15	SSA	(1, 3, 4, 5, 2)	0.7964	0.04
				H-1	(1, 3, 5, 4, 2)	0.7967	
7	8	5	7	SSA	(1, 1, 1, 1, 1, 1, 1)	0.7372	0
				H-1	(1, 1, 1, 1, 1, 1, 1)	0.7372	
8	8	5	14	SSA	(1, 2, 2, 3, 2, 2, 2)	0.7838	0.3
				H-1	(1, 1, 3, 3, 3, 2, 1)	0.7863	
9	8	5	21	SSA	(1, 2, 3, 5, 4, 3, 3)	0.7971	0.1
				H-1	(1, 2, 3, 6, 5, 2, 2)	0.7979	
10	10	6	9	SSA	(1, 1, 1, 1, 1, 1, 1, 1, 1)	0.7340	0
				H-1	(1, 1, 1, 1, 1, 1, 1, 1, 1)	0.7340	
11	10	6	18	SSA	(1, 2, 2, 2, 3, 3, 2, 2, 1)	0.7873	0.01
				H-1	(1, 1, 2, 2, 4, 3, 2, 2, 1)	0.7874	
12	10	6	27	SSA	(1, 3, 2, 3, 5, 6, 3, 3, 1)	0.7981	0.1
				H-1	(1, 1, 4, 3, 6, 6, 3, 2, 1)	0.7975	

Table 5. Comparison of proposed algorithm with dual algorithm (DA). (DA* refers to best of DA in terms of throughput).

Case	N	B	Method	Allocation	P	Percent. diff. from DA*
1	10	346	DA-1.1	(26, 38, 43, 44, 44, 44, 42, 39, 26)	0.8729	0.01
			DA-1.2	(26, 38, 43, 44, 44, 44, 42, 39, 26)	0.8729	
			DA-1.3	(27, 38, 43, 44, 44, 44, 42, 38, 26)	0.8729	
			DA-1.4	(26, 38, 43, 44, 44, 44, 42, 39, 26)	0.8729	
			H-1	(30, 39, 43, 43, 39, 45, 38, 40, 29)	0.8730	
2	12	243	DA-2.1	(58, 26, 21, 25, 23, 16, 15, 13, 14, 20, 12)	0.8871	0.04
			DA-2.2	(58, 27, 21, 25, 23, 16, 15, 12, 14, 20, 12)	0.8871	
			DA-2.3	(58, 27, 21, 25, 23, 16, 15, 13, 14, 19, 12)	0.8870	
			DA-2.4	(58, 27, 21, 25, 23, 16, 15, 12, 14, 20, 12)	0.8871	
			H-1	(47, 31, 14, 29, 26, 19, 20, 12, 15, 18, 12)	0.8875	
3	5	31	DA	(7, 10, 10, 4)	0.5075	0
			H-1	(5, 12, 11, 3)	0.5075	
4	7	54	DA	(8, 10, 13, 10, 9, 4)	0.0124	0
			H-1	(7, 11, 13, 9, 9, 5)	0.0124	

In order to ensure that the proposed algorithm is suitable for characterisation, we compare the solutions of nine-station designs against the optimal buffer allocation found by efficient enumeration (*EE*): In the last step (Step 4) of the original algorithm, instead of focusing on the $\lceil N/4 \rceil$ locations, each of the locations

Table 6. Experimental factors and their levels used in characterisation.

Factors	Levels
Line length	15
Total buffer size	14
	28
Bottleneck location	None
	Beginning of line
	Centre of line
Mean processing time of bottleneck	1.2
	2

which are not previously considered as receiver*s* are labelled as receiver*s* one by one, starting from the centre to the right and left of the line.

In the comparison of our heuristic with the solution of *EE*, we use 48 designs that have the following features: the line has a single extreme/standard bottleneck at the beginning/centre; the number of buffer units is equal to 8/16. The *cv* of the line is set at 6 different values: 0.25, 0.5, 0.75, 1, 1.25, and 1.5, respectively. Our heuristic yields the optimal solution in 44 out of 48 design points. In the remaining instances, the solutions of the proposed heuristic are very close to the optimal solutions in terms of the buffer allocation pattern. This encourages us to use the heuristic as a tool for characterising the optimal buffer allocation. The results are presented in three parts: single bottleneck, multiple bottleneck, mean and variance imbalance cases. Single bottleneck and multiple bottleneck cases are presented in two sub-sections: mean imbalance and variance imbalance cases. In the mean imbalance case, bottlenecks are formed by inflating the mean processing times whereas in the variance imbalance case only the variance is varied by keeping the mean constant at one time unit. Specifically, the standard deviation of the bottleneck station is increased up to three time units while keeping the standard deviation of the other stations at their original values (i.e. 0.5 time units). In the mean and variance imbalance cases, mean processing times and variance are both varied by keeping the mean and variances of the non-bottlenecks at the base level.

5.2.1 Single bottleneck.

5.2.1.1 *Mean imbalance case*. *Observation 1:* The increase in *cv* of processing times displays an interesting behaviour in buffer allocation: as *cv* of the line increases up to a certain threshold value, the location near the bottleneck draws more buffers towards itself. If the *cv* exceeds this threshold value, the effect of the bottleneck on buffer allocation diminishes. This behaviour, which we will call hump behaviour hereafter, is observed if the bottleneck is even at the middle (see table 7). This can be explained as follows:

As *cv* increases, coupling between all stations (i.e. blocking/starving frequency of each station) increases; but since the bottleneck is more critical in terms of the efficiency of the line, more buffers are drawn by the bottleneck to reduce coupling between the bottleneck and the stations near the bottleneck. However, after a certain threshold value of *cv*, the coupling between the non-bottleneck stations become

Table 7. Optimal buffer allocation obtained by *EE*. Line length is 9. Mean processing time of station 1 is 2.

Bottleneck station	Standard deviation	*B*	Allocation
1	0.25	8	(2, 1, 0, 1, 1, 1, 1, 1)
		16	(2, 2, 2, 2, 2, 2, 2, 2)
	0.5	8	(4, 1, 1, 1, 0, 1, 0, 0)
		16	(4, 2, 2, 2, 2, 2, 1, 1)
	0.75	8	(4, 1, 1, 1, 1, 0, 0, 0)
		16	(8, 2, 1, 1, 2, 1, 1, 0)
	1	8	(3, 1, 1, 1, 1, 1, 0, 0)
		16	(7, 2, 2, 2, 1, 1, 1, 0)
	1.25	8	(2, 1, 1, 1, 1, 1, 1, 0)
		16	(5, 2, 2, 2, 2, 2, 1, 0)
	1.5	8	(2, 1, 1, 1, 1, 1, 1, 0)
		16	(4, 2, 2, 2, 2, 2, 2, 0)
5	0.25	8	(1, 1, 1, 1, 1, 1, 1, 1)
		16	(2, 2, 2, 2, 2, 2, 2, 2)
	0.5	8	(0, 0, 0, 4, 4, 0, 0, 0)
		16	(1, 1, 2, 4, 4, 2, 2, 0)
	0.75	8	(0, 0, 1, 3, 3, 1, 1, 0)
		16	(0, 2, 1, 5, 6, 1, 1, 0)
	1	8	(0, 1, 1, 2, 2, 1, 1, 0)
		16	(0, 1, 2, 5, 5, 1, 2, 0)
	1.25	8	(0, 1, 1, 2, 2, 1, 1, 0)
		16	(1, 1, 2, 4, 4, 2, 1, 1)
	1.5	8	(0, 1, 1, 2, 2, 1, 1, 0)
		16	(1, 1, 2, 4, 4, 2, 1, 1)

Table 8. Optimal buffer allocation obtained by *EE*. Line length is 9. Mean processing time of station 1 is 2. Standard deviation of station 1 and station 2 is fixed at 0.5.

Standard deviation	*B*	Allocation
0.5	8	(4, 1, 1, 1, 0, 1, 0, 0)
	16	(4, 2, 2, 2, 2, 2, 1, 1)
0.75	8	(3, 2, 1, 1, 0, 1, 0, 0)
	16	(8, 1, 2, 2, 1, 1, 1, 0)
1	8	(2, 1, 1, 1, 1, 1, 1, 0)
	16	(4, 4, 2, 2, 1, 1, 1, 1)
1.25	8	(1, 2, 1, 1, 1, 1, 1, 0)
	16	(3, 3, 2, 2, 2, 2, 2, 0)
1.5	8	(1, 1, 1, 1, 1, 1, 1, 1)
	16	(2, 3, 2, 2, 2, 2, 2, 1)

so intense that the effect of the bottleneck station on buffer allocation in the entire system gradually decreases (see tables 8 and 9).

Observation 2: As the line length increases, the bottleneck begins to draw fewer buffers towards itself (figure 4). This is because of the fact that the increase in number of non-bottleneck stations increases the coupling in the system and, hence, non-bottleneck stations gradually begin to draw more buffers.

Table 9. Final buffer allocation obtained by the proposed heuristic. Line length is 15. Mean processing time of station 1 is 2.

Bottleneck station	Standard deviation	*B*	Allocation
1	0.25	14	(2, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
		28	(2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2)
	0.5	14	(5, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0)
		28	(5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 0)
	0.75	14	(6, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0)
		28	(11, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 0, 0, 0)
	1	14	(3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0)
		28	(9, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 1, 0, 0)
	1.25	14	(2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0)
		28	(6, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 0)
	1.5	14	(2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0)
		28	(4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0)
8	0.25	14	(0, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1)
		28	(2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2)
	0.5	14	(0, 0, 0, 1, 1, 1, 4, 5, 1, 1, 0, 0, 0, 0)
		28	(0, 1, 2, 2, 2, 2, 4, 5, 2, 2, 2, 2, 2, 0)
	0.75	14	(0, 0, 0, 1, 1, 1, 4, 4, 1, 1, 1, 0, 0, 0)
		28	(0, 0, 0, 1, 2, 2, 9, 9, 2, 1, 2, 0, 0, 0)
	1	14	(0, 0, 1, 0, 2, 1, 3, 3, 1, 1, 1, 1, 0, 0)
		28	(0, 0, 1, 2, 2, 2, 7, 7, 2, 2, 1, 2, 0, 0)
	1.25	14	(0, 0, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 0)
		28	(0, 0, 2, 2, 2, 2, 6, 6, 2, 2, 2, 1, 1, 0)
	1.5	14	(0, 0, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 0)
		28	(0, 1, 2, 2, 2, 2, 3, 4, 5, 2, 2, 2, 1, 0)

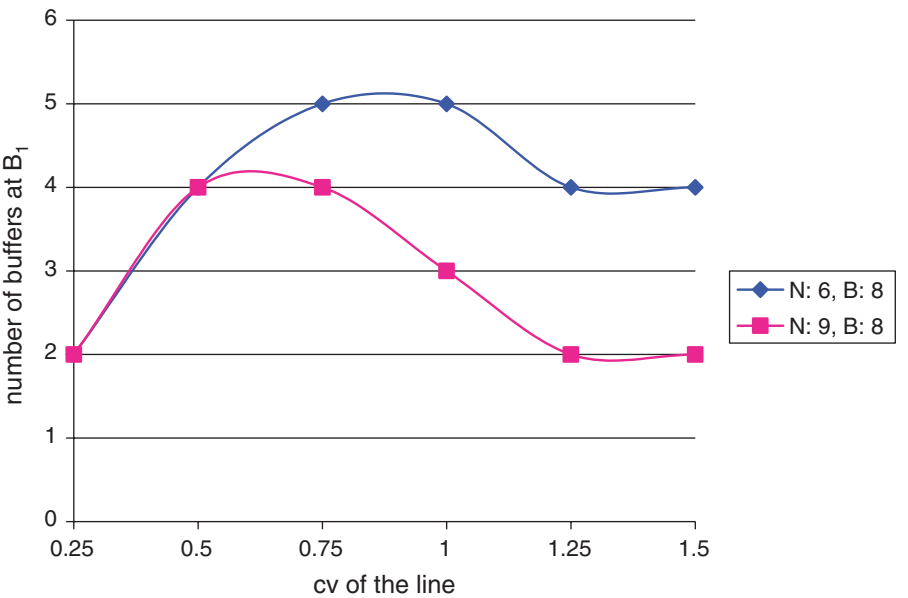

Figure 4. The number of buffer units drawn by the mean bottleneck with the increase in *cv*. Bottleneck is at Station 1 with a mean processing time of 2, and a fixed buffer capacity of 8 units.

Table 10. Final buffer allocation obtained by the heuristic. Line length is 15; standard deviation of non-bottleneck stations is 0.5.

Bottleneck station	*Cv* of bottleneck	*B*	Allocation
1	1	14	(2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0)
		28	(3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1)
	1.5	14	(2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0)
		28	(5, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1)
	2	14	(2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0)
		28	(7, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1)
	3	14	(3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0)
		28	(9, 2, 2, 2, 1, 2, 1, 2, 2, 1, 2, 1, 1, 0)
8	1	14	(0, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 0)
		28	(1, 1, 2, 2, 2, 2, 4, 3, 2, 2, 2, 2, 2, 1)
	1.5	14	(0, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 0)
		28	(1, 1, 1, 2, 2, 2, 5, 4, 2, 2, 2, 2, 2, 1)
	2	14	(0, 0, 1, 1, 1, 1, 3, 2, 1, 1, 1, 1, 1, 0)
		28	(1, 1, 1, 2, 2, 2, 5, 5, 2, 2, 2, 1, 1, 1)
	3	14	(0, 0, 1, 1, 1, 1, 3, 3, 1, 1, 1, 0, 1, 0)
		28	(0, 1, 2, 1, 2, 2, 6, 6, 2, 2, 1, 2, 1, 0)

5.2.1.2 *Variance imbalance case*. *Observation 3:* Similar to the mean imbalance case, the bottleneck station formed by inflating the standard deviation of the processing time attracts more buffers than the other stations. However, this level of attraction is not as intense as the mean imbalance case (see table 10). This can be explained as follows: Although the variance bottleneck reduces the efficiency of the line due to its higher *cv*, on the average it processes the same number of parts per unit time as non-bottleneck stations. In contrast, in the mean bottleneck case, the bottleneck station is the slowest station and hence reduces line efficiency directly.

5.2.2 Multiple bottlenecks. In this section, we examine the effect of two identical bottleneck stations located at different locations along the line. Specifically, these bottleneck stations are located at the extreme ends of the line, one at the beginning and the other in the middle, and both in the middle. Again, the analysis is carried out for mean and variance imbalance cases separately.

5.2.2.1 *Mean imbalance case*.

(i) *Mean bottleneck at station 1 and station 15*
Observation 4: If the severity of the bottleneck stations is high (i.e. the mean processing times of bottleneck stations are doubled), the bottleneck stations draw most of buffers toward themselves and a few buffer units remain at the centre of the line. As in the case of single bottleneck case, the drawing effect of bottleneck stations increases up to a certain threshold *cv* value; it decreases after *cv* exceeds that threshold value (see table 11).

(ii) *Mean bottleneck at Station 1 and Station 8*
Observation 5: The bottleneck station located at the centre of the line (Station 8) draws more buffer units than the other bottleneck station (Station 1). This result confirms the bowl phenomenon reported in the

Table 11. Final buffer allocation obtained by the proposed heuristic. Line length is 15. Mean processing time of the bottleneck stations is 2.

Bottleneck stations	Standard deviations	*B*	Allocation
1 and 15	0.5	14	(3, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 3)
		28	(4, 1, 2, 2, 2, 1, 3, 3, 2, 1, 1, 1, 1, 4)
	0.75	14	(3, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 3)
		28	(8, 0, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 7)
	1	14	(2, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 2)
		28	(6, 0, 1, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 6)
	1.25	14	(2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0)
		28	(4, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 4)
	1.5	14	(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
		28	(3, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 3)
1 and 8	0.5	14	(3, 1, 1, 1, 1, 1, 4, 1, 1, 0, 0, 0, 0, 0)
		28	(5, 0, 1, 1, 2, 0, 16, 2, 1, 0, 0, 0, 0, 0)
	0.75	14	(2, 1, 0, 1, 1, 1, 3, 3, 1, 1, 0, 0, 0, 0)
		28	(4, 2, 2, 1, 2, 2, 7, 4, 2, 2, 0, 0, 0, 0)
	1	14	(2, 1, 0, 1, 1, 1, 2, 2, 2, 1, 0, 1, 0, 0)
		28	(4, 1, 2, 2, 1, 1, 6, 5, 2, 3, 0, 1, 0, 0)
	1.25	14	(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
		28	(1, 2, 2, 2, 2, 2, 3, 2, 3, 2, 2, 2, 2, 1)
	1.5	14	(0, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 0)
		28	(1, 2, 2, 2, 2, 2, 3, 3, 3, 2, 2, 2, 2, 1)

literature (Erel *et al*. 2004). Furthermore, the buffer location facing the other bottleneck station, B_7, receives more buffer units in the final allocation.

(iii) *Mean bottleneck at Station 7 and Station 9*
Observation 6: As intuitively expected, the buffer locations between these two bottleneck stations draw most of buffers. Similar to the single bottleneck case, the number of buffers allocated around the bottleneck stations decreases as *cv* increases.

5.2.2.2 *Variance imbalance case*. *Observation 7:* The results of the variance imbalance case are similar to the ones of the mean imbalance cases. Only exception is that the level of buffer attraction of bottleneck stations is relatively low in the variance imbalance case.

5.2.3 Mean and variance imbalance case. In this section, we examine the optimal buffer pattern for the lines that have both mean and variance imbalance. Specifically, we consider three cases:

1. The mean bottleneck in the middle of the line, variance bottleneck at the beginning.
2. The high-mean and high-variance station at the beginning.
3. The high-mean and high-variance station in the middle.

Observation 8: The results show that mean bottleneck station draws a much higher number of buffers than do the variance bottleneck. This confirms the conjectures of the previous studies stating that mean imbalance has greater effect on the optimal buffer allocation than variance imbalance. However, if variance imbalance

Table 12. Final buffer allocation obtained by the proposed heuristic. Line length is 15. Mean bottleneck is at the beginning; variance bottleneck is in the middle. The *cv* of the line is 0.5.

Mean imbalance	Variance imbalance	*B*	Allocation
2	1	14	(3, 0, 0, 0, 0, 0, 0, 7, 4, 0, 0, 0, 0, 0, 0)
		28	(3, 0, 0, 1, 2, 4, 11, 5, 1, 0, 1, 0, 0, 0)
	1.5	14	(4, 0, 1, 0, 1, 1, 4, 3, 0, 0, 0, 0, 0, 0)
		28	(6, 1, 1, 2, 1, 1, 12, 4, 0, 0, 0, 0, 0, 0)
	2	14	(7, 0, 1, 0, 0, 1, 3, 2, 0, 0, 0, 0, 0, 0)
		28	(12, 1, 2, 2, 2, 2, 4, 3, 0, 0, 0, 0, 0, 0)
1.2	1	14	(2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0)
		28	(3, 2, 2, 2, 2, 2, 3, 3, 2, 2, 2, 1, 1, 1)
	1.5	14	(3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0)
		28	(4, 2, 2, 2, 2, 2, 3, 3, 2, 2, 1, 1, 1, 1)
	2	14	(3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0)
		28	(6, 2, 2, 2, 2, 1, 3, 3, 2, 1, 2, 1, 1, 0)

is extremely high, a bottleneck station draws more buffers than mean bottleneck (see table 12). Furthermore, the effect of bottlenecks on buffer allocation is greater when both mean and variance bottleneck stations are in the same location in the line.

6. Discussion

In this paper, we study unpaced and asynchronous serial production lines with reliable stations. Specifically, we develop a new heuristic procedure for production lines with reliable and unreliable stations. Our computational experiences indicate that the proposed algorithm is very efficient, in terms of both solution quality and computational requirements. In general, it yields better results than the existing methods on the problem data sets used in the experiments. In addition, we characterise the optimal buffer allocation. The main observations from this characterisation study are as follows:

- In the mean bottleneck case, we observe an interesting hump behaviour that the locations near the bottleneck draw more buffer units towards themselves as the *cv* of the line increases up to a certain value. If the *cv* exceeds this threshold value, the effect of the bottleneck on buffer allocations diminishes. This implies that buffer allocation is one of the most important design factors in labour intensive lines (non-automated lines) with high variability.
- As the line length increases, the bottleneck begins to draw fewer buffer units towards itself, with the increase in *cv*.
- Bottleneck draws buffer units towards itself; if the severity of the bottleneck is high, its effect on buffer allocation is high.
- The centre of the line intrinsically draws buffers toward itself (bowl phenomenon).
- Symmetrically placed bottlenecks have very little effect on buffer allocation unless the imbalance is very high. In the case of high imbalance, both of the bottlenecks draw significant numbers of buffer units.
- As compared to mean bottleneck, variance bottleneck has relatively less effect on the optimal buffer allocation.

- In the case of asymmetric bottlenecks, the one closer to the centre of the line draws more buffers.

As a further research topic, we plan to adapt the proposed buffer allocation algorithm to topologies such as assembly systems and disassembly systems. The characterisation study can also be extended to these systems so that good solution properties can be utilised to generate initial solutions for iterative algorithms in this area.

References

Altiok, T. and Stidham Jr., S., The allocation of interstage buffer capacities in production lines. *IIE Trans*., 1983, **15**, 292–299.

Buzacott, J.A. and Shantikumar, J.G., Stochastic Models of Manufacturing Systems, 1993 (Prentice Hall: Englewood Cliffs, NJ).

Chow, W., Buffer capacity analysis for sequential production lines with variable process times. *Int. J. Prod. Res*., 1987, **25**, 1183–1196.

Conway, R., Maxwell, W., Mclain, J.O. and Thomas, L.J., The role of work-in-process inventory in serial production lines. *Op. Res*., 1988, **36**, 229–241.

Dallery, Y. and Gershwin, S.B., Manufacturing flow line systems: a review of models and analytical results. *Queueing Syst*., 1992, **12**, 3–94.

Dallery, Y. and Frein, Y., On decomposition methods for tandem queuing networks with blocking. *Op. Res*., 1993, **41**, 386–399.

Diamantidis, A.C. and Papadopoulos, C.T., A dynamic programming algorithm for the buffer allocation problem in homogeneous asymptotically reliable serial production lines. *Math. Prob. in Eng*., 2004, **3**, 209–223.

Erel, E., Sabuncuoglu, I. and Kok, G.A., Analysis of assembly systems for interdeparture time variability and throughput. *IIE Trans*., 2002, **34**, 23–40.

Erel, E., Sabuncuoglu, I. and Kok, G.A., Analysis of serial production line systems for interdeparture time variability and WIP inventory. *Int. J. Op. & Quant. Manage*., 2004, **10**, 275–295.

Freeman, M.C., The effects of breakdowns and inter-stage storage on production line capacity. *J. Indust. Eng*., 1964, **15**, 194–200.

Gershwin, S.B. and Schor, J.E., Efficient algorithms for buffer space allocation. *Ann. Op. Res*., 2000, **93**, 117–144.

Gurkan, G., Simulation optimisation of buffer allocations in production lines with unreliable stations. *Ann. Op. Res*., 2000, **93**, 177–216.

Harris, J.H. and Powell, S.G., An algorithm for optimal buffer placement in reliable serial lines. *IIE Trans*., 1999, **31**, 287–302.

Heavey, C., Papadopoulos, H.T. and Browne, J., The throughput rate of multistation unreliable production lines. *Euro. J. Op. Res*., 1993, **68**, 69–89.

Hendricks, K.B. and McClain, J.O., The output processes of serial production lines of general stations with finite buffers. *Manage. Sci*., 1993, **39**, 1194–1201.

Hillier, F.S. and So, K.C., The effect of the coefficient of variation of operation times on the allocation of storage space in production line systems. *IIE Trans*., 1991a, **23**, 198–206.

Hillier, F.S. and So, K.C., The effect of station breakdowns and interstage storage on the performance of production line systems. *Int. J. Prod. Res*., 1991b, **29**, 2043–2055.

Hillier, F.S., So, K.C. and Boling, R.W., Notes: Toward characterising the optimal allocation of storage space in production line systems with variable processing times. *Manage. Sci*., 1993, **39**, 126–133.

Hillier, M.S., Characterising the optimal allocation of storage space in production line systems with variable processing times. *IIE Trans*., 2000, **32**, 1–8.

Ho, Y.C., Eyler, M.A. and Chien, T.T., A gradient technique for general buffer storage design in a production line. *Int. J. Prod. Res*., 1979, **17**, 557–580.

Jafari, M.A. and Shantikumar, J.G., Determination of optimal buffer storage capacitiesand optimal allocation in multistage automatic lines. *IIE Trans.*, 1989, **21**, 130–135.

Kim, S. and Lee, H.J., Allocation of buffer capacity to minimise average work-in-process. *Prod. Planning & Control.*, 2001, **12**, 706–716.

Knott, K. and Sury, R.J., A study of work-time distributions on unpaced tasks. *IIE Trans.*, 1987, **19**, 50–55.

Martin, G.E. and Lau, H.S., Dynamics of the output interval distribution in unpaced lines. *Comp. & Indust. Eng.*, 1990, **18**, 391–399.

Miltenburg, G.J., Variance of the number of units produced on a transfer line with buffer inventories during a period of length T. *Naval Res. Logist.*, 1987, **34**, 811–822.

Papadopoulos, C.T. and Spinellis, D.D., A simulated annealing approach for buffer allocation in reliable production lines. *Ann. Op. Res.*, 2000, **93**, 373–384.

Papadopoulos, C.T., Spinellis, D.D. and Smith, J.M., Stochastic algorithms for buffer allocation in reliable production lines. *Math. Prob. in Eng.*, 2000, **5**, 441–458.

Papadopoulos, H.T. and Vouros, G.A., Buffer allocation in unreliable production lines using a knowledge based system. *Comp. & Op. Res.*, 1998, **25**, 1055–1067.

Papadopoulos, H.T. and Vidalis, M.I., Optimal buffer storage allocation in balanced reliable production lines. *Int. Trans. in Op. Res.*, 1998, **5**, 325–339.

Papadopoulos, H.T. and Vidalis, M.I., Minimising WIP inventory in reliable production lines. *Int. J. Prod. Econ.*, 2001a, **70**, 185–197.

Papadopoulos, H.T. and Vidalis, M.I., A heuristic algorithm for the buffer allocation in unreliable unbalanced production lines. *Comp. & Indust. Eng.*, 2001b, **41**, 261–277.

Powell, S.G., Buffer allocation in unbalanced three-station serial lines. *Int. J. Prod. Res.*, 1994, **32**, 2201–2217.

Powell, S.G. and Pyke, D.F., Allocation of buffers to serial production lines with bottlenecks. *IIE Trans.*, 1996, **28**, 18–29.

Schor, J.E., Efficient algorithms for buffer allocation. MS thesis, MIT, Cambridge, MA 1995.

Selvi, O., The line balancing algorithm for optimal buffer allocation in production lines. MS thesis, Bilkent University, Ankara, Turkey, 2002.

Seong, D., Chang, S.Y. and Hong, Y., Heuristic algorithms for buffer allocation in a production line with unreliable stations. *Int. J. Prod. Res.*, 1995, **33**, 1989–2005.

Sevastyanov, B.A., Influence of storage bin capacity on the average standstill time of a production line. *Theory of Prob. and Its Appl.*, 1962, **7**, 429–438.

So, C.K., Optimal buffer allocation strategy for minimising work-in-process inventory in unpaced production lines. *IIE Trans.*, 1997, **29**, 81–88.

Spendley, W. and Hext, G.R., Sequential application of simplex designs in optimisation and evolutionary operations. *Technometrics*, 1962, **4**, 441–461.

Yamashita, H. and Altiok, T., Buffer capacity allocation for a desired throughput in production lines. *IIE Trans.*, 1998, **30**, 883–891.