Buffer Allocation via Bottleneck-Based Variable Neighborhood Search

Sixiao Gao 1,*, Toshimitsu Higashi 2, Toyokazu Kobayashi 2, Kosuke Taneda 2, Jose I. U. Rubrico 3 and Jun Ota 4

1 Department of Precision Engineering, The University of Tokyo, Tokyo 113-8654, Japan
2 Murata Machinery, Ltd., Aichi 484-0076, Japan; thigashi@me.com (T.H.); toyokazu.kobayashi@lny.muratec.co.jp (T.K.); kosuke.taneda@hkt.muratec.co.jp (K.T.)
3 Department of Mathematics, Physics, and Computer Science, University of the Philippines Mindanao, Davao City 8000, Philippines; jurubrico@up.edu.ph
4 Research into Artifacts Center for Engineering, School of Engineering, The University of Tokyo, Tokyo 113-8656, Japan; ota@race.t.u-tokyo.ac.jp
* Correspondence: gao@race.t.u-tokyo.ac.jp

Received: 14 September 2020; Accepted: 26 November 2020; Published: 30 November 2020

Abstract: This study addresses the challenging problem of efficient buffer allocation in production lines. Suitable locations for buffer allocation are determined to satisfy the desired throughput, while a suitable balance between solution quality and computation time is achieved. A throughput calculation approach that yields the state probability of production lines is adopted to evaluate the effectiveness of candidate buffer allocation solutions. To generate candidate buffer allocation solutions, an active probability index based on state probability is proposed to rapidly detect suitable locations of buffer allocations. A variable neighborhood search algorithm is used to maintain acceptable solution quality; an additional neighborhood structure is used in the case where no satisfactory solution is generated in the initial neighborhood structure. Extensive numerical experiments demonstrate the efficacy of the proposed approach. The proposed approach can facilitate agile design of production lines in industry by rapidly estimating production line topologies.

Keywords: efficient buffer allocation; bottleneck; variable neighborhood search; throughput calculation

1. Introduction

A production line consists of materials, storage area, and working area. Materials enter the production line, move between the storage area and the working area, and finally leave the production line. In queueing theory, materials are referred to as jobs, working areas as servers, and storage areas as buffers. Buffers are extremely important to compensate for variations and to ensure smooth operation of the servers. Excessive buffers increase costs and system redundancy while insufficient buffers lead to blocking in the production. Therefore, ensuring a suitable buffer allocation is important in a production line.

Agile manufacturing [1], which aims to respond rapidly to customer and market requirements, is widely adopted in industry. Agile design [1], which facilitates agile manufacturing, has attracted considerable attention from researchers and engineers. In agile design, buffers need to be efficiently allocated to predict production line design for meeting design requirements. The potential success and value of the design can then be evaluated within a short computation time. Therefore, it is necessary to achieve a good balance between solution quality and computation time for various kinds of production line topologies. Numerous studies have focused on the buffer allocation problem in...
production lines [2–5]. Buffer allocation methodologies can be generally categorized into enumeration, dynamic programming, metaheuristic methods, and search methods [5,6].

Enumeration is a method that employs all feasible buffer configurations to guarantee the optimality of the solution [7]. Although previous study [8] proposed the enumeration method to design buffers for the complete solution space, this method is limited to production lines with a small-scale topology because of its combinatorial complexity and time consumption. Although “small-scale” is relative and depends on the problem, the method is usually applicable to production line topologies with less than 20 machines.

Dynamic programming solves the buffer problem in a highly intuitive manner based on a non-closed-form objective function [9]. Diamantidis and Papadopoulos [10] used dynamic programming to address the problem of buffer allocation in serial production lines. Sadr and Malhamé [11] proposed a dynamic programming-based algorithm to design the buffer size in partially homogeneous unreliable transfer lines. Both methods are only applicable to limited production line topologies with a small number of machines. In summary, dynamic programming is usually applicable to production lines with a small-scale topology because the number of solution stages and states increases exponentially with the number of buffers and servers in the production line, which leads to an extremely long computation time.

Metaheuristic methods are high-level techniques that generate sufficiently good solutions for the buffer allocation problem. Yelkenci, Kose and Kilinc [12] proposed a hybrid evolutionary algorithm-based simulation optimization approach to decide the near optimal size of each buffer storage in open serial production lines. Costa et al. [13] combined a novel parallel tabu search algorithm with a neighborhood generation mechanism to determine a suitable buffer size for a serial production line. Beşikçi et al. [14] proposed a decision support system (DSS) employing an artificial neural network-based prediction model to determine the buffer allocation. Lin and Chiu [15] solved the stochastic resource allocation problem by combining hybrid particle swarm optimization with optimal budget computing allocation. These methods can obtain buffer allocation solutions with high solution quality. However, the main disadvantage of these methods is that one method is sufficiently effective only for the design of limited production line topologies. Moreover, the computation time of some metaheuristic methods depends on the initial buffer allocation solution, which may also lead to unstable solution quality.

Search methods select suitable locations to allocate buffers using some guide information. Their main disadvantage is that it is extremely difficult to achieve a balance between solution quality and computation time for different production line topologies. Some previous studies [16–20] achieved buffer allocation based on search methods. Specific parameters are used to rapidly determine suitable locations for buffer allocation. However, these methods are applicable to limited production line topologies such as serial topologies. Other studies [5,6] focused on various production line topologies and proposed buffer allocation approaches with effective guide information to allocate buffers. However, the computation time needs to be further decreased under the constraint of solution quality requirements if such approaches are to be used for efficient buffer allocation.

Although previous studies achieved buffer allocation with high solution quality, a long computation time is usually required to achieve the high solution quality, which does not meet computation time requirements of efficient buffer allocation. Hence, there is a need for a buffer allocation approach that can rapidly allocate buffers to the production line with acceptable solution quality. The objective of this study is to develop an efficient buffer allocation approach that achieves a good balance between solution quality and computation time. The solution quality of the buffer allocation should be higher than previous methods in available computation time. Furthermore, from the perspective of industry practical application, the proposed approach can be applied to a production line with 50 buffers and servers, which can be regarded as a large-scale production line; the available computation time of buffer allocation for this production line typically lies within 8 h.
In the study, a throughput calculation approach that can yield the state probability of a production line is adopted to evaluate the effectiveness of candidate buffer allocation solutions efficiently. The bottleneck-based variable neighborhood search algorithm is proposed to generate the candidate buffer allocation solutions efficiently. Based on state probability, the active probability index as a bottleneck indicator is proposed to search suitable buffer updating locations, which facilitates efficient buffer allocation by decreasing invalid buffer allocation attempts. Moreover, the variable neighborhood search algorithm is used to maintain acceptable solution quality using a proper additional neighborhood structure in the case no satisfactory solution is generated in the initial neighborhood structure. Finally, numerical examples are provided to evaluate the effectiveness of the proposed approach for a production line with a different topology and topology scale.

The remainder of this article is organized as follows. Section 2 presents the problem statement. The solution methodology is described in Section 3. Section 4 provides numerical examples to verify the effectiveness of the proposed solution methodology. The article ends with some conclusions and the scope for future work, in Section 5.

2. Problem Statement

In this study, the following assumptions are considered for production lines:

- Jobs enter a server or buffer in a production line at an external arrival rate according to a Poisson process that is widely used in the analysis of production lines [21].
- Jobs leave a server or buffer at a service rate with an exponential distribution that is widely used in the analysis of production lines [21].
- Both servers and buffers in a production line have their own service rates by which the working rate of the servers and buffers can be described.
- The capacities of the server and the buffer are set as “1” to simplify modelling; note that the capacity can be set as any value without affecting the effectiveness of the proposed approach. If the capacity of a server or a buffer is one, it implies that a server or buffer can deal with or contain only one job at a time.
- Although both blocking mechanisms—blocking before service and blocking after services—can be applied to the proposed approach to schedule job movement when blocking occurs in production lines [21], in the study, the former is adopted.
- Designed buffers that are allocated by the proposed approach have service rates that are equal to the highest service rate in an initial production line.

In [22], the goal of the dual buffer allocation problem is to maximize throughput of a production line subject to total designed buffer size.

In this study, computation time is also considered, and therefore, the efficient buffer allocation problem in this study is to find a buffer configuration vector for a production line to maximize throughput subject to available computation time and total designed buffer size decided by the available computation time. The problem can be described as

\[
\text{Find } B_{sol} = (b_{h,i}, \cdots) \\
\text{maximize } T_{sol} \\
\text{subject to } t_{sol} \leq t_{available}, \\
N_{sol} \leq N_{max}
\]

where \( B_{sol} \) represents the buffer configuration vector which is the combination of designed buffer size in the solution, \( i \) represents a node that describes a server or buffer, \( b_{h,i} \) represents the designed buffer size allocated between nodes \( h \) and its following node \( i \), \( T_{sol} \) is the obtained throughput in the solution, \( t_{sol} \) is the computation time of buffer allocation and \( t_{available} \) is the available computation time.
Furthermore, $N_{sol}$ is the designed buffer size and $N_{\text{max}}$ is the maximum designed buffer size, which is decided by $t_{\text{available}}$. The term $t_{\text{available}}$ typically lies within 8 h for a large-scale production line and 4 h for a small-scale production line.

3. Solution Methodology

Previous studies typically achieved the desired throughput by allocating several buffers at different locations in the production line. This method allows adding buffers at unsuitable locations; however, it increases the computation time of buffer allocation. Thus, to decrease unsuitable buffer allocation, it is effective to add one buffer at the bottleneck of a production line in each buffer update.

Figure 1 presents the framework of the proposed bottleneck-based variable neighborhood search algorithm. The Algorithm 1 presents the pseudocode of the proposed bottleneck-based variable neighborhood search algorithm. The algorithm detects the bottleneck of a production line using an active probability index. The initial neighborhood structure is set to detect the bottleneck with a small number of searches, which decreases the computation time of buffer allocation. When no satisfactory solution is generated in the initial neighborhood structure, an additional neighborhood structure is used to assess more candidate nodes, which improves the solution quality of the buffer allocation. The algorithm ends with no throughput increasing by allocating buffers or available computation time being exceeded.

![Figure 1. Framework of the bottleneck-based variable neighborhood search algorithm.](image-url)
Algorithm 1: bottleneck-based variable neighborhood search algorithm

1. Initialize parameters
2. $T_{sol} = 1$
3. Input arrival rates $\lambda_i$, service rates of nodes $\mu_i$, nodes $n_{od} = \{1, \cdots, i, l, \cdots\}$, edges $edg = \{(\cdots, (i, l), \cdots)\}$ between nodes, service discipline $sed = \{(\cdots, (i, FCFS), \cdots)\}$ at merging or splitting node $i$
4. while $T_{sol} > T_{comparison}$ and $t_{sol} < t_{available}$ do
5. $j = j + 1$
6. if $j = 1$ then
7. Calculate system state probability $sta$ by QMTAA
8. else
9. if $j > 2$ then
10. $T_{comparison} = T_{hig}^{j-1}$
11. end if
12. Update $\lambda_i$, $\mu_i$, $n_{od}$, $edg$, $sed$, $sta$
13. Calculate active probability index $\beta_i$ of node $i$ based on $sta$
14. Calculate total number of candidate bottlenecks $V_{bot}$
15. for $k = 1$ to initial search times $C_{ori}$ do
16. $V_{bot} = V_{bot} + C_{ori}$
17. Add one buffer before one candidate bottleneck based on $\beta_i$ from high value to low value and generate new $\lambda_i$, $\mu_i$, $n_{od}$, $edg$, $sed$.
18. Calculate throughput of the new $\lambda_i$, $\mu_i$, $n_{od}$, $edg$, $sed$ by QMTAA and put it into throughput set $thr$
19. Recover $\lambda_i$, $\mu_i$, $n_{od}$, $edg$, $sed$
20. end for
21. $T_{hig}^{j} = \max thr$
22. while $\frac{T_{hig}^{j} - T_{hig}^{j-1}}{T_{hig}^{j-1}} < \varepsilon$ and $V_{bot} > 0$ do
23. $V_{bot} = V_{bot} - C_{add}$
24. Set $thr = \emptyset$ and put $T_{hig}^{j}$ into $thr$
25. for $k = 1$ to additional search times $C_{add}$ do
26. Add one buffer before one candidate bottleneck based on $\beta_i$ from high value to low value and generate new $\lambda_i$, $\mu_i$, $n_{od}$, $edg$, $sed$.
27. Calculate throughput of the new $\lambda_i$, $\mu_i$, $n_{od}$, $edg$, $sed$ by QMTAA and put it into throughput set $thr$
28. Recover $\lambda_i$, $\mu_i$, $n_{od}$, $edg$, $sed$
29. end for
30. $T_{hig}^{j} = \max thr$
31. end while
32. $T_{sol} = T_{hig}^{j}$
33. $N_{sol} = N_{hig}^{j}$
34. $B_{sol} = B_{hig}^{j}$
35. end if
36. end while

3.1. Notations

This section presents the notations used in the proposed algorithm:

- $i$: A node that models a server or buffer.
- $n_i$: Number of jobs at node $i$.
- $\pi(n_i = k)$: Steady-state probability of $k$ jobs being at node $i$. 

• $\lambda_i$: External Poisson arrival rate at node $i$; i.e., the average number of jobs that enter a production line from node $i$ per unit time.
• $\mu_i$: Exponential mean service rate at node $i$; i.e., the average number of jobs that leave node $i$ per unit time.
• $j$: Number of buffer updates.
• $T_i$: Throughput at which jobs leave node $i$ per unit time.
• $T_{\text{high}}$: Temporary highest throughput rate.
• $T_{\text{high}}^j$: Obtained highest throughput rate at $j$th buffer update.
• $T_{\text{sol}}$: Throughput in the solution.
• $N_{\text{high}}$: Buffer size corresponding to the obtained highest throughput rate at $j$th buffer update.
• $N_{\text{sol}}$: Buffer size in the solution.
• $B_{\text{high}}^j$: Buffer configuration vector corresponding to the obtained highest throughput rate at the $j$th buffer update.
• $B_{\text{sol}}$: Buffer configuration vector in the solution.
• $C_{\text{ori}}$: Search times of the initial neighborhood.
• $C_{\text{add}}$: Search times of an additional neighborhood.
• $V_{\text{bot}}$: Number of nodes that may be the bottleneck.
• $V_{\text{mer}}$: Number of merging nodes.
• $V_{\text{spl}}$: Number of splitting nodes.
• $\beta_i$: Active probability index.

3.2. Queue Module-Based Throughput Analysis Approach (QMTAA)

The queue module-based throughput analysis approach (QMTAA) developed by Gao et al. [23] is efficient for calculating the throughput of production lines. Considering efficient buffer allocation, QMTAA is used in the study. Figure 2 show a production line model. In the modelling of QMTAA, products are called jobs; a server of a buffer is modelled as a node and denoted as $i$; service rate $\mu_i$ represents the performance of node $i$; and arrival rate $\lambda_i$ represents the external arrival jobs from input node $i$. At merging or splitting nodes, service disciplines determine job movement priority. The output parameters of QMTAA are throughput $T_i$. Figure 3 shows the framework of the throughput analysis in QMTAA. For more detailed information on the modelling and solution of QMTAA, the reader can refer to the literature [23].
3.3. Active Probability Index (API)

A bottleneck is a blocking point in a production line. A bottleneck indicator (BI), such as inventory [24] and active time [25], is an evaluation parameter that can be used to identify the bottleneck of the production line. The inventory indicator is the probability of full capacity, and a machine with a high inventory can easily become the bottleneck of a production line. The active time is the time when the machine is in the processing state; a machine with a long active time has a high possibility to be the bottleneck of a production line. Active time is effective in finding the real bottleneck [25]. However, QMTAA calculates the throughput of the production line based on steady-state probabilities but not on processing time. Therefore, based on the definition of the active time, the active probability index as a bottleneck indicator is proposed:

\[
\beta_i = \pi(n_i = 1, n_l = 0) + \pi(n_h = 1, n_i = 1, n_l = 0) \quad (5)
\]

Here, \(\pi(n_i = 1, n_l = 0)\) is the state probability of one job being at node \(i\) and no jobs being at its following node \(l\). The state of \(\pi(n_i = 1, n_l = 0)\) is described as shown in Figure 4; the red circle represents the job at node \(i\). The term \(\pi(n_h = 1, n_i = 1, n_l = 0)\) is the state probability of one job being at node \(h\) and its following node \(i\), respectively, while no jobs are at node \(l\). The state of \(\pi(n_h = 1, n_i = 1, n_l = 0)\) is described as shown in Figure 5.

![Figure 4. State of \(\pi(n_i = 1, n_l = 0)\).](image)

![Figure 5. State of \(\pi(n_h = 1, n_i = 1, n_l = 0)\).](image)

In the case of a high arrival rate, \(\pi(n_i = 1, n_l = 0)\) is effective to detect the bottleneck of a production line. However, when the external arrival rate is low, and few jobs are in a production line, the states of each node may be similar. This can decrease the effectiveness of \(\pi(n_i = 1, n_l = 0)\). Considering that jobs are usually blocked before the bottleneck as the state is shown in Figure 5, \(\pi(n_h = 1, n_i = 1, n_l = 0)\) is used to improve the effectiveness of \(\beta_i\) in the case of low arrival rates. Using QMTAA, the state

---

**Figure 3. Framework of QMTAA.**

---

**Figure 4. State of \(\pi(n_i = 1, n_l = 0)\).**

**Figure 5. State of \(\pi(n_h = 1, n_i = 1, n_l = 0)\).**
probabilities of nodes in a production line can be obtained via a throughput calculation process. By substituting the state probabilities into Equation (5), the specific values of the active probability index of each node can be obtained.

If node $i$ is a merging node as shown in Figure 6, two active probability indexes can be obtained as

$$
\beta_{i-g} = \pi(n_i = 1, n_l = 0) + \pi(n_k = 1, n_i = 1, n_l = 0)
$$

(6)

$$
\beta_{i-h} = \pi(n_i = 1, n_l = 0) + \pi(n_p = 1, n_i = 1, n_l = 0)
$$

(7)

where $\beta_{i-g}$ refers to the active probability index for the location between nodes $g$ and $i$; meanwhile $\beta_{i-h}$ refers to the active probability index for the location between nodes $h$ and $i$.

![Figure 6. Merging node $i$.](image)

If node $i$ is a splitting node as shown in Figure 7, there are two routes after node $i$. Because each job at node $i$ can only enter one route, two active probability indexes of two routes can be calculated, respectively. The maximum value between the two active probability indexes is used as the active probability index for node $i$.

$$
\beta_i = \max \left\{ \pi(n_i = 1, n_l = 0) + \pi(n_h = 1, n_i = 1, n_l = 0), \pi(n_i = 1, n_p = 0) + \pi(n_h = 1, n_i = 1, n_p = 0) \right\}
$$

(8)

![Figure 7. Splitting node $i$.](image)

3.4. Variable Neighborhood Search (VNS)

To achieve a balance between computation time and solution quality of buffer allocation, a proper initial neighborhood structure is set to detect the bottleneck of a production line rapidly, where an additional neighborhood structure is set to improve the solution quality in case no satisfactory throughput is generated in the initial neighborhood structure. Here, neighborhood structure refers to candidate bottlenecks.

3.4.1. Initial Neighborhood Structure

Neighborhood search times that are the number of candidate bottlenecks considered can be set to different values according to different problems. For instance, one choice is to set all possible neighborhoods as the neighborhood structure and the other choice is to select only a set of possible
neighborhoods [26]. Considering the effectiveness of the API and the efficient calculation of QMTAA, the initial neighborhood search times $C_{ori}$ is set as

$$C_{ori} = \min\left\{ \frac{V_{bot}}{2}, \sigma \right\}$$

(9)

where $V_{bot}$ is the number of total candidate bottlenecks.

According to the state probabilities of the nodes, the API values of the nodes can be obtained using Equation (5). According to the API values from high to low, $C_{ori}$ nodes are selected as initial candidate bottlenecks. If $V_{bot}/2$ is greater than $\sigma$, the initial neighborhood search time $C_{ori}$ is $\sigma$; otherwise, $C_{ori}$ is set as $V_{bot}/2$. According to our experiment results and error control, $\sigma$ is set to 10.

3.4.2. Neighborhood Change Criterion

The neighborhood change criterion is used when no satisfactory throughput is generated. Compared with the throughput $T_{hig}^{j-1}$ before one buffer update, if the current throughput $T_{hig}^{j}$ is improved by less than a fixed percentage, the neighborhood structure will be changed and additional neighborhood structure will be tested. The equation is described as

$$\frac{T_{hig}^{j} - T_{hig}^{j-1}}{T_{hig}^{j-1}} < \varepsilon$$

(10)

According to our experiment results and error control, $\varepsilon$ is set to 0.1%.

3.4.3. Additional Neighborhood Structure

If no satisfactory throughput is generated by the initial neighborhood structure, the neighborhood structure is expanded and additional candidate neighborhoods are tested. In general, the topology of a production line and the jobs flowing in it become more complex with increasing merging or splitting [27]; therefore, it is believed that the number of additional neighborhood searches should be related to the number of merging or splitting nodes in the production line. The number of additional neighborhood searches is set as

$$C_{add} = \max\left\{ V_{mer}, V_{spl} \right\}$$

(11)

where $V_{mer}$ and $V_{spl}$ are the numbers of merging and splitting nodes in the production line, respectively. For the application of additional neighborhoods, according to the API values from high to low, the initial candidate bottlenecks are ignored, and $C_{add}$ nodes are selected as the additional candidate bottlenecks. Then, one buffer is added before the additional candidate bottlenecks to search a better solution.

4. Numerical Example

The five comparison methods according to different search algorithms and neighborhood definition are listed in Table 1. The five methods are as follows:

- The proposed buffer allocation approach that generates candidate bottlenecks by the API and detects the real bottleneck by VNS, which is denoted by API-VNS.
- Park [24] used inventory to evaluate the candidate bottlenecks of a production line and used the beam search (BS) algorithm to search for the real bottleneck. Therefore, the inventory-based beam search algorithm (Inv-BS) is used as another method for comparison.
- Lutz et al. [28] and Papadopoulos et al. [29] used the tabu search (TS) algorithm to obtain suitable buffer allocation solutions. The solution quality of this method was shown to be high, and the computation time was controllable. Therefore, TS is used as a method for comparison. However, in prior studies, TS was typically used to maximize the throughput of a production line subject to a fixed number of designed buffers, whereas the designed buffer size is decided by available
computation time in the proposed approach. For the comparison, it is assumed that in each buffer update, one buffer is added and the TS attempts to maximize the throughput. If the obtained throughput does not increase even when buffers are added, or available computation time is exceeded, the buffer allocation process stops. Otherwise, the loop of one buffer addition and throughput calculation is performed. This extended tabu search algorithm is denoted by ETS. Moreover, in the numerical examples, a random mechanism is used to generate candidate buffer allocation solutions in ETS; therefore, the average values based on five calculations are used.

- To further evaluate the effectiveness of the API, the API-BS generated by BS [24] and the proposed API is set as the fourth method for comparison.
- To evaluate the effectiveness of VNS further, Inv-VNS generated by inventory [24] and the proposed VNS is set as the fifth method for comparison.

### Table 1. Five methods for comparison in the numerical example.

| Neighborhood Definition | API | Inventory [24] | Random |
|-------------------------|-----|----------------|--------|
| **Search algorithms**   | VNS | API-VNS        | Inv-VNS | ×     |
|                         | BS [24] | API-BS     | Inv-BS | ×     |
|                         | TS [29] | ×            | ×      | ETS   |

In the numerical examples, nodes in the production line models represent servers and buffers. Both blue and green nodes represent buffers; the blue nodes transport jobs, and the green nodes not only transport jobs but they also change their movement direction. Therefore, the service rates of blue nodes are higher than those of the green nodes. Servers marked by black nodes have the lowest service rates.

Two experiments are performed in the numerical examples to evaluate the effectiveness of the proposed approach for a production line with a different topology and topology scale. Each experiment involves two input patterns where all initial values except for the external arrival rates are the same. The two different external arrival rates are used to represent a high arrival job flow and a low arrival job flow. The effectiveness of the proposed approach is evaluated in the large-scale experiment and the available computation time is set as 8 h. The other experiment is presented to show the application of the proposed approach in a small-scale production line. Considering that there are a small number of nodes in the experiment, the available computation time is set as 4 h. All five methods are applied to the same sets of data. Moreover, all five methods are written in Python 3.6 and executed in all experiments on a personal computer with a 2.6 GHz Intel Core 2 Duo CPU.

To evaluate the comparison results, two aspects are discussed:

- Solution quality refers to the obtained solution throughput because a buffer allocation approach with a high solution quality usually generates a high solution throughput. In the case of the same solution throughput, the buffer allocation approach that generates the same throughput with fewer designed buffer sizes has a higher solution quality.
- Computation time refers to the computation time spent to achieve the same throughput, because an efficient buffer allocation can allocate buffers and achieve an objective throughput rapidly.
4.1. Small-Scale Topology Experiment

4.1.1. Setting

A production line with a small-scale topology was tested, as shown in Figure 8. The production line includes three servers (nodes 3, 6, and 9) and 12 buffers. All input parameters for experiment 1 are listed in Table 2.

![Small-scale topology](image)

**Figure 8.** Small-scale topology.

**Table 2.** Input parameters of experiment 1.

| Items                          | Parameters |
|-------------------------------|------------|
| Arrival rates                 | $\lambda_1 = \lambda$ (jobs/s) |
| Service rates of buffers 1, 4, 7, 11, 13, 15 | $\mu_1 = \mu_4 = \mu_7 = \mu_11 = \mu_{13} = \mu_{15} = \mu_{inv}$ (jobs/s) |
| Service rates of buffers 2, 5, 8, 10, 12, 14 | $\mu_2 = \mu_5 = \mu_8 = \mu_{10} = \mu_{12} = \mu_{14} = \mu_{con}$ (jobs/s) |
| Service rates of servers 3, 6, 9 | $\mu_3 = \mu_6 = \mu_9 = \mu_{ser}$ (jobs/s) |
| Service discipline of merging | First come, first served: jobs arriving at the merging node enter it first preemptively. |
| Service discipline of splitting | Random: jobs at the splitting node move to ensuing nodes randomly. |

4.1.2. Results

Table 3 lists the results of the five methods at the computation time when buffer allocation is finished. Figure 9 shows the computation time versus throughput curves for the five methods in the case of $\lambda = 0.4$. The results show that API-VNS and API-BS can generate higher throughput than the other three methods in the first 2 h. However, API-VNS achieved the same throughput more rapidly than API-BS. Subsequently, ETS obtained a higher throughput than API-VNS because of its high solution quality for available computation time. Figure 10 shows the computation time versus throughput curves for the five methods in the case of $\lambda = 0.6$. The results show that API-VNS can achieve the highest throughput among the comparison methods, which shows the advantage of its solution quality in available computation time. Moreover, compared to other four methods, API-VNS obtained the same throughput with least computation time.

Figures 11 and 12 show the scatter plots of the designed buffer size versus the solution throughput for the five methods in the cases of $\lambda = 0.4$ and $\lambda = 0.6$, respectively. In the case of $\lambda = 0.4$, ETS achieved the highest solution throughput, followed by API-VNS and API-BS. However, ETS allocated less designed buffers than API-VNS and API-BS. ETS adds several buffers in each buffer update, which test more candidate buffer allocation solutions. Therefore, it can achieve the highest solution quality in the experiment. Moreover, API-VNS and API-BS obtain the same solution throughputs as they both use API; Inv-BS and Inv-VNS also obtain the same solution throughput because both use the inventory indicator. Furthermore, API-VNS and API-BS have the highest solution quality because both achieve the highest solution throughput with the same designed buffer size.
Table 3. Results of the five methods in experiment 1.

| Pattern (\(\lambda, \mu_{nor}, \mu_{con}, \mu_{ser}\)) | Algorithm | Computation Time \(t_{sol}\) (s) | Obtained Throughput \(T_{sol}\) (jobs/s) | Designed Buffer Size \(N_{sol}\) | Obtained Buffer Configuration |
|-------------------------------------------------|-----------|---------------------------------|---------------------------------|-------------------------------|---------------------------------|
| (0.4, 1.0, 0.5, 0.2) | API-VNS | 1012 | 0.2403 | 13 | (10,2,1,0,0,0,0,0,0,0,0) |
| | Inv-BS | 1605 | 0.2200 | 10 | (1,0,2,0,1,0,3,0,3,0,0) |
| ETS | Average: | 14,400 | | | |
| | Average: | 14,400 | | | |
| | API-BS | 1476 | 0.2403 | 13 | (10,2,1,0,0,0,0,0,0,0,0) |
| | Inv-VNS | 1895 | 0.2200 | 10 | (1,0,2,0,1,0,3,0,3,0,0) |
| | API-VNS | 3644 | 0.2714 | 19 | (0,7,6,1,2,1,0,1,0,1,0) |
| (0.6, 1.0, 0.5, 0.2) | Inv-BS | 1148 | 0.2192 | 8 | (0,0,0,3,4,0,0,1,0,0,0) |
| ETS | Average: | 14,400 | | | |
| | Average: | 14,400 | | | |
| | API-BS | 1530 | 0.2650 | 14 | (0,7,7,0,0,0,0,0,0,0,0) |
| | Inv-VNS | 2399 | 0.2664 | 18 | (1,7,5,1,1,1,0,0,0,0,0,2) |
Figure 9. Computation time vs. throughput curves for the five methods in the case of $\lambda = 0.4$ in experiment 1.

Figure 10. Computation time vs. throughput curves for the five methods in the case of $\lambda = 0.6$ in experiment 1.

Figure 11. Scatter plot of designed buffer size vs. solution throughput for the five methods in the case of $\lambda = 0.4$ in experiment 1.
Figure 12. Scatter plot of designed buffer size vs. solution throughput for the five methods in the case of $\lambda = 0.6$ in experiment 1.

Unlike that in the case of $\lambda = 0.4$, API-VNS and API-BS obtain different solution throughputs, and Inv-BS and Inv-VNS also obtain different solution throughputs. API-VNS and API-BS use API as the BI to detect bottlenecks. However, the real bottleneck is not detected in the initial neighborhood structure, and VNS is used to search for more candidate bottlenecks. Therefore, API-VNS can achieve a higher solution throughput. Similarly, Inv-VNS achieves a higher solution throughput than Inv-BS. Compared to the API, the inventory indicator is not sufficiently effective in the experiment. Therefore, API-VNS can obtain a higher solution throughput than Inv-VNS in the case of $\lambda = 0.6$.

Figure 13 shows the production line topology after buffers are allocated by API-VNS in the case of $\lambda = 0.4$. A total of ten buffers were allocated before node 2, while three buffers were allocated in the branch with node 3. Because the external arrival rate is low and many jobs tend to arrive at node 14 rapidly via the branches with node 3, the splitting node 2 easily became the bottleneck; therefore, many buffers were allocated before it to compensate for the variation of job flow that attempts to enter via the branches with node 3. Figure 14 shows the production line topology after buffers are allocated by API-VNS in the case of $\lambda = 0.6$. A total of 13 buffers were allocated in the branch with node 3, meanwhile 4 buffers were allocated in the branch with node 6; only 2 buffers were allocated in the branches with node 9. Compared to the case of $\lambda = 0.4$, the external arrival rate increased in the case of $\lambda = 0.6$. Therefore, it is difficult for jobs to arrive at node 14 via only the branches with node 3. Jobs from splitting node 2 arrive at merging node 14 by the 3 branches with nodes 3, 6, and 9. Because the branch with node 3 is the shortest for the jobs to arrive at splitting node 14, most jobs tend to enter it, and more buffers are needed here compared to that at other branches.

Figure 13. Production line topology after buffers are allocated by API-VNS in the case of $\lambda = 0.4$ in experiment 1.
Figure 14. Production line topology after buffers are allocated by API-VNS in the case of $\lambda = 0.6$ in experiment 1.

4.2. Large-Scale Topology Experiment

4.2.1. Setting

A production line with a large-scale topology including 35 nodes was tested, as shown in Figure 15. The production line includes three servers (nodes 13, 16, and 19) and 32 buffers. All input parameters for experiment 2 are listed in Table 4.

Figure 15. Large-scale topology.
Table 4. Input parameters of experiment 2.

| Items                        | Parameters                             |
|------------------------------|----------------------------------------|
| Arrival rates                | $\lambda_1 = \lambda_4 = \lambda_7 = \lambda$ (jobs/s) |
| Service rates of buffers     | $\mu_1 = \mu_3 = \cdots = \mu_{33} = \mu_{35} = \mu_{con}$ (jobs/s) |
| Service rates of servers     | $\mu_2 = \mu_5 = \cdots = \mu_{31} = \mu_{34} = \mu_{con}$ (jobs/s) |
| Service discipline of merging| First come, first served: jobs arriving at the merging node enter it first preemptively. |
| Service discipline of splitting| Random: jobs at the splitting node move to ensuing nodes randomly. |

4.2.2. Results

Table 5 lists the results of the five methods at the computation time when buffer allocation is completed. Figures 16 and 17 show the computation time versus throughput curves for the five methods in the cases of $\lambda = 0.1$ and $\lambda = 0.2$, respectively. The results show that both API-VNS and API-BS can achieve the highest throughput among the five methods, even though API-VNS takes a little longer for the throughput solution compared to API-BS. This delay is attributed to the application of the variable neighborhood search. When the throughput obtained by API-VNS increases on an extremely small scale, the variable neighborhood search is triggered, and an additional neighborhood structure is used. Therefore, compared to API-BS, the computation time of buffer allocation for API-VNS is slightly longer. Moreover, for the production line with a large-scale topology, the computation time of API-VNS is less than 8 h. It is interesting that Inv-BS and Inv-VNS generated very different results, even though they used the same BI. Both Inv-VNS and Inv-BS use inventory as a bottleneck indicator. However, an additional neighborhood structure is used when the real bottleneck is not detected in the initial neighborhood structure for Inv-VNS. Therefore, Inv-VNS and Inv-BS generated very different results in the experiment.

![Figure 16](image_url)  
Figure 16. Computation time vs. throughput curves for the five methods in the case of $\lambda = 0.1$ in experiment 2.
Table 5. Results of the five methods in experiment 2.

| Pattern          | Algorithm | Computation Time $t_{sol}$ (s) | Obtained Throughput $T_{sol}$ (jobs/s) | Designed Buffer Size $N_{sol}$ | Obtained Buffer Configuration $B_{sol}$ |
|------------------|-----------|--------------------------------|------------------------------------|---------------------------------|---------------------------------|
| $(A_r, \mu_{nor}, \mu_{con}, \mu_{ser})$ | API-VNS   | 21,430                         | 0.2263                             | 15                              | $(0,0,0,0,0,0,0,0,1,0,0,0,1,9,2,0,0)$ |
| $(0.1, 1.0, 0.5, 0.2)$ | Inv-BS    | 633                             | 0.2033                             | 1                               | $(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0)$ |
|                  | ETS       | Average: 28,800                 | Average:                           | Average:                        | Average:                        |
|                  |           | 0.2139                          | 4                                  | 0.2154                          | 5                               |
|                  |           | 0.2103                          | 3                                  | 0.2203                          | 3                               |
|                  |           | 0.2255                          | 4                                  | 0.2255                          | 4                               |
|                  |           | 0.2097                          | 3                                  | 0.2097                          | 3                               |
|                  |           | 0.2089                          | 3                                  | 0.2089                          | 3                               |
|                  | API-BS    | 18,376                          | 0.2263                             | 15                              | $(0,0,0,0,0,0,2,0,0,1,0,0,0,1,9,2,0,0)$ |
|                  | Inv-VNS   | 15,518                          | 0.2253                             | 8                               | $(2,0,0,0,2,0,2,0,1,1,0,0,0,0,0,0,0,0,0)$ |
Table 5. Cont.

| Pattern       | Algorithm | Computation Time \( t_{sol} \) (s) | Obtained Throughput \( T_{sol} \) (jobs/s) | Designed Buffer Size \( N_{sol} \) | Obtained Buffer Configuration \( B_{sol} = (b_{1,2}, b_{3,5}, b_{4,6}, b_{7,8}, b_{9,10}, b_{11,12}, b_{13,13}, b_{14,15}, b_{15,16}, b_{16,22}, b_{17,18}, b_{18,19}, b_{19,20}, b_{21,22}, b_{23,24}, b_{25,26}, b_{27,28}, b_{30,31}, b_{33,34}) \) |
|---------------|-----------|-------------------------------------|----------------------------------------|----------------------------------|------------------------------------------------------------------------------------|
| \((0.2, 1.0, 0.5, 0.2)\) | API-VNS   | 23924                              | 0.2292                                 | 12                               | \((0,0,0,0,0,1,0,0,0,1,0,0,0,0,4,5,1,0,0)\)                                     |
|               | Inv-BS    | 745                                 | 0.2033                                 | 1                                | \((0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)\)                                    |
|               | ETS       | Average: 28,800                     |                                        |                                  |                                                                                     |
|               | API-BS    | 22,371                              | 0.2292                                 | 12                               | \((0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,4,5,1,0,0)\)                                     |
|               | Inv-VNS   | 21,884                              | 0.2248                                 | 10                               | \((0,0,0,0,2,0,0,6,1,1,0,0,0,0,0,0,0,0,0,0,0)\)                                    |
Figures 16 and 17 show the scatter plots of the designed buffer size versus the solution throughput for the five methods in the cases of $\lambda = 0.1$ and $\lambda = 0.2$, respectively. API-VNS and API-BS achieved the highest solution throughput with the same designed buffer size, followed by Inv-VNS and ETS. The solution throughput obtained by Inv-BS was the lowest. Both API-VNS and API-BS use the API as the BI to detect the bottleneck of a production line. For large-scale topology, the effectiveness of the API is so high that all real bottlenecks are detected by the initial neighborhood structure. Therefore, API-VNS and API-BS have a similar solution quality in the experiment. Because the effectiveness of the inventory indicator for the large-scale topology is low, it is difficult to detect the real bottleneck and increase the buffer size. Therefore, the solution throughput obtained by Inv-BS is the lowest, which shows the worst solution quality among all comparison methods. ETS adds several buffers in each buffer update and it tests more candidate buffer allocation solutions; therefore, it allocated buffers and increased the throughput very slowly in the experiment.

Figure 17. Computation time vs. throughput curves for the five methods in the case of $\lambda = 0.2$ in experiment 2.

Figure 18. Scatter plot of designed buffer size vs. solution throughput for the five methods in the case of $\lambda = 0.1$ in experiment 2.
Figure 19. Scatter plot of designed buffer size vs. solution throughput for the five methods in the case of $\lambda = 0.2$ in experiment 2.

Figure 20 shows the production line topology after buffers are allocated by API-VNS in the case of $\lambda = 0.1$. Most buffers were allocated before node 26. Because the external arrival rate is low, many jobs can arrive at the merging node 24 without considerable blocking. However, when the jobs leave node 24 and attempt to enter node 26, it is difficult to enter the node rapidly because the node requires some time to change job movement, and node 26 has a lower service rate than that of node 25. Therefore, node 26 easily becomes the bottleneck of the production line and requires many buffers allocated before it. Figure 21 shows the production line topology after buffers are allocated by API-VNS in the case of $\lambda = 0.2$. Most buffers were allocated before nodes 24 and 26. In this case, the external arrival rate increased, and it was difficult for the jobs to arrive at node 24 using only the branches with node 13. Jobs from splitting node 12 arrive at merging node 24 by the three branches with nodes 13, 16, and 19. To decrease the blocking at merging node 24, compared to the case of $\lambda = 0.1$, more buffers were allocated between node 23 and node 24 in the case of $\lambda = 0.2$.

Figure 20. Production line topology after buffers are allocated by API-VNS in the case of $\lambda = 0.1$ in experiment 2.
4.3. Discussion

In terms of solution quality, API-VNS can achieve the highest solution quality among five comparison methods in the large-scale experiment. In the small-scale experiment, API-VNS can achieve the highest solution quality in the case of $\lambda = 0.6$, while ETS achieves a higher solution quality than API-VNS in the case of $\lambda = 0.4$. This is because API-VNS can achieve a high solution quality with a short available computation time. If the computation time is extended, the solution quality of API-VNS may be degraded because API-VNS adds only one buffer in each buffer update. In the case of $\lambda = 0.4$, available computation time is very long for the production line with small-scale topology. Even though the buffer is allocated at the real bottleneck of the production line in each buffer update by API-VNS, it is still possible for the final solution to be a local optimal solution because a large number of buffers are allocated with a long available computation time. Solution quality of ETS is hardly affected even though the available computation time is very long, because it adds several buffers at a time and can thus consider more candidate buffer allocation solutions; however, it is usually difficult for ETS to provide a high solution quality in the available computation time. Inv-BS, API-BS, and Inv-VNS cannot achieve the same solution quality as API-VNS in some cases because of the disadvantages of using the inventory indicator and BS. Inventory is not effective enough to represent characteristics of the real bottleneck; meanwhile it is easy for BS to obtain a local optimal solution due to its limited search times.

As for computation time, API-VNS achieves the same throughput with the least computation time, which shows the advantages of its computation time. Because the API is more effective than inventory to represent characteristics of the real bottleneck, API-BS is found to be better than Inv-BS. Compared to BS that has limited search times, VNS can test more candidate bottlenecks in the case of no satisfactory solution generated, therefore, Inv-VNS is found to be better than Inv-BS. ETS usually requires a long computation time to obtain the same throughput because of its buffer updating regulation, which may test too many candidate buffer allocation solutions. Moreover, with the number of nodes increasing in a production line, computation time of the proposed approach also largely increases. Although it is difficult to estimate the production line scale that leads to unreasonable computation time, we believe that the proposed approach can be applicable to most of the production lines in industry.

In summary, API-VNS achieves a good balance between solution quality and computation time in available computation time, especially for the efficient buffer allocation of a production line with a large-scale topology.
5. Conclusions and Future Work

This study presented an efficient buffer allocation approach based on a bottleneck indicator and a variable neighborhood search algorithm to solve the buffer allocation problem in production lines. The active probability index as the bottleneck indicator was proposed to predict the most promising buffer updating locations. Based on the active probability index, the real bottleneck of the production line can be detected with a small number of searches, thereby decreasing computation time.

The variable neighborhood search algorithm was used to improve solution quality. If no satisfactory solution was generated for the bottleneck in the initial neighborhood structure, an additional neighborhood structure was used to search for a better solution. To prove the effectiveness of the proposed buffer allocation approach, it was compared with four other buffer allocation methods. The numerical examples showed that the proposed approach achieves a good balance between solution quality and computation time. In the large-scale experiment, the proposed approach achieves the highest solution quality and allocates buffers more rapidly than the other four methods. It requires less than 8 h even for the buffer allocation of the production line with 50 nodes.

In summary, the study contributes toward efficient buffer allocation for the rapid estimation of topology and the potential value of a production line. In particular, in the phase of feasibility analysis, the proposed approach can be used to rapidly estimate the handling capability and cost of a production line.

The proposed buffer allocation approach can only solve the efficient buffer allocation problem for a production line with one kind of job. In the future, the proposed approach should be improved to allocate buffers efficiently for a production line with multiple kinds of jobs.

Author Contributions: Conceptualization, S.G., T.H., T.K., K.T., J.I.U.R., and J.O.; methodology, S.G.; software, S.G.; validation, S.G., T.H., T.K., K.T. and J.O.; formal analysis, S.G.; investigation, S.G.; writing—original draft preparation, S.G.; writing—review and editing, S.G. and O.J.; supervision, O.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gunasekaran, A. Agile Manufacturing: A Framework for Research and Development. Int. J. Prod. Econ. 1999, 62, 87–105. [CrossRef]
2. Altıok, T.; Stidham, S. The Allocation of Interstage Buffer Capacities in Production Lines. IIE Trans. 1983, 15, 292–299. [CrossRef]
3. Jafari, M.A.; Shanthikumar, J.G. Determination of Optimal Buffer Storage Capacities and Optimal Allocation in Multistage Automatic Transfer Lines. IIE Trans. 1989, 21, 130–135. [CrossRef]
4. Tempelmeier, H. Practical Considerations in the Optimization of Flow Production Systems. Int. J. Prod. Res. 2003, 41, 149–170. [CrossRef]
5. Weiss, S.; Schwarz, J.A.; Stolletz, R. The Buffer Allocation Problem in Production Lines: Formulations, Solution Methods, and Instances. IISE Trans. 2019, 51, 456–485. [CrossRef]
6. Smith, J.M.; Cruz, F.R.B.; Van Woensel, T. Topological Network Design of General, Finite, Multi-Server Queueing Networks. Eur. J. Oper. Res. 2010, 201, 427–441. [CrossRef]
7. Huang, M.-G.; Chang, P.-L.; Chou, Y.-C. Buffer Allocation in Flow-Shop-Type Production Systems with General Arrival and Service Patterns. Comput. Oper. Res. 2002, 29, 103–121. [CrossRef]
8. Hillier, M.S. Characterizing the Optimal Allocation of Storage Space in Production Line Systems with Variable Processing Times. IIE Trans. 2000, 32, 1–8. [CrossRef]
9. Sennott, L.I. Stochastic Dynamic Programming and the Control of Queueing Systems; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2008.
10. Diamantidis, A.C.; Papadopoulos, C.T. A Dynamic Programming Algorithm for the Buffer Allocation Problem in Homogeneous Asymptotically Reliable Serial Production Lines. Math. Probl. Eng. 2004, 2004, 209–223. [CrossRef]
11. Sadr, J.; Malhame, R. Decomposition/Aggregation-Based Dynamic Programming Optimization of Partially Homogeneous Unreliable Transfer Lines. *IEEE Trans. Autom. Contr.* 2004, 49, 68–81. [CrossRef]

12. Kose, S.Y.; Kilinci, O. A Multi-Objective Hybrid Evolutionary Approach for Buffer Allocation in Open Serial Production Lines. *J. Intell. Manuf.* 2020, 31, 33–51. [CrossRef]

13. Costa, A.; Alfieri, A.; Matta, A.; Fichera, S. A Parallel Tabu Search for Solving the Primal Buffer Allocation Problem in Serial Production Systems. *Comput. Oper. Res.* 2015, 64, 97–112. [CrossRef]

14. Beşikçi, E.B.; Arslan, O.; Turan, O.; Öլçer, A.I. An Artificial Neural Network Based Decision Support System for Energy Efficient Ship Operations. *Comput. Oper. Res.* 2016, 66, 393–401. [CrossRef]

15. Lin, J.T.; Chiu, C.-C. A Hybrid Particle Swarm Optimization with Local Search for Stochastic Resource Allocation Problem. *J. Intell. Manuf.* 2018, 29, 481–495. [CrossRef]

16. Lin, J.T.; Chiu, C.-C. A Segmentation Approach for Solving Buffer Allocation Problems in Large Production Systems. *Int. J. Prod. Res.* 2016, 54, 6121–6141. [CrossRef]

17. Li, J. Continuous Improvement at Toyota Manufacturing Plant: Applications of Production Systems Engineering Methods. *Int. J. Prod. Res.* 2013, 51, 7235–7249. [CrossRef]

18. Vergara, H.A.; Kim, D.S. A New Method for the Placement of Buffers in Serial Production Lines. *Int. J. Prod. Res.* 2009, 47, 4437–4456. [CrossRef]

19. Seo, D.-W.; Lee, H. Stationary Waiting Times in m-Node Tandem Queues with Production Blocking. *IEEE Trans. Autom. Control* 2011, 56, 958–961. [CrossRef]

20. MacGregor Smith, J. Joint Optimisation of Buffers and Network Population for Closed Finite Queueing Systems. *Int. J. Prod. Res.* 2016, 54, 1–25. [CrossRef]

21. Balsamo, S. Queueing Networks with Blocking: Analysis, Solution Algorithms and Properties. In Network Performance Engineering. Lecture Notes in Computer Science; Kouvatsos, D.D., Ed.; Springer: Berlin, Germany, 2011.

22. Gershwin, S.B.; Schor, J.E. Efficient Algorithms for Buffer Space Allocation. *Ann. Oper. Res.* 2000, 93, 117–144. [CrossRef]

23. Gao, S.; Rubrico, J.I.U.; Higashi, T.; Kobayashi, T.; Taneda, K.; Ota, J. Efficient Throughput Analysis of Production Lines Based on Modular Queues. *IEEE Access* 2019, 7, 95314–95326. [CrossRef]

24. Park, T. A Two-Phase Heuristic Algorithm for Determining Buffer Sizes of Production Lines. *Int. J. Prod. Res.* 1993, 31, 613–631. [CrossRef]

25. Roser, C.; Nakano, M.; Tanaka, M. A practical bottleneck detection method. In Proceedings of the Winter Simulation Conference Proceedings, Arlington, VA, USA, 9–12 December 2001. [CrossRef]

26. Demir, L.; Tunali, S.; Eliiyi, D.T. An adaptive tabu search approach for buffer allocation problem in unreliable non-homogenous production lines. *Comput. Oper. Res.* 2012, 39, 1477–1486. [CrossRef]

27. Suri, R. An Overview of Evaluative Models for Flexible Manufacturing Systems. *Ann. Oper. Res.* 1985, 3, 13–21. [CrossRef]

28. Lutz, C.M.; Davis, K.R.; Sun, M. Determining Buffer Location and Size in Production Lines Using Tabu Search. *Eur. J. Oper. Res.* 1998, 106, 301–316. [CrossRef]

29. Papadopoulos, C.T.; O’Kelly, M.E.J.; Tsadiras, A.K. A DSS for the Buffer Allocation of Production Lines Based on a Comparative Evaluation of a Set of Search Algorithms. *Int. J. Prod. Res.* 2013, 51, 4175–4199. [CrossRef]

Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).