

Article

A Recursive Optimization Approach for Buffer Allocation in Large Production Lines

Lei Li *, Menggao Dong and Jian Zhou

National Key Laboratory of Equipment State Sensing and Smart Support, National University of Defense Technology, Changsha 410073, China; dmgl78@126.com (M.D.); zhoujian178@126.com (J.Z.)

* Correspondence: li_lei@nudt.edu.cn; Tel.: +86-731-87005511

Abstract: Buffer allocation is a critical issue in the design stage of manufacturing systems, as buffer capacities may have a great impact on system performance. In this paper, we consider the problem of minimizing the total buffer capacity of a flow line to achieve a desired production rate. A recursive optimization approach is proposed to solve the problem in large production lines. Instead of optimizing a long line directly, the proposed approach decomposes it into two sub-lines, optimizes them recursively, and combines their solutions to find the optimal buffer distribution of the original line. Two different recursive algorithms are developed and their performance is demonstrated by comparing them with a gradient search algorithm. The numerical results show that the recursive algorithms are almost as accurate as the gradient algorithm, but much more efficient, especially for large production lines.

Keywords: manufacturing flow line; buffer allocation problem; design optimization; recursive algorithm



Citation: Li, L.; Dong, M.; Zhou, J. A Recursive Optimization Approach for Buffer Allocation in Large Production Line. *Appl. Sci.* **2024**, *14*, 9953. <https://doi.org/10.3390/app14219953>

Academic Editor: Luis Javier Garcia Villalba

Received: 28 September 2024

Revised: 25 October 2024

Accepted: 28 October 2024

Published: 31 October 2024



Copyright: © 2024 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 (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

A production line is composed of a series of machines, with buffers separating each pair of consecutive machines. Material enters the system through the first machine, flows through the intermediate buffers and machines, and finally, exits the system through the last machine. The material flow may be disrupted by several random factors such as machine failures and repairs. When a machine fails, its downstream buffer may become empty, causing the downstream machine to be starved; similarly, the upstream buffer may become full, causing the upstream machine to be blocked. These starvation and blockage phenomena lead to a reduction in throughput (production rate).

The throughput of a production line can be improved by increasing buffer capacities to reduce material flow disruptions. However, buffers require additional capital investment and floor space, which may be costly. Moreover, buffers increase the level of work-in-progress (WIP) inventory, resulting in longer lead times. If the buffers of a production line are too large, the capital investment and WIP inventory will exceed the benefit of increased throughput. Thus, buffer allocation is a critical issue in production line design.

The buffer allocation problem (BAP) has been one of the central themes in manufacturing systems research for the past 30 years [1]. According to Weiss et al. [2], the BAP is classified into three categories: (1) the primal problem aims to minimize the total buffer capacity to achieve a desired throughput; (2) the dual problem attempts to maximize the throughput with a fixed amount of buffer capacity; (3) the other variants of BAP additionally consider the WIP, the cycle time, the waiting time of customer orders, or their combinations in the objective function. In this study, we focus on the primal problem, which is called BAP-2 in Demir et al. [3] or the lean buffering problem in Li and Meerkov [4].

It is well known that the BAP is an NP-hard combinatorial optimization problem [5,6]. The computation time for solving a BAP increases significantly with the length of production line. For example, Shi and Gershwin [7] describe a 30-machine line which takes

about 50 min to optimize the buffer allocation. However, long computation times are unacceptable in the planning process of production lines. The designers may consider various design options, each with its unique set of machine parameters, and the BAP has to be solved for each one. Therefore, it is desirable to develop efficient methods to solve large-scale BAPs in a short amount of time.

In this paper, we propose a novel approach to speed up the solution process of BAPs. The key idea is to decompose a long production line into two sub-lines, optimize them recursively, and combine their solutions to find the optimal buffer allocation of the original line. Two recursive algorithms are developed. Their performance is compared to that of a gradient search algorithm. Numerical results show that they achieve the same level of accuracy as the gradient algorithm, but the computation time is significantly reduced, especially in the case of long production lines.

The rest of the paper is organized as follows. Section 2 presents a brief review of BAPs. Section 3 formulates the problem. Section 4 describes the performance evaluation method of production lines. Section 5 presents the recursive algorithms for BAPs. Numerical results are given in Section 6. Finally, Section 7 concludes the paper.

2. Literature Review

In the literature, there is a substantial amount of research works on BAPs. For a comprehensive review, the reader is referred to Weiss et al. [2], Demir et al. [3], Gershwin and Schor [8], Hudson et al. [9], and Papadopoulos and Heavey [10].

According to Weiss et al. [2], the solution methods to BAPs can be classified into explicit solutions, integrated optimization methods, and iterative optimization methods. The three categories differ in how they incorporate the performance evaluation of a given buffer allocation. The iterative optimization method is the most widely used method; it consists of an evaluative method and a generative method. The two methods are combined in a closed-loop configuration: the evaluative method calculates the performance measure of a buffer allocation candidate, the generative method uses the evaluation result to select a new candidate, and this candidate is further evaluated. In the following, we review the most relevant works on the evaluative and generative methods.

Typical evaluative methods include simulation, exact analytical methods, and approximate analytical methods. Comprehensive surveys of performance evaluation techniques were provided by Dallery and Gershwin [11], Papadopoulos and Heavey [10], Li et al. [12], Papadopoulos, Li, and O'Kelly [1], among others. Simulation is a flexible tool for evaluating complex realistic manufacturing systems [13], but it is very time consuming, especially when many alternative system configurations have to be evaluated [14,15]. Exact analytical methods are based on Markovian analysis and are only applicable to small systems due to the state-space explosion problem. Therefore, considerable research effort was devoted to approximate analytical methods, such as decomposition methods [16,17] and aggregation methods [4,18]. Our work is based on decomposition methods.

Gershwin [16] proposed a well-known decomposition method for the approximate analysis of synchronous production lines with geometrically distributed machine failure and repair times. The core idea of this method was to decompose a long production line into a set of equivalent two-machine systems with unknown parameters. A set of decomposition equations were derived to determine the unknown parameters and the performance measures of the original line was obtained by evaluating the two-machine systems. Dallery, David, and Xie [17] proposed a similar method for the analysis of continuous flow lines with exponentially distributed failure and repair times. Moreover, they developed an efficient computational algorithm (referred to as the DDX algorithm) to solve the decomposition equations. Burman [19] extended the decomposition method for evaluating inhomogeneous continuous flow lines and developed a more efficient algorithm. In general, the above decomposition methods provide accurate performance estimates. However, the accuracy of these methods may be poor when the machine reliability parameters have different orders of magnitude. To solve this problem, Dallery and Le

Bihan [20] proposed an improved method by using generalized exponential distributions instead of exponential distributions to model the repair times of the virtual machines in the decomposed systems. Le Bihan and Dallery [21] developed a more accurate method by using hyper-exponential distributions to model the repair times. Other extensions of the decomposition methods include Levantesi et al. [22], Bierbooms et al. [23] and Colledani and Gershwin [24], among others.

As far as the generative methods are concerned, the existing methods can be roughly classified into the enumeration method, metaheuristics, dynamic programming, and search algorithms [2]. The enumeration method can guarantee the optimality of allocation results, but is only applicable to small systems [25]. In the literature, it is typically used as a benchmark for other generative methods [26].

Metaheuristics are high-level problem-independent algorithmic frameworks that provide a set of guidelines to develop heuristic optimization algorithms. Metaheuristics algorithms applied to BAPs include simulated annealing [27], genetic algorithms [28], tabu search [29], ant colony optimization [30], artificial immune [31], and big bang–big crunch [32], among others. Some researchers proposed hybrid algorithms that combine multiple metaheuristics algorithms [33], or combine metaheuristics with other search algorithms [34]. The main disadvantage of metaheuristics is that they do not make use of problem-specific properties [2,3].

Dynamic programming methods [35–37] decompose the BAP into sub-problems, which are solved recursively along the forward or backward path of a production line. The existing dynamic programming methods have to be used in conjunction with specific performance evaluation methods. As a result, their applications are limited.

Search methods are based on mathematical principles or heuristic rules to find optimal or near-optimal buffer allocations. Unlike metaheuristics, search methods typically exploit some specific properties of BAPs. For example, several gradient search algorithms assume the production rate function is continuous, monotonic, and concave [7,8,38]. It can be expected that the gradient algorithms that exploit these properties are more efficient than the metaheuristics algorithms that do not. However, gradient algorithms may lead to non-integer solutions, and thus, some rounding techniques have to be employed. Other search methods include Powell's method [39], the Hooke and Jeeves method [40], rule-based local search algorithms [41], and heuristic algorithms [42,43].

Artificial intelligence is one of the digital technologies emerging from Industry 4.0 [44] and shows great potential in manufacturing system optimization [45]. Can and Heavey [46] obtain training datasets of buffer allocations for neural networks by a genetic algorithm. Tsadiras et al. [47] propose an artificial neural network-based decision support system for solving the buffer allocation problem in reliable production lines.

Recently, several researchers have proposed decomposition-based search methods for solving BAPs in large production lines. Shi and Gershwin [48] developed a segmentation approach for BAPs, which divides a production line into several overlapping sub-lines. These sub-lines are optimized separately under the constraint of the original line. The optimal solutions of the sub-lines are combined to obtain a near optimal solution of the original line. Li et al. [49] proposed employing three-machine, two-buffer systems as building blocks for optimizing long production lines. The original BAP is solved by decomposing a long line into a group of overlapping three-machine, two-buffer systems and solving the simple BAPs in the decomposed subsystems. Xi et al. [25] divided a large production line into multiple disconnected subsystems using conditional variables. The subsystems are optimized independently under additional conditions, and their solutions are integrated to achieve a near-optimal solution for the original line.

This paper proposes two novel decomposition-based search algorithms for BAPs. Unlike the existing methods, we decompose a production line into sub-lines in a recursive manner, and the original BAP is solved by recursively solving the BAPs in the sub-lines. Numerical experiments show the recursive algorithms are very efficient for optimizing long production lines.

3. Problem Formulation

3.1. Production Line Model

A serial production line consists of K machines ($M_1, \dots, M_K$) separated by $K - 1$ buffers ($B_1, \dots, B_{K-1}$), as depicted in Figure 1. We consider the classic continuous flow model of production lines [17]:

1. The material is modelled as a continuous fluid.
2. Each machine is unreliable and can be in either the up or down state. The processing rate of machine M_i is s_i , i.e., if M_i operates in isolation, when it is up, a quantity of material $s_i dt$ is processed in an infinitesimal time dt . If all machines have the same processing rate, the line is said to be homogeneous. In the case of homogeneous lines, time is scaled and the processing rate is 1.
3. Buffer B_i has a capacity of $b_i \geq 0$, $i = 1, \dots, K - 1$, meaning that it can accommodate b_i units of material.
4. Machine M_i , $i = 2, \dots, K$, is starved at time t if one of its upstream machines is down and all buffers between this machine and M_i are empty. The first machine M_1 is never starved.
5. Machine M_i , $i = 1, \dots, K - 1$, is blocked at time t if one of its downstream machines is down and all buffers between this machine and M_i are full. The last machine M_K is never blocked.
6. The machines can fail only when they are working, i.e., they are up and neither starved nor blocked.
7. The time to failure of machine M_i , $i = 1, \dots, K$, is exponentially distributed, with rate λ_i . The mean time to failure (MTTF) is $1/\lambda_i$.
8. The time to repair of machine M_i , $i = 1, \dots, K$, is exponentially distributed, with rate μ_i . The mean time to repair (MTTR) is $1/\mu_i$.

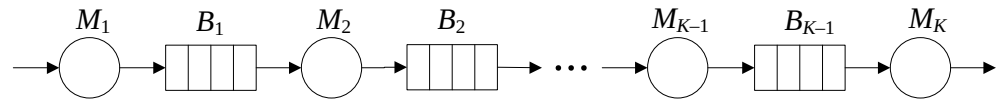


Figure 1. Serial production line.

3.2. Buffer Allocation Problem

We consider the primal BAP (Gershwin and Schor [8]), which seeks a buffer allocation $b = (b_1, \dots, b_{K-1})$ that minimizes the total buffer capacity $N = \sum_{i=1}^{K-1} b_i$ while a desired production rate P^* is achieved. Formally,

$$\min_b N = \sum_{i=1}^{K-1} b_i \quad (1)$$

subject to

$$P(b_1, \dots, b_{K-1}) \geq P^*, \quad (2)$$

$$b_i \text{ non-negative integers, } i = 1, \dots, K - 1, \quad (3)$$

where $P(b_1, \dots, b_{K-1})$ denotes the production rate of the line as a function of buffer capacities $b_1, \dots, b_{K-1}$. The production rate is defined as the average amount of material produced by the last machine M_K per time unit in the steady state.

In the design stage of a production line, the designers specify the machine parameters and a desired production rate. The buffer capacity vector $b = (b_1, \dots, b_{K-1})$ is the decision variable. Then, the BAP is well-defined. To solve the problem, two tools are needed. The first tool is a performance evaluation method that calculates the production rate of a given buffer vector. The second tool is an optimization method that generate an optimal or near-optimal solution for the BAP. The two tools are discussed in the following two sections.

4. Performance Evaluation Method

Gershwin [16] proposed a well-known decomposition method for approximate performance evaluation of homogeneous production lines. This method was widely used in the study of BAPs [8,32,48]. As illustrated in Figure 2, the idea of decomposition is to decompose a K -machine production line L into a set of two-machine, one-buffer subsystems $L(i)$, $i = 1, \dots, K - 1$. The virtual machine $M_u(i)$ of $L(i)$ represents the portion of line L upstream of buffer B_i , i.e., machine M_1 to machine M_i . Similarly, the virtual machine $M_d(i)$ represents the portion of line L downstream of buffer B_i , i.e., machine M_{i+1} to machine M_K . Moreover, the buffer $B(i)$ of $L(i)$ has the same capacity, N_i , as the original buffer B_i . The decomposition principle is to determine the parameters of virtual machines such that the behavior of material flow through buffer $B(i)$ is as close to that of the flow through buffer B_i as possible. Dallery, David, and Xie [17] established the decomposition equations of virtual machine parameters for the continuous flow production line model described in the previous section. They also proposed an efficient algorithm, called the DDX algorithm, to solve the equations.

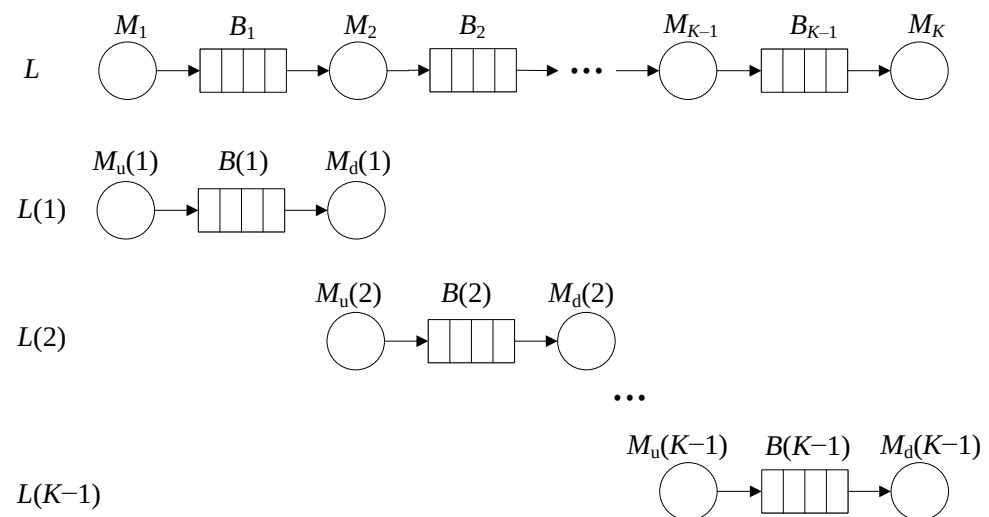


Figure 2. Decomposition method for approximate performance evaluation.

We do not use Gershwin's decomposition method to evaluate the production rate of homogeneous lines. The reason is that the accuracy of Gershwin's decomposition method may be unacceptable when the reliability parameters of different machines have different orders of magnitude [21]. This may lead to incorrect results when comparing the performance of different buffer configurations. For example, consider the six-machine line described in Table 1. The production rate of the line under several different buffer configurations was evaluated by the DDX algorithm and the results are presented in Table 2. Note that the buffer configurations C1–C5 are obtained from C0 by increasing the buffer capacity b_i by 1, $i = 1, \dots, 5$, respectively. It can be observed that for each buffer B_i , while its capacity increases, the production rate obtained by the DDX algorithm decreases. This phenomenon is very strange and should not occur, since the production rate is a monotonic function with respect to buffer capacities [50]. Table 2 also presents the production rate obtained by simulation. To avoid incorrect comparisons due to simulation errors, the random variable sequences in the simulations of different buffer configurations were set to be the same, i.e., the sequences of times to failure and the sequences of times to repair of each machine were the same for all the simulations. These simulations were terminated when each machine had encountered at least 10,000 repair events. As can be seen, the simulated production rates satisfy monotonicity.

To avoid the above problem, we use the robust decomposition method [21] to evaluate homogeneous production lines. Unlike the decomposition method in [17], which models

the repair times of virtual machines by exponential distributions, the robust decomposition method uses two-stage hyper-exponential distributions to model the repair times. The hyper-exponential distributions allow the use of a three-moment approximation instead of a single-moment approximation of the repair times. In this way, the accuracy is greatly improved. As can be seen from Table 2, the production rates evaluated by the robust decomposition method are much closer to the simulated production rates than the DDX algorithm. More importantly, the robust decomposition method leads to the same results as the simulation method when comparing the performance of different buffer configurations. For example, both methods show that buffer configuration C5 is the best one among C0–C5, whereas the DDX method shows that C0 is.

Table 1. A six-machine production line.

i	1	2	3	4	5	6
λ_i	0.122	0.005	0.117	0.058	0.132	0.050
μ_i	0.393	0.013	0.484	0.436	0.354	0.326

Table 2. Production rate of the six-machine line under different buffer configurations using different evaluation methods.

Case	Buffer Capacities	Production Rate		
		DDX	Robust	Simulation
C0	(21, 19, 11, 9, 4)	0.6601	0.5968	0.5881
C1	(22, 19, 11, 9, 4)	0.6600	0.5969	0.5883
C2	(21, 20, 11, 9, 4)	0.6590	0.5978	0.5888
C3	(21, 19, 12, 9, 4)	0.6599	0.5979	0.5889
C4	(21, 19, 11, 10, 4)	0.6599	0.5983	0.5891
C5	(21, 19, 11, 9, 5)	0.6600	0.5994	0.5903

The robust decomposition method is only applicable to homogeneous production lines. For inhomogeneous lines, we use the ADDX algorithm in [19] to evaluate the production rate. It is an extended DDX algorithm and is widely used in the study of buffer allocation problems [8,48,51].

5. Solution Approach

The core idea of the proposed approach is to decompose a long production line into two short sub-lines, optimize them recursively, and combine their solutions to construct the solution to the BAP in the original line. In the following, we first introduce a gradient search algorithm for BAPs, and then, extend it to two recursive algorithms to improve computational efficiency.

5.1. Gradient Search Algorithm

Consider the production rate function $P(b_1, \dots, b_{K-1})$. Using the robust decomposition method, its gradient can be evaluated numerically as follows:

$$g_i = \frac{P(b_1, \dots, b_i + \Delta b, \dots, b_{K-1}) - P(b_1, \dots, b_i, \dots, b_{K-1})}{\Delta b}, \quad (4)$$

where Δb is a small positive number and $i = 1, \dots, K - 1$. To solve the BAP, a simple algorithm could start with an empty buffer configuration and move in the gradient direction until the desired production rate is reached. However, this algorithm may lead to non-integer solutions, and thus, some rounding techniques would have to be applied [8,51].

To avoid becoming involved in the analysis of various rounding techniques, we use Algorithm 1, which always yields integer solutions. The algorithm starts with an empty buffer configuration and searches in the direction determined by the maximum

gradient component $g_j = \max \{g_1, \dots, g_{K-1}\}$. Then, the buffer capacity b_j is increased by one buffer unit. The above procedure is repeated until the desired production rate is achieved. The numerical experiments in Section 6 show the algorithm works well for small production lines. It always achieves the optimal solutions in a short time. However, for large production lines, the gradient algorithm is time consuming. It is not difficult to see that the computational complexity of the algorithm is $O(NK)$, where N is the total buffer capacity of the optimal solution. For a long production line (i.e., K is large), the total capacity N is typically large as well. As a result, the computational complexity is high.

Algorithm 1 Gradient search algorithm

Step 1: Initialization

set the initial solution $(b_1, \dots, b_{K-1})$ to $(0, \dots, 0)$.

Step 2:

if $P(b_1, \dots, b_{K-1}) \geq P^*$, return $(b_1, \dots, b_{K-1})$ as the best solution.

Step 3:

calculate the gradient $(g_1, \dots, g_{K-1})$ according to Equation (4), where $\Delta b = 1$.

Step 4:

find the index j such that $g_j = \max \{g_1, \dots, g_{K-1}\}$.

$b_j := b_j + 1$.

go to Step 2.

5.2. Recursive Binary Decomposition Algorithm

One way to improve the efficiency of the gradient search algorithm is to start with a good initial solution. For this purpose, we propose a recursive binary decomposition (RBD) algorithm.

Consider the production line L in Figure 3, which consists of machines $M_1, \dots, M_K$ and buffers $B_1, \dots, B_{K-1}$. The middle machine of L is M_m , where $m = \lceil (K+1)/2 \rceil$, i.e., m is the smallest integer greater than or equal to $(K+1)/2$. The line L is decomposed into a left sub-line L_1 , which consists of machines $M_1, \dots, M_m$ and buffers $B_1, \dots, B_{m-1}$, and a right sub-line L_2 , which consists of machines $M_m, \dots, M_K$ and buffers $B_m, \dots, B_{K-1}$. The two sub-lines operate independently. Moreover, each sub-line is recursively decomposed until the decomposed sub-lines have only two machines.

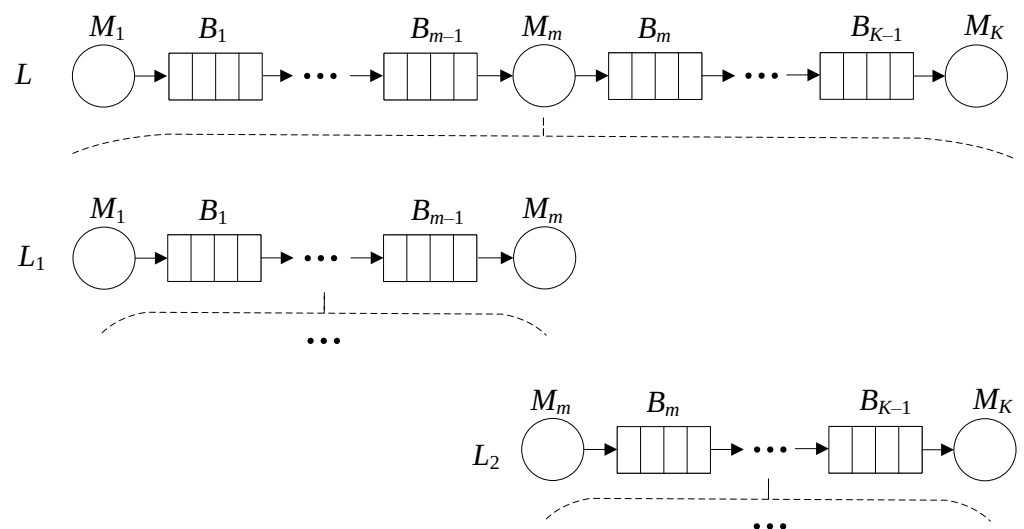


Figure 3. Recursive binary decomposition of a production line.

Consider the BAP in L with a desired production rate P^* , represented by $\text{BAP}(M_1, B_1, \dots, M_K, P^*)$. Let $b = (b_1^*, \dots, b_{K-1}^*)$ be the optimal solution to this problem. Then, $(b_1^*, \dots, b_{m-1}^*)$ can guarantee the sub-line L_1 achieves the desired production rate P^* , because the last

machine M_m in L_1 is never blocked, meaning that $P_{L_1}(b_1^*, \dots, b_{m-1}^*) > P_L(b_1^*, \dots, b_{K-1}^*) \geq P^*$ (see [52] for a strict proof). Let $x = (x_1, \dots, x_{m-1})$ be the optimal solution to the sub-problem $\text{BAP}(M_1, B_1, \dots, M_m, P^*)$ in L_1 . It follows that $\sum_{i=1}^{m-1} x_i \leq \sum_{i=1}^{m-1} b_i^*$, i.e., the sum of the buffer capacities x_i in L_1 provides a lower bound for the sum of the respective buffer capacities b_i^* in the original line L . In the RBD algorithm, we use x_i as a lower bound of b_i^* , $i = 1, \dots, m-1$ to speed up the search process. Similarly, let $y = (y_m, \dots, y_{K-1})$ be the optimal solution to the sub-problem $\text{BAP}(M_m, B_m, \dots, M_K, P^*)$ in L_2 , and y_i is used as a lower bound of the respective buffer capacity b_i^* in the original line L , $i = m, \dots, K-1$. The main steps of the RBD algorithm are exhibited as a flowchart in Figure 4 and the pseudocode is given in Algorithm 2.

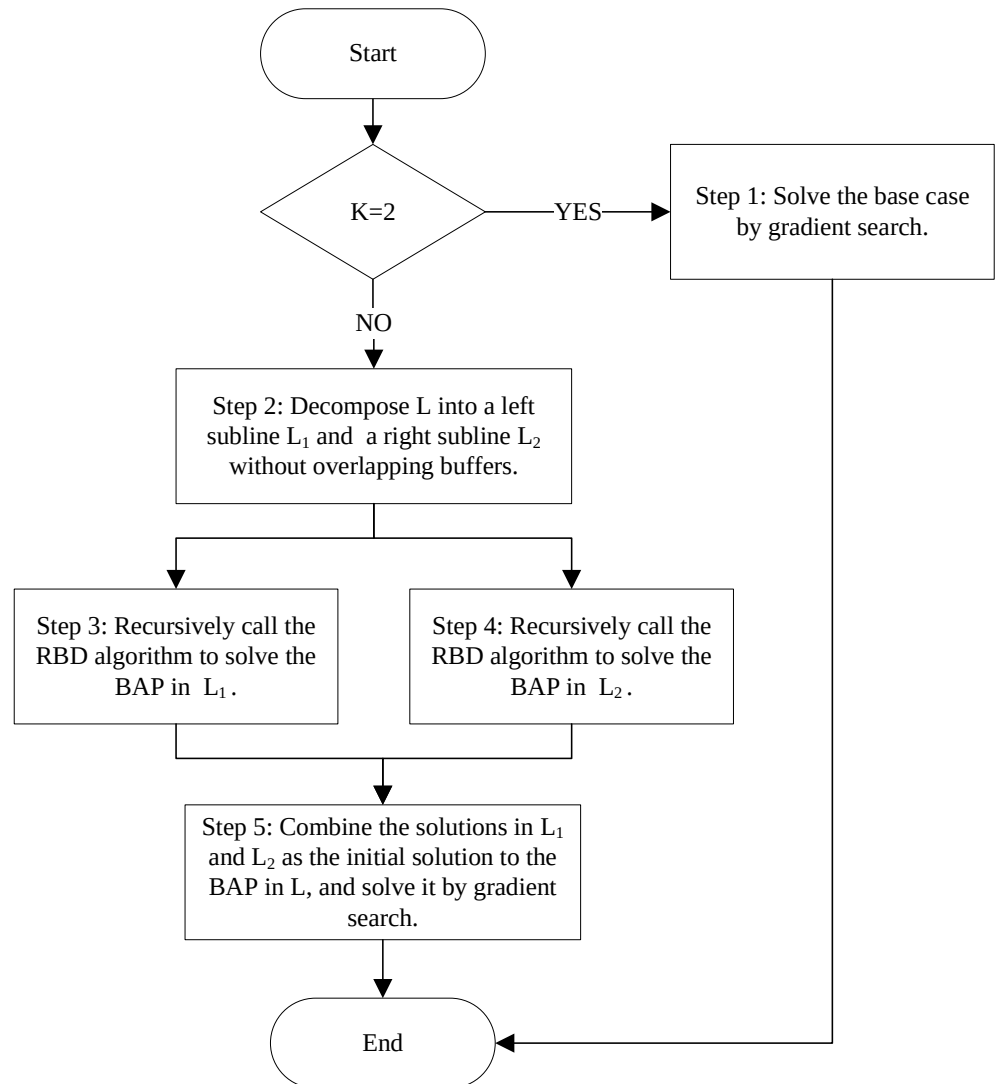


Figure 4. Flowchart for the RBD algorithm.

Note that the sub-lines L_1 and L_2 may have multiple optimal solutions. For example, consider a five-machine, four-buffer line with identical machines. The left sub-line $L_1 = (M_1, B_1, M_2, B_2, M_3)$ may have two optimal solutions $(a, a+1)$ and $(a+1, a)$, because when the current solution of the gradient algorithm is (a, a) , the components g_1 and g_2 are the same due to symmetry. The right sub-line L_2 may also have the ambiguity problem. In our implementation, the following rule is used: if L_1 has more than one maximum gradient component, the buffer on the right is prioritized higher; if L_2 has more than one maximum gradient component, the buffer on the left is prioritized higher. According to

this rule, the initial solution to the original BAP is $(a, a + 1, a + 1, a)$, which is consistent with the inverted bowl phenomenon [53].

Algorithm 2 Recursive binary decomposition algorithm (RBD)

Step 1: Solve the base case

If $K = 2$

call the gradient search algorithm to solve $BAP(M_1, B_1, M_2, P^*)$.

return the obtained solution b^* .

Step 2: Decompose the line L into a left sub-line L_1 and a right sub-line L_2

$m := \lceil (K + 1)/2 \rceil$

$L_1 := (M_1, B_1, \dots, M_m)$

$L_2 := (M_m, B_m, \dots, M_K)$

Step 3: Solve the BAP in the left sub-line L_1

call the RBD algorithm to recursively solve $BAP(M_1, B_1, \dots, M_m, P^*)$.

let $x = (x_1, \dots, x_{m-1})$ be the obtained solution.

Step 4: Solve the BAP in the right sub-line L_2

call the RBD algorithm to recursively solve $BAP(M_m, B_m, \dots, M_K, P^*)$.

let $y = (y_m, \dots, y_{K-1})$ be the obtained solution.

Step 5: Solve the BAP in the original line L

$b := (x_1, \dots, x_{m-1}, y_m, \dots, y_{K-1})$

call the gradient search algorithm to solve $BAP(M_1, B_1, \dots, M_K, P^*)$ with the initial solution b .

return the obtained solution b^* .

Table 3 presents an example to further illustrate how the RBD algorithm works. Consider a five-machine line L with $s_i = 1$, $MTTF_i = 90$ and $MTTR_i = 10$, $i = 1, \dots, 5$. The desired production rate P^* is 0.85. The RBD algorithm decomposes the line L into two sub-lines L_1 and L_2 . Each sub-line is recursively solved by the RBD algorithm. In the recursive execution, L_1 is further decomposed into $L_{1.1}$ and $L_{1.2}$. Since the length of both $L_{1.1}$ and $L_{1.2}$ is 2, they belong to the base case and are solved by gradient search. Then, the solutions of $L_{1.1}$ and $L_{1.2}$ are combined as the initial solution (13, 13) for L_1 and the gradient algorithm generates the optimal solution of L_1 , i.e., (30, 31). The BAP in L_2 is solved similarly and the solution is (31, 30). The optimal solutions of L_1 and L_2 are combined to provide the initial solution (30, 31, 31, 30) for the gradient algorithm to solve the original BAP in L , and the final solution (37, 50, 51, 37) is obtained.

Table 3. Solution process of the RBD algorithm for a five-machine line with $MTTF_i = 90$, $MTTR_i = 10$, $i = 1, \dots, 5$, and $P^* = 0.85$.

Label	Line/ Sub-Lines	Left Sub-Line	Right Sub-Line	Initial Solution b	Optimal Solution b^*	$P(b^*)$
L	$(M_1, B_1, \dots, M_5)$	L_1	L_2	(30, 31, 31, 30)	(37, 50, 51, 37)	0.850054
L_1	$(M_1, B_1, \dots, M_3)$	$L_{1.1}$	$L_{1.2}$	(13, 13)	(30, 31)	0.850417
$L_{1.1}$	(M_1, B_1, M_2)			(0)	(13)	0.850610
$L_{1.2}$	(M_2, B_2, M_3)			(0)	(13)	0.850610
L_2	$(M_3, B_3, \dots, M_5)$	$L_{2.1}$	$L_{2.2}$	(13, 13)	(31, 30)	0.850417
$L_{2.1}$	(M_3, B_3, M_4)			(0)	(13)	0.850610
$L_{2.2}$	(M_4, B_4, M_5)			(0)	(13)	0.850610

5.3. Recursive Overlapping Decomposition Algorithm

In the previous algorithm, a production line is divided into two sub-lines without overlapping buffers, i.e., each buffer of the sub-lines corresponds to a different buffer in the original line. In this subsection, we propose a recursive overlapping decomposition (ROD) algorithm, which allows the sub-lines have overlapping buffers, i.e., the buffers of the sub-lines may correspond to the same buffer in the original line.

Consider the production line L in Figure 5, which consists of machines $M_1, \dots, M_K$ and buffers $B_1, \dots, B_{K-1}$. Let $s = \lfloor K/3 \rfloor$ be the largest integer less than or equal to $K/3$. The line L is decomposed into a left sub-line L_1 , which consists of machines $M_1, \dots, M_{K-s}$ and buffers $B_1, \dots, B_{K-s-1}$, and a right sub-line L_2 , which consists of machines $M_{s+1}, \dots, M_K$ and buffers $B_{s+1}, \dots, B_{K-1}$. Clearly, when $K > 3$, the two sub-lines have overlapping buffers $B_{s+1}, \dots, B_{K-s-1}$. As in the previous case, the sub-lines L_1 and L_2 operate independently and they are recursively decomposed until the decomposed sub-lines have only three machines. Figure 6 shows the flowchart of the ROD algorithm and the pseudocode is given in Algorithm 3.

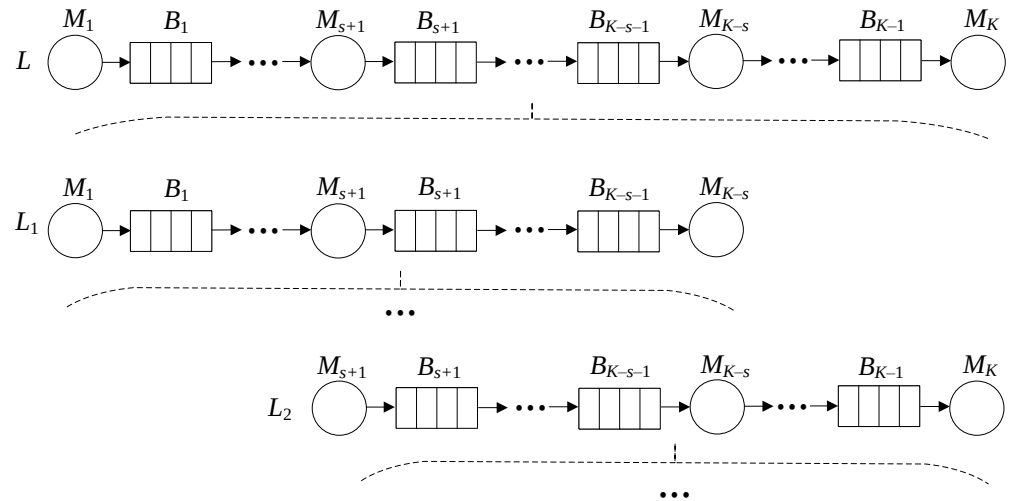


Figure 5. Recursive overlapping decomposition of a production line.

Algorithm 3 Recursive overlapping decomposition algorithm (ROD)

Step 1: Solve the base case

If $K \leq 3$

call the gradient search algorithm to solve $\text{BAP}(M_1, B_1, \dots, M_K, P^*)$.
return the obtained solution b^* .

Step 2: Decompose the line L into a left sub-line L_1 and a right sub-line L_2

$s := \lfloor K/3 \rfloor$

$L_1 := (M_1, B_1, \dots, M_{K-s})$

$L_2 := (M_{s+1}, B_{s+1}, \dots, M_K)$

Step 3: Solve the BAP in the left sub-line L_1

call the ROD algorithm to recursively solve $\text{BAP}(M_1, B_1, \dots, M_{K-s}, P^*)$.
let $x = (x_1, \dots, x_{K-s-1})$ be the obtained solution.

Step 4: Solve the BAP in the right sub-line L_2

call the ROD algorithm to recursively solve $\text{BAP}(M_{s+1}, B_{s+1}, \dots, M_K, P^*)$.
let $y = (y_{s+1}, \dots, y_{K-1})$ be the obtained solution.

Step 5: Solve the BAP in the original line L

$b := (x_1, \dots, x_s, \max(x_{s+1}, y_{s+1}), \dots, \max(x_{K-s-1}, y_{K-s-1}), y_{K-s}, \dots, y_{K-1})$

call the gradient search algorithm to solve $\text{BAP}(M_1, B_1, \dots, M_K, P^*)$ with the initial solution b .

return the obtained solution b^* .

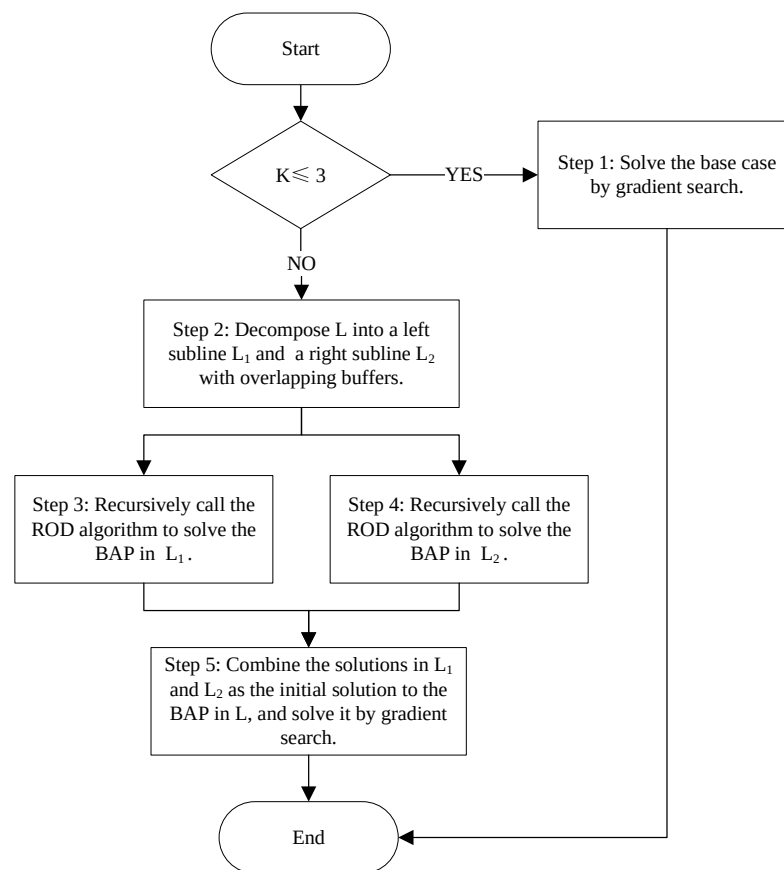


Figure 6. Flowchart for the ROD algorithm.

Table 4 shows how the ROD algorithm solves the BAP described in the previous subsection. The line L is decomposed into two sub-lines L_1 and L_2 , which are further decomposed into sub-lines $L_{1,1}$, $L_{1,2}$, $L_{2,1}$, and $L_{2,2}$. These three-machine, two-buffer systems belong to the base case and they have the same solution (30, 31). Then, the solutions of $L_{1,1}$ and $L_{1,2}$ are combined as the initial solution (30, 31, 31) for the gradient algorithm to obtain the optimal solution of L_1 , i.e., (35, 46, 36). The BAP in L_2 is solved similarly. The optimal solutions of L_1 and L_2 are combined to provide the initial solution (35, 46, 46, 36) for the gradient algorithm to solve the original BAP in L . As can be seen, the initial solution is quite close to the final solution (37, 50, 51, 37), and thus, it can speed up the search process of the gradient algorithm.

Table 4. Solution process of the ROD algorithm for a five-machine line with $MTTF_i = 90$, $MTTR_i = 10$, $i = 1, \dots, 5$, and $P^* = 0.85$.

Label	Machines and Buffers	Left Sub-Line	Right Sub-Line	Initial Solution b	Optimal Solution b^*	$P(b^*)$
L	$(M_1, B_1, \dots, M_5)$	L_1	L_2	(35, 46, 46, 36)	(37, 50, 51, 37)	0.850054
L_1	$(M_1, B_1, \dots, M_4)$	$L_{1,1}$	$L_{1,2}$	(30, 31, 31)	(35, 46, 36)	0.850218
$L_{1,1}$	$(M_1, B_1, M_2, B_2, M_3)$			(0, 0)	(30, 31)	0.850417
$L_{1,2}$	$(M_2, B_2, M_3, B_3, M_4)$			(0, 0)	(30, 31)	0.850417
L_2	$(M_2, B_2, \dots, M_5)$	$L_{2,1}$	$L_{2,2}$	(30, 31, 31)	(35, 46, 36)	0.850218
$L_{2,1}$	$(M_2, B_2, M_3, B_3, M_4)$			(0, 0)	(30, 31)	0.850417
$L_{2,2}$	$(M_3, B_3, M_4, B_4, M_5)$			(0, 0)	(30, 31)	0.850417

6. Numerical Experiments

In this section, numerical experiments are provided to show the accuracy and efficiency of the recursive algorithms. As discussed in Section 4, homogeneous production lines are evaluated by the robust decomposition method [21] and inhomogeneous lines are evaluated by the ADDX algorithm [19]. Since the accuracy of the ADDX algorithm is not as good as the robust decomposition method, most experiments are based on homogeneous lines. First, we consider a set of BAPs in small production lines with $K = 5$. The solutions obtained by the recursive algorithms are compared to the optimal solutions obtained by complete enumeration. Then, we consider three sets of BAPs in long production lines with $K = 20, 40, 80$, respectively. The complete enumeration method is not applicable to large-scale BAPs due to the extremely long computation time. We use the gradient search algorithm to solve these BAPs, and the results serve as benchmarks for comparing the performance of the recursive algorithms. Finally, we consider several BAPs in an inhomogeneous line. All the algorithms were implemented in the C# language and run on a computer with a 2.7 GHz Intel Core i7 processor and 8 GB RAM.

6.1. Five-Machine Lines

We studied one hundred five-machine production lines with machine reliability parameters randomly generated from the following sets:

$$MTTF_i = \frac{1}{\lambda_i} \in \{5, 6, \dots, 200\}, i = 1, \dots, K, \quad (5)$$

$$MTTR_i = \frac{1}{\mu_i} \in \{5, 6, \dots, 200\}, i = 1, \dots, K. \quad (6)$$

Moreover, a constraint was used to guarantee the generated machine parameters were realistic: the efficiency of each machine had to be greater than or equal to 0.85, i.e.,

$$e_i = \frac{MTTF_i}{MTTF_i + MTTR_i} \geq 0.85, i = 1, \dots, K. \quad (7)$$

If the parameters of a machine violated the above constraint, they were rejected and regenerated.

The desired production rate P^* of the 100 lines was set to 0.8. We solved the BAPs by the gradient, RBD, and ROD algorithms, respectively. The results (see Supplementary Materials) show that all the algorithms lead to the optimal solutions, which were verified by complete enumeration. For example, consider the production line in Table 5. Each algorithm obtains the optimal solution (57, 68, 58, 45) with a production rate of 0.800048. For the 100 small lines, all the algorithms are very efficient and the average computation time of each algorithm is within 0.1 s.

Table 5. A five-machine production line.

i	1	2	3	4	5
$MTTF_i$	138	184	137	173	154
$MTTR_i$	23	21	21	16	24

6.2. Twenty-Machine Lines

Similar to the previous experiment, one hundred 20-machine production lines were randomly generated and the desired production rate P^* was set to 0.8. We compare the performance of the RBD and ROD algorithms to that of the gradient algorithm. Since the

objective of the BAP is to minimize the total buffer capacity of a production line, we define the percentage error of each algorithm as follows:

$$\epsilon = \left| \frac{N^* - N^{best}}{N^{best}} \right| \times 100\%, \quad (8)$$

where N^* is the minimum total buffer capacity obtained by the algorithm under consideration and N^{best} is the best minimum total buffer capacity obtained by all the three algorithms.

The results show that all the algorithms lead to the same N^* for each line, and thus, the percentage error of each algorithm is zero. However, this does not mean they lead to the same solution, since the optimal buffer allocation may not be unique. For example, consider the production line described in Table 6. The optimization results obtained by the three algorithms are given in Table 7. As can be seen, the solutions obtained by the RBD and ROD algorithms are the same, but they differ from the solution obtained by the gradient algorithm in b_{10}^* and b_{11}^* . Nevertheless, all the algorithms lead to the same minimum total buffer capacity, i.e., $N^* = 869$.

Table 6. A twenty-machine production line.

i	1	2	3	4	5	6	7	8	9	10
$MTTF_i$	147	42	128	197	139	96	175	137	53	84
$MTTR_i$	21	7	13	11	23	10	18	8	6	5
i	11	12	13	14	15	16	17	18	19	20
$MTTF_i$	132	96	193	128	70	98	135	200	79	167
$MTTR_i$	15	16	28	22	12	7	12	24	6	17

Table 7. Optimization results for the twenty-machine production line.

Algorithm	$b^* = (b_1^*, \dots, b_{19}^*)$	N^*	$P(b^*)$
Gradient	(43, 42, 35, 39, 48, 36, 25, 24, 25, 29, 60, 99, 113, 98, 46, 36, 35, 21, 15)	869	0.800055
RBD	(43, 42, 35, 39, 48, 36, 25, 24, 25, 28, 61, 99, 113, 98, 46, 36, 35, 21, 15)	869	0.800056
ROD	(43, 42, 35, 39, 48, 36, 25, 24, 25, 28, 61, 99, 113, 98, 46, 36, 35, 21, 15)	869	0.800056

Regarding the computational efficiency, the three algorithms differ significantly. As illustrated in Figure 7, the recursive algorithms show a great advantage over the gradient algorithm. The average computation time for optimizing the 100 lines by the gradient algorithm is 100.1 s, while the average computation times using the RBD and ROD algorithms are 13.0 s and 3.7 s, respectively. Clearly, the recursive algorithms are much more efficient than the gradient algorithm.

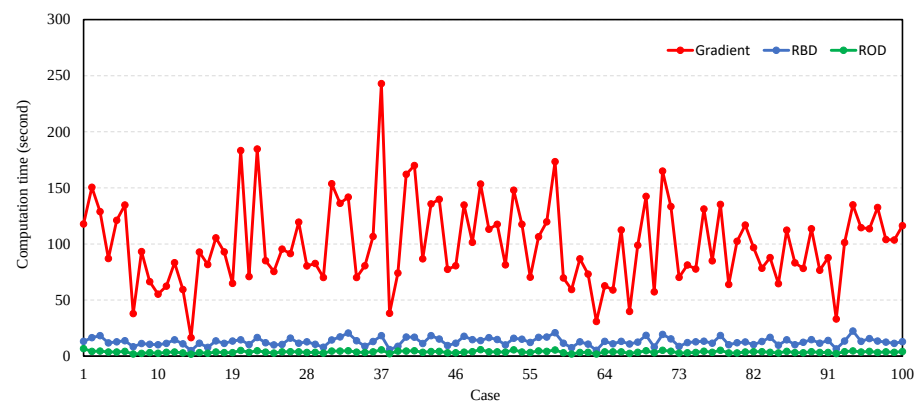


Figure 7. Computation times for optimizing the 100 twenty-machine lines.

6.3. Forty-Machine Lines

As in the previous two experiments, one hundred 40-machine production lines were randomly generated and the desired production rate P^* was set to 0.8. Table 8 compares the performance of the gradient, RBD, and ROD algorithms. The gradient and RBD algorithms always obtain the best result N^{best} in the 100 problems, whereas the ROD algorithm obtains N^{best} in 76 problems. Although the ROD algorithm does not obtain N^{best} in the remaining 24 problems, the maximum deviation is just 1 buffer unit, with a maximum percentage error of only 0.08%.

The deviation of the ROD algorithm is due to the non-uniqueness of the solutions to the decomposed sub-problems. Note that the optimal solution to an integer-valued BAP is essentially an approximation of the optimal solution to the corresponding real-valued BAP. Thus, an integer-valued BAP may have multiple optimal solutions. Consider the pseudocode of the ROD algorithm in Table 6. Due to non-uniqueness, the solutions of the sub-lines L_1 and L_2 , i.e., $(x_1, \dots, x_{K-s-1})$ and $(y_{s+1}, \dots, y_{K-1})$, may not be accurate lower bounds of the respective buffer capacities in the original line L . One way to avoid this problem is to reduce x_i , $i = 1, \dots, K - s - 1$ and y_i , $i = s + 1, \dots, K - 1$ by one buffer unit. For example, consider the production line described in Table 9. The minimum total buffer capacities obtained by the gradient, RBD, and ROD algorithms are 1489, 1489, and 1490, respectively, with corresponding computation times of 2849.7 s, 119.8 s, and 11.4 s. If the ROD algorithm is modified as described above, it leads to the same solution as the gradient and RBD algorithm, but the computation time increases from 11.4 s to 182.7 s. We feel that the modification is unnecessary, because the ROD algorithm is accurate enough. The average percentage error for the 100 BAPs is only 0.02%.

Figure 8 compares the computation times of the three algorithms. The average time for optimizing the 100 lines by the gradient algorithm is 2961.7 s, while the average computation times using the RBD and ROD algorithms are 134.7 s and 15.8 s, respectively. Once again, the recursive algorithms show a great advantage over the gradient algorithm.

Table 8. Performance comparison of the gradient, RBD, and ROD algorithms for 100 BAPs in forty-machine production lines.

Algorithm	#Problems to Find N^{best}	Max. Deviation from N^{best}	Avg. ϵ	Max. ϵ	Avg. Computation Times (s)
Gradient	100	0	0.00%	0.00%	2961.7
RBD	100	0	0.00%	0.00%	134.7
ROD	76	1	0.02%	0.08%	15.8

Table 9. A forty-machine production line.

i	1	2	3	4	5	6	7	8	9	10
$MTTF_i$	110	78	47	193	148	171	191	162	182	83
$MTTR_i$	17	8	8	12	23	13	28	5	17	12
i	11	12	13	14	15	16	17	18	19	20
$MTTF_i$	195	142	69	87	161	130	105	162	180	150
$MTTR_i$	16	12	6	14	6	13	9	9	18	16
i	21	22	23	24	25	26	27	28	29	30
$MTTF_i$	181	96	132	183	139	55	62	151	192	139
$MTTR_i$	20	5	13	6	19	8	9	19	11	24
i	31	32	33	34	35	36	37	38	39	40
$MTTF_i$	168	170	164	199	189	146	162	150	166	122
$MTTR_i$	7	29	28	22	5	9	6	16	23	5

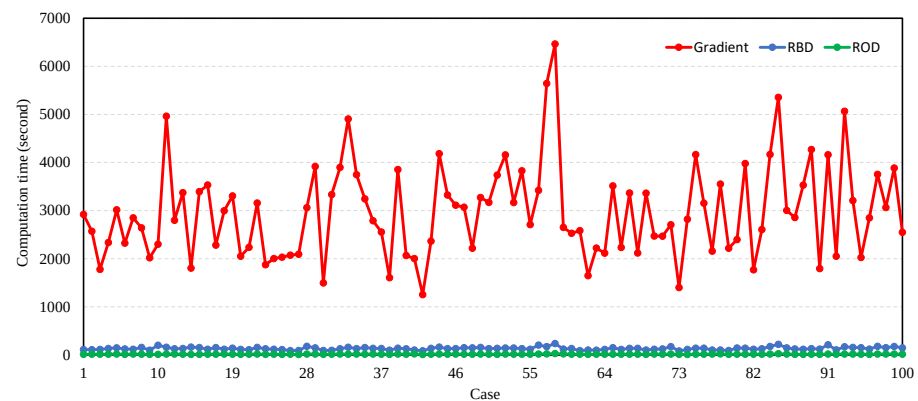


Figure 8. Computation times for optimizing the 100 forty-machine lines.

6.4. Eighty-Machine Line

We consider an 80-machine line that is composed of two 40-machine blocks. Each block is the same as the 40-machine line described in Table 9. The desired production rate P^* was set to 0.8. The gradient, RBD, and ROD algorithms produced minimum total buffer capacities of 3092, 3092, and 3095, respectively, with corresponding computation times of 71,448.8 s, 987.3 s, and 45.2 s. The gradient algorithm takes an incredibly long time to solve this problem, whereas the ROD algorithm takes only a short time to obtain a near-optimal solution with an error of less than 0.1%.

6.5. Inhomogeneous Line

Table 10 shows the data of a real-life production line, which are taken from [54]. Clearly, the line is inhomogeneous, since the machine processing times are different. The isolated production rate of each machine is presented in the last column of Table 11. Note that the worst machine (M_{18}) has an isolated production rate of 1.948, which is an upper bound of the production rate of the whole line. Table 11 presents the results for BAPs with different target production rates. Tempelmeier [54] studied the case of $P^* = 1.80$ and obtained the result $N^* = 485$, which is the same as the results obtained by the proposed algorithms. We further studied the cases of $P^* = 1.90$ and $P^* = 1.94$. It can be seen that the RBD and ROD algorithms lead to the same total buffer capacity as the gradient algorithm, but they are much faster.

Table 10. A real-life production line.

i	Processing Time (min) $\tau_i = 1/s_i$	Availability e_i	$MTTR_i$ (min)	Isolated Production Rate $\rho_i = s_i e_i$
1	0.47	0.978	49	2.081
2	0.47	0.98	41	2.085
3	0.493	0.984	55	1.996
4	0.488	0.993	28	2.035
5	0.463	0.984	47	2.125
6	0.463	0.977	28	2.11
7	0.498	0.987	68	1.982
8	0.467	0.97	35	2.075
9	0.473	0.967	43	2.044
10	0.487	0.996	33	2.045
11	0.498	0.997	36	2.002
12	0.478	0.996	31	2.082
13	0.488	0.97	60	1.988

Table 10. Cont.

i	Processing Time (min) $\tau_i = 1/s_i$	Availability e_i	$MTTR_i$ (min)	Isolated Production Rate $\rho_i = s_i e_i$
14	0.488	0.988	39	2.025
15	0.482	0.975	38	2.023
16	0.43	0.989	48	2.281
17	0.473	0.981	47	2.097
18	0.5	0.992	55	1.948
19	0.467	0.974	43	2.071

Table 11. Optimization results for the real-life production line.

Target Rate P^*	Algorithm	Total Capacity N^*	Actual Rate $P(b^*)$	Computation Time (s)
1.80	Gradient	485	1.800202	46.1
	RBD	485	1.800202	9.2
	ROD	485	1.800202	7.1
1.90	Gradient	1201	1.900065	63.3
	RBD	1201	1.900065	6.8
	ROD	1201	1.900064	4.5
1.94	Gradient	1958	1.940010	86.6
	RBD	1958	1.940010	8.7
	ROD	1958	1.940009	3.9

6.6. Discussion

The above experiments show that the accuracy of the RBD and ROD algorithms is quite good. For production lines with five machines, they always produce the optimal solutions. For production lines with no more than 20 machines, they are as accurate as the gradient algorithm. Although the ROD algorithm has a small margin of error for longer lines (less than 0.1%), the accuracy is acceptable, considering the fact that the machine parameter data available on a factory floor are rarely within 1% of their actual values. In terms of computational efficiency, the RBD and ROD algorithms show a great advantage over the gradient algorithm. For long production lines, the computation time can be reduced by more than 95%. Therefore, the proposed algorithms achieve a good trade-off between solution quality and computation time.

In general, a gradient-based search algorithm is more efficient than combinatorial approaches (e.g., genetic algorithm, simulated annealing, etc.), since it exploits the concavity of BAPs [7,8]. Theoretical analysis and extensive numerical experiments show this property does hold. See [7,8] for a detailed discussion. For production systems with large buffers, the error of the proposed recursive algorithms can be neglected, but they are more efficient than the gradient algorithm for BAPs in long production lines. However, for just-in-time production systems with small buffer capacities, it is recommended to use the gradient algorithm directly, because the computational complexity is not high when the total buffer capacity is small.

Another limitation of the RBD and ROD algorithms is that the gradient of the production rate has to be evaluated. In some cases, the gradient is unavailable. For example, if the machine failure time is not exponentially distributed, the production rate cannot be evaluated by the robust decomposition method or the ADDX method. In this case, the proposed algorithms cannot be applied. An alternative solution approach is to combine metaheuristics algorithms with simulation-based evaluation methods [30].

7. Conclusions

In this paper, we propose two recursive algorithms (i.e., the RBD algorithm and the ROD algorithm) for solving the buffer allocation problem of minimizing the total buffer capacity of a line to achieve a desired production rate. The algorithms decompose a long line into two sub-lines, optimize them recursively, and combine their solutions to find the optimal buffer distribution of the original line. The two algorithms differ in how the line is decomposed: the RBD algorithm does not allow overlapping buffers in the decomposed sub-lines, whereas the ROD algorithm does. Numerical experiments were performed to compare the performance of the two algorithms to that of a gradient search algorithm. The results show that both algorithms achieve the same level of accuracy as the gradient algorithm, but they are much more efficient. For long production lines, the two recursive algorithms reduced the computation time dramatically. The RBD algorithm is slightly more accurate than the ROD algorithm, whereas the ROD algorithm is faster.

In a future study, the proposed algorithms will be extended to production systems with complex topologies, such as assembly/disassembly systems. Another future research direction is the adoption of artificial intelligence techniques to solve various types of buffer allocation problems.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/app14219953/s1>, Data: randomly generated production lines.

Author Contributions: Conceptualization, L.L.; methodology, L.L.; software, L.L.; validation, L.L. and J.Z.; formal analysis, L.L.; investigation, L.L. and M.D.; resources, L.L. and M.D.; data curation, L.L. and J.Z.; writing—original draft preparation, L.L. and M.D.; writing—review and editing, L.L.; visualization, L.L. and J.Z.; supervision, L.L. and J.Z.; project administration, L.L.; funding acquisition, L.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported by the National Natural Science Foundation of China (Grant No. 52075539).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The authors confirm that the data supporting the findings of this study are available within the article and Supplementary Materials.

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

References

1. Papadopoulos, C.T.; Li, J.; O’Kelly, M.E.J. A classification and review of timed Markov models of manufacturing systems. *Comput. Ind. Eng.* **2019**, *128*, 219–244. [\[CrossRef\]](#)
2. Weiss, S.; Schwarz, J.A.; Stollatz, R. The Buffer Allocation Problem in Production Lines: Formulations, Solution Methods, and Instances. *IIE Trans.* **2019**, *51*, 456–485. [\[CrossRef\]](#)
3. Demir, L.; Tunali, S.; Eliyi, D.T. The state of the art on buffer allocation problem: A comprehensive survey. *J. Intell. Manuf.* **2014**, *25*, 371–392. [\[CrossRef\]](#)
4. Li, J.; Meerkov, S.M. *Production Systems Engineering*; Springer: New York, NY, USA, 2009.
5. Dolgui, A.; Ereemeev, A.; Kovalyov, M.Y.; Sigaev, V. Complexity of Buffer Capacity Allocation Problems for Production Lines with Unreliable Machines. *J. Math. Model. Algorithms Oper. Res.* **2013**, *12*, 155–165. [\[CrossRef\]](#)
6. Smith, J.M.; Cruz, F.R.B. The Buffer Allocation Problem for General Finite Buffer Queueing Networks. *IIE Trans.* **2005**, *37*, 343–365. [\[CrossRef\]](#)
7. Shi, C.; Gershwin, S.B. An Efficient Buffer Design Algorithm for Production Line Profit Maximization. *Int. J. Prod. Econ.* **2009**, *122*, 725–740. [\[CrossRef\]](#)
8. Gershwin, S.B.; Schor, J.E. Efficient algorithms for buffer space allocation. *Ann. Oper. Res.* **2000**, *93*, 117–144. [\[CrossRef\]](#)
9. Hudson, S.; McNamara, T.; Shaaban, S. Unbalanced lines: Where are we now? *Int. J. Prod. Res.* **2015**, *53*, 1895–1911. [\[CrossRef\]](#)
10. Papadopoulos, H.T.; Heavey, C. Queueing Theory in Manufacturing Systems Analysis and Design: A Classification of Models for Production and Transfer Lines. *Eur. J. Oper. Res.* **1996**, *92*, 1–27. [\[CrossRef\]](#)
11. Dallery, Y.; Gershwin, S.B. Manufacturing flow line systems: A review of models and analytical results. *Queueing Syst.* **1992**, *12*, 3–94. [\[CrossRef\]](#)

12. Li, J.; Blumenfeld, D.E.; Huang, N.; Alden, J.M. Throughput analysis in production systems: Recent advances and future topics. *Int. J. Prod. Res.* **2009**, *47*, 3823–3851. [\[CrossRef\]](#)
13. Romero-Silva, R.; Shaaban, S. Influence of Unbalanced Operation Time Means and Uneven Buffer Allocation on Unreliable Merging Assembly Line Efficiency. *Int. J. Prod. Res.* **2019**, *57*, 1645–1666. [\[CrossRef\]](#)
14. Spinellis, D.; Papadopoulos, C.T. A Simulated Annealing Approach for Buffer Allocation in Reliable Production Lines. *Ann. Oper. Res.* **2000**, *93*, 373–384. [\[CrossRef\]](#)
15. Helber, S.; Schimmelpfeng, K.; Stolletz, R.; Lagershausen, S. Using linear programming to analyze and optimize stochastic flow lines. *Ann. Oper. Res.* **2011**, *182*, 193–211. [\[CrossRef\]](#)
16. Gershwin, S.B. An efficient decomposition method for the approximate evaluation of tandem queues with finite storage space and blocking. *Oper. Res.* **1987**, *35*, 291–305. [\[CrossRef\]](#)
17. Dallery, Y.; David, R.; Xie, X.L. Approximate analysis of transfer lines with unreliable machines and finite buffers. *IEEE Trans. Autom. Control* **1989**, *34*, 943–953. [\[CrossRef\]](#)
18. Lim, J.T.; Meerkov, S.M.; Top, F. Homogeneous asymptotically reliable serial production lines: Theory and a Case Study. *IEEE Trans. Autom. Control* **1990**, *35*, 524–534. [\[CrossRef\]](#)
19. Burman, M.H. New Results in Flow Line Analysis. Ph.D. Thesis, OR Center, MIT, Cambridge, MA, USA, 1995.
20. Dallery, Y.; Le Bihan, H. An Improved Decomposition Method for the Analysis of Production Lines with Unreliable Machines and Finite Buffers. *Int. J. Prod. Res.* **1999**, *37*, 1093–1117. [\[CrossRef\]](#)
21. Le Bihan, H.; Dallery, Y. A robust decomposition method for the analysis of production lines with unreliable machines and finite buffers. *Ann. Oper. Res.* **2000**, *93*, 265–297. [\[CrossRef\]](#)
22. Levantesi, R.; Matta, A.; Tolio, T. Performance evaluation of production lines with random processing times, multiple failure modes and finite buffer capacity—Part II: Decomposition. In *Analysis and Modeling of Manufacturing Systems*; Gershwin, S.B., Dallery, Y., Papadopoulos, C.T., Smith, J.M., Eds.; Kluwer Academic: Dordrecht, The Netherlands, 2003; pp. 201–219.
23. Bierbooms, R.; Adan, I.; Vuuren, M. Approximate Performance Analysis of Production Lines with Continuous Material Flows and Finite Buffers. *Stoch. Model.* **2013**, *29*, 1–30. [\[CrossRef\]](#)
24. Colledani, M.; Gershwin, S.B. A decomposition method for approximate evaluation of continuous flow multi-stage lines with general markovian machines. *Ann. Oper. Res.* **2013**, *209*, 5–40. [\[CrossRef\]](#)
25. Xi, S.; Chen, Q.; Smith, J.M.; Mao, N.; Yu, A.; Zhang, H. A New Method for Solving Buffer Allocation Problem in Large Unbalanced Production Lines. *Int. J. Prod. Res.* **2020**, *58*, 6846–6867. [\[CrossRef\]](#)
26. 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. [\[CrossRef\]](#)
27. Spinellis, D.; Papadopoulos, C.; Smith, J.M. Large Production Line Optimization Using Simulated Annealing. *Int. J. Prod. Res.* **2000**, *38*, 509–541. [\[CrossRef\]](#)
28. Dolgui, A.; Ereemeev, A.; Kolokolov, A.; Sigaev, V. A Genetic Algorithm for the Allocation of Buffer Storage Capacities in a Production Line with Unreliable Machines. *J. Math. Model. Algorithms* **2002**, *1*, 89–104. [\[CrossRef\]](#)
29. Demir, L.; Tunali, S.; Løkketangen, A. A Tabu Search Approach for Buffer Allocation in Production Lines with Unreliable Machines. *Eng. Optim.* **2011**, *43*, 213–231. [\[CrossRef\]](#)
30. Yegul, M.F.; Erenay, F.S.; Striepe, S.; Yavuz, M. Improving Configuration of Complex Production Lines Via Simulation-Based Optimization. *Comput. Ind. Eng.* **2017**, *109*, 295–312. [\[CrossRef\]](#)
31. Massim, Y.; Yalaoui, F.; Amodeo, L.; Chatelet, E.; Zebblah, A. Efficient combined immune-decomposition algorithm for optimal buffer allocation in production lines for throughput and profit maximization. *Comput. Oper. Res.* **2010**, *37*, 611–620. [\[CrossRef\]](#)
32. Koyuncuoglu, M.U.; Demir, L. A Comparison of Combat Genetic and Big Bang–Big Crunch Algorithms for Solving the Buffer Allocation Problem. *J. Intell. Manuf.* **2021**, *32*, 1529–1546. [\[CrossRef\]](#)
33. Kose, S.Y.; Kilincci, O. A Multi-Objective Hybrid Evolutionary Approach for Buffer Allocation in Open Serial Production Lines. *J. Intell. Manuf.* **2020**, *31*, 33–51. [\[CrossRef\]](#)
34. Dolgui, A.; Ereemeev, A.; Sigaev, V. HBBA: Hybrid Algorithm for Buffer Allocation in Tandem Production Lines. *J. Intell. Manuf.* **2007**, *18*, 411–420. [\[CrossRef\]](#)
35. Chan, F.T.S.; Ng, E.Y.H. Comparative Evaluations of Buffer Allocation Strategies in a Serial Production Line. *Int. J. Adv. Manuf. Technol.* **2002**, *19*, 789–800. [\[CrossRef\]](#)
36. 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. [\[CrossRef\]](#)
37. 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. [\[CrossRef\]](#)
38. Magnanini, M.C.; Walter, T.; Tullio, A.M.T. A novel algorithm for optimal buffer allocation in automated asynchronous unreliable lines. *Int. J. Adv. Manuf. Technol.* **2022**, *122*, 4405–4419. [\[CrossRef\]](#)
39. Yuzukirmizi, M.; Smith, J.M. Optimal buffer allocation in finite closed networks with multiple servers. *Comput. Ind. Eng.* **2008**, *35*, 2579–2598. [\[CrossRef\]](#)
40. Papadopoulos, H.T.; Vidalis, M.I. Optimal Buffer Storage Allocation in Balanced Reliable Production Lines. *Int. Oper. Res.* **1998**, *5*, 325–339. [\[CrossRef\]](#)

41. Weiss, S.; Matta, A.; Stolletz, R. Optimization of Buffer Allocations in Flow Lines with Limited Supply. *IIE Trans.* **2018**, *50*, 191–202. [\[CrossRef\]](#)
42. Papadopoulos, H.T.; Vidalis, M.I. A Heuristic Algorithm for the Buffer Allocation in Unreliable Unbalanced Production Lines. *Comput. Ind. Eng.* **2001**, *41*, 261–277. [\[CrossRef\]](#)
43. Sabuncuoglu, I.; Erel, E.; Gocgun, Y. Analysis of Serial Production Lines: Characterisation Study and a New Heuristic Procedure for Optimal Buffer Allocation. *Int. J. Prod. Res.* **2006**, *44*, 2499–2523. [\[CrossRef\]](#)
44. Dos Santos, L.M.A.L.; da Costa, M.B.; Kothe, J.V.; Benitez, G.B.; Schaefer, J.L.; Baierle, I.C.; Nara, E.O.B. Industry 4.0 collaborative networks for industrial performance. *J. Manuf. Technol. Manag.* **2021**, *32*, 245–265. [\[CrossRef\]](#)
45. Schaefer, J.L.; Tardio, P.R.; Baierle, I.C.; Nara, E.O.B. GIANN—A Methodology for Optimizing Competitiveness Performance Assessment Models for Small and Medium-Sized Enterprises. *Adm. Sci.* **2023**, *13*, 56. [\[CrossRef\]](#)
46. Can, B.; Heavey, C. A comparison of genetic programming and artificial neural networks in metamodeling of discrete-event simulation models. *Comput. Oper. Res.* **2012**, *39*, 424–436. [\[CrossRef\]](#)
47. Tsadiras, A.K.; Papadopoulos, C.T.; O’Kelly, M.E.J. An Artificial Neural Network Based Decision Support System for Solving the Buffer Allocation Problem in Reliable Production Lines. *Comput. Ind. Eng.* **2013**, *66*, 1150–1162. [\[CrossRef\]](#)
48. Shi, C.; Gershwin, S.B. A segmentation approach for solving buffer allocation problems in large production systems. *Int. J. Prod. Res.* **2016**, *54*, 6121–6141. [\[CrossRef\]](#)
49. Li, L.; Qian, Y.L.; Yang, Y.M.; Du, K. A Fast Algorithm for Buffer Allocation Problem. *Int. J. Prod. Res.* **2016**, *54*, 3243–3255. [\[CrossRef\]](#)
50. Dallery, Y.; Liu, Z.; Towsley, D. Properties of fork/join queueing networks with blocking under various operating mechanisms. *IEEE Trans. Robot. Autom.* **1997**, *13*, 503–518. [\[CrossRef\]](#)
51. Schor, J.E. Efficient Algorithms for Buffer Allocation. Master’s Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.
52. Weiss, S.; Stolletz, R. Buffer Allocation in Stochastic Flow Lines via Sample-Based Optimization with Initial Bounds. *OR Spectr.* **2015**, *37*, 869–902. [\[CrossRef\]](#)
53. Hillier, F.S.; So, K.C.; Boling, R.W. Notes: Toward characterizing the optimal allocation of storage space in production line systems with variable processing times. *Manag. Sci.* **1993**, *39*, 126–133. [\[CrossRef\]](#)
54. Tempelmeier, H. Practical considerations in the optimization of flow production systems. *Int. J. Prod. Res.* **2003**, *41*, 149–170. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.