AppliedMath

MDPI

Article

Assessing by Simulation the Effect of Process Variability in the SALB-1 Problem

Luis A. Moncayo-Martínez * and Elias H. Arias-Nava

Department of Industrial Engineering and Operations, Instituto Tecnológico Autónomo de México , Río Hondo 1, Mexico City 01080, Mexico; elias.arias@itam.mx
* Correspondence: luis.moncayo@itam.mx

Abstract: The simple assembly line balancing (SALB) problem is a significant challenge faced by industries across various sectors aiming to optimise production line efficiency and resource allocation. One important issue when the decision-maker balances a line is how to keep the cycle time under a given time across all cells, even though there is variability in some parameters. When there are stochastic elements, some approaches use constraint relaxation, intervals for the stochastic parameters, and fuzzy numbers. In this paper, a three-part algorithm is proposed that first solves the balancing problem without considering stochastic parameters; then, using simulation, it measures the effect of some parameters (in this case, the inter-arrival time, processing times, speed of the material handling system which is manually performed by the workers in the cell, and the number of workers who perform the tasks on the machines); finally, the add-on OptQuest in SIMIO solves an optimisation problem to constrain the cycle time using the stochastic parameters as decision variables. A Gearbox instance from literature is solved with 15 tasks and 14 precedence rules to test the proposed approach. The deterministic balancing problem is solved optimally using the open solver GLPK and the Pyomo programming language, and, with simulation, the proposed algorithm keeps the cycle time less than or equal to 70 s in the presence of variability and deterministic inter-arrival time. Meanwhile, with stochastic inter-arrival time, the maximum cell cycle is 72.04 s. The reader can download the source code and the simulation models from the GitHub page of the authors.

Keywords: SALB; simulation; stochastic optimisation; SIMIO; OptQuest; GLPK

MSC: 90-10

Citation: Moncayo-Martínez, L.A.; Arias-Nava, E.H. Assessing by Simulation the Effect of Process Variability in the SALB-1 Problem. *AppliedMath* **2023**, *3*, 563–581. https://doi.org/10.3390/appliedmath3030030

Academic Editor: Armin Fügenschuh

Received: 13 June 2023
Revised: 8 July 2023
Accepted: 17 July 2023
Published: 28 July 2023

1. Introduction

In the time of supply chain (SC) digitalisation, or Industry 4.0, it is relatively easy to track events that happen throughout the SC's stages. Therefore, nowadays, SC managers must understand, monitor, and control the operations of sourcing, logistics, production, and retail delivery [1]. From the production point of view, one important event is delivering products to the customer on time and in the correct quantity. Thus, production managers must set a cycle time to produce enough to meet the demand in the available time (e.g., an eight-hour shift). To this end, engineers in manufacturing (since Henry Ford's time) developed the assembly lines that have been frequently cited as one of the greatest inventions of the modern era due to their ability to increase productivity [2].

In the manufacturing engineering literature, this is known as the assembly line balancing problem (ALB), in which a product is fully assembled once all the required operations or tasks (to produce it) are completed in a specific order. The ALB problem aims to distribute the tasks among the available manufacturing cells to achieve a balanced workload and maximise utilisation, defined as the amount of time for which the cell is not idle. Mass manufacturers use assembly lines in various industrial sectors, including the manufacturing of white goods, consumer electronics, lorries, and aeroplanes [3].

Solving the ALB problem is not easy because there are a lot of constraints such as limited resources, process variabilities, and precedence relationships. Still, when they are optimised, the operating costs decrease, making the company more competitive [4]. From the optimisation point of view, finding an optimal solution to the assembly line problem is challenging and computationally expensive due to the exponential growth of the search space. Therefore, it falls under the classification of NP-hard problems.

Since the first mathematical formulation of the ALB problem in 1955 [5], this issue has attracted the attention of researchers and practitioners. The ALB problem is divided into Simple (SALB) and General (GALB) ALB problems. In the SALB problem, various tasks or operations are performed sequentially to produce a final product. In the GALB problem, U-shaped lines, parallel stations, or processing alternatives extend the SALB problem.

Meanwhile, in the SALB-1 problem, the objective is to reduce the number of manufacturing cells given a constant cycle time; in the SALB-2 problem, the aim is to minimise the cycle time given a constant number of cells. An important characteristic of the problem is whether it has deterministic and/or stochastic parameters (e.g., processing times).

According to the most recent work about a comprehensive review of new trends in the ALB problem (see [6–8]), the stochastic ALB problem and its variants have been solved by formulating a mathematical model, adding stochasticity to the processing times, and solving the problem using tailor-made heuristics, metaheuristics, or constraint programming. On the other hand, simulations have been used to test some stochastic scenarios. In this paper, some relevant works are briefly summarised, and the insights and novelties of the proposed approach are highlighted at the end of this section.

The most used source of stochasticity reported in the specialised literature (see [7]) is processing time. Sometimes, it is assumed that the processing time varies over time without any specific pattern [9], but other approaches assume that the information is limited; thus, the mean and standard deviation are the two pieces of known information [10]. Other works model the variability using interval processing times, i.e., the processing time has a lower, a nominal, and an upper value; using this approach, the cycle time [11] and the number of stations [12] have been minimised. Fuzzy numbers, which allow for a range of potential values with variable degrees of membership or possibility, are another technique used to model the variability of processing times. Using triangular fuzzy numbers, the fuzzy ALB problem is solved by maximising the line efficiency [13].

Concerning the solution methods when non-deterministic processing time is taken into account, stochastic programming models [14], constraint programming[15], and metaheuristics [16] are the most widely used solution techniques.

In [17], not only is the processing time non-deterministic, but also elements of the flow of the process, i.e., the arrival of orders or changes in the demand, are stochastic. To solve the problem, a mixed-integer programming model is formulated to maximise the stations' utilisation to equally distribute the workload among the stations. The programming model is solved using constraint programming to assign stations to tasks, and queue theory to compute the solution's performance.

A tailor-made heuristic, named the multi-started neighbourhood search heuristic, is proposed to solve a motorcycle assembly line in [4]. The only source of variability is the processing times which follow a normal distribution. Their results are compared by modifying the classical ranked positional weight method and tested via simulation to create several scenarios without any optimisation capability. Another tailor-made exact heuristic is developed in [11] for a two-part model that minimises the cycle time. The first part keeps the station time under a given bound, and the second part assigns the maximum percentage of tasks to the station. The problem is solved optimally using Bender decomposition. The SALB-2 problem was solved to minimise the cycle time for a number of workstations subjected to the probability that the workload does not exceed the cycle time for the whole assembly line [18].

Meta-heuristics have been applied to solve the stochastic ALB problem. Modelling the processing time as a triangular fuzzy membership function, a genetic algorithm (GA)

approach was used in [19] to minimise the fuzzy cycle time and the fuzzy index representing the workload of the line; this approach is extended in [13] by maximising the line efficiency. Another multi–objective ALB problem is solved via simulated annealing in [20]. One objective is to distribute the workload as equally as possible (this is called the smoothed index), and the other is to minimise the cost. The source of stochasticity in their model is the probability that the sum of the mean task times does not exceed the station capacity. Particle Swarm Optimisation solves a multi–objective model in [21], which minimises the cycle time, the equipment cost, and the smoothed index when the only known information is the lower and upper bound of tasks' processing time, and it depends on the workers' learning.

Regarding simulation, two methodologies are used when the ALB is solved with stochasticity. In the first one, a mathematical model is formulated and then some scenarios for various variables are created; see [22,23]. In the second one, a simulation model is created using software; then, some scenarios are created to test it (see [24–27]), and in some cases, the simulation model is created, then OptQuest is used to optimise a desired variable; see [28–30].

A mathematical model is developed to minimise the overlapping and stopped operations in [22]; then, a genetic algorithm-based solution method is used to solve it, and finally, some scenarios are created to compute and check the effect of variability. A similar approach is used in [23], in which a mathematical model is formulated to minimise the cycle time. To solve it, the authors used a standard optimisation solver to find the answer and then simulation software to test the number of pallets. In these models, the objective is to stress the optimisation solution computed by the mathematical model, but there is no evidence of using simulation as a tool to find the value of the variables to obtain the solution to the mathematical model.

In some applications, such as the apparel industry [24], garment production [25] or automotive manufacturing [26], the ALB problem is solved using standard simulation software to create some scenarios in which the value of the number of operators and the buffer size is changed. For these works, there are no optimisation capabilities or optimisation model.

The optimisation capabilities of OptQuest have been applied to optimise a five–stage production cell in [28]. In this case, the model is implemented in commercial software without a previous study of optimality; i.e., there is no mathematical model to optimise the balancing of the number of cells to accomplish the desired cycle time or minimise the number of cells. Similar to the work in a trouser assembly line in [29] and in a kids' pants line in [30], OptQuest is used without previous optimisation criteria defined by a mathematical model.

In summary, in the papers presenting a systematic review of the literature on the ALB problem, it is reported that the only source of uncertainty is the processing times [6–8]. Those works do not consider the pace at which jobs or parts arrive at the first cell, i.e., the inter-arrival time. Moreover, the physical space (or layout) in which the assembly line will operate is also not considered. The inter-arrival time adds uncertainty because the time at which the jobs or parts enter the first cell is not fixed; thus, the cycle time could be greater than the one required to deliver products on time. On the other hand, the physical space adds variability because the the distances among the machines are considered to be zero for the solution methods; thus, the speed of the material handling system (i.e., the speed at which workers move the processing units from station to station) could increase the cells' cycle time. Another issue in the cell layout is the number of workers; this is important given that it is assumed that there is one worker in each cell.

This work aims to balance a simple assembly line (SALB-1) so that, in the presence of variability, the sum of cells' time content is less than or equal to the required cycle time to deliver products on time. The line is balanced by solving the mixed-integer programming model using the GLPK solver via Pyomo. Then, variability is added to the balancing using a simulation model with the following stochastic parameters: inter-arrival time, processing times, and the number of workers to measure the effect on the overall cycle time. Also, a

20×10 m^2 shop layout is set in which the cells are physically placed; thus, the simulation enables the decision-maker to monitor and quantify how process variability impacts the line performance. It aids their comprehension of the trade–offs brought on by unpredictability. Finally, using the add–on OptQuest in SIMIO, the values of the stochastic parameter that satisfies the cycle time are optimised.

The proposed algorithm is relevant since most of the parameters in real-world implementations are stochastic; as a result, solutions derived using deterministic optimum or near-optimum approaches cannot be guaranteed to comply with the cycle time. Managers will not deliver products on schedule as a result. Some works reported in the literature include variability in the tasks' processing time, but there is no evidence that, when implemented, their solution will accomplish the cycle time. In this work, the simulation tests the line balancing, including variability not only in the processing time but also in the inter-arrival time, number of workers, and the speed of the material handling system by placing the line in a layout. The suggested three-part approach is connected to both the requirement for stability essential to handle uncertainty effectively, as pointed out in [3], as well as the extension of the ALB issue with non-deterministic parameters, as described in [6]. A gearbox case in [31], which required a cycle time of 70 s, is used to demonstrate the methodology.

2. Materials and Methods

2.1. Mixed-Integer Programming (MIP) Model

The SALB–1 problem is represented by a graph $G = \{V, E\}$ in which the set of vertices $V = \{1, \ldots, n\}$ represents the tasks, indexed as $i = 1, \ldots, n$. The set of edges $E = \{(i, j)\}$ stands for the precedence relations; thus, the edge (i, j) means that task i must be processed before task j. There is also a set of cells $K = \{1, \ldots, m\}$ indexed as $k = 1, \ldots, m$. Notice that n and m represent the maximum number of tasks and stations, respectively. Meanwhile, C is the required cycle time, and t_i is the processing time of task k.

The following two decision variables are defined:

$$x_{ik} = \begin{cases} 1, \text{ if task } i \text{ is assigned to station } k, \\ 0, \text{ otherwise.} \end{cases}, \qquad y_k = \begin{cases} 1, \text{ if station } k \text{ is open,} \\ 0, \text{ otherwise.} \end{cases}$$

the mixed-integer programming model that returns the minimum number of cells to satisfy the cycle time is:

$$\min \quad \sum_{k=1}^{m} y_k \tag{1a}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} t_i x_{ik} \leq C y_k, \qquad \forall \quad k \in K, \tag{1b}$$

$$\sum_{k=1}^{m} x_{ik} = 1, \qquad \forall \quad i \in V, \tag{1c}$$

$$\sum_{k=1}^{m} k x_{ik} \leq \sum_{k=1}^{m} k x_{jk}, \qquad \forall \quad (i, j) \in E, \tag{1d}$$

$$y_{k+1} \leq y_k, \qquad k = 1, 2, \ldots, m-1, \tag{1e}$$

$$x_{ik} \in (0, 1) \; \forall \, i, k, \quad y_k \in (0, 1) \; \forall \, k \tag{1f}$$

where the objective function that minimises the number of open cells is Equations (1a) and (1b) ensures that the cycle time C is not exceeded by the time content of a task allocated to an open cell. Equation (1c) ensures that a task is only assigned to one cell; meanwhile, Equation (1d) guarantees that given the precedence relationship $(i, j) \in E$ and given that the task i is assigned to a cell k, the task j is allocated in the same cell k or a further one $k + 1, k + 2, \ldots$; e.g., if task i is assigned to cell $k = 3$ (i.e., $x_{i3} = 1$); then

$3x_{i3} \leq 3x_{j3} + 4x_{j4} + \cdots + mx_{jm}$. Although $(i, j) \in E$, operations i and j can be allocated in the same cell, but i must first be processed. This is not achieved by the formulation, but it must be carried out when the cells are being set in the physical shop (layout). Equation (1d) assures that if the task i is assigned to cell k, the successor operations j can be in cells $k, k+1, k+2, \ldots$. Equation (1e) assures that only k cells are opened; e.g., if only two cells must be open, then $y_{2+1} = y_3 = 0$. Therefore, if $y_4 \leq y_3$, the only possible value is $y_4 = 0$, until $y_m \leq y_{m-1}$. Finally, Equation (1f) sets the decision variables as binary. The solution of the model in Equation (1) is a number of sets S_k of tasks i assigned to a cell k, as shown in Equation (2)

$$S_k = \{i \in V \mid x_{ik} = 1\} \quad \forall \quad k = 1, \ldots, m \tag{2}$$

the cell time (T_k) is the sum of the processing times of the tasks assigned to cell k; it is called the cell time content, as defined in Equation (3).

$$T_k = \sum_{i \in S_k} t_i \quad \forall \quad k = 1, \ldots, m \tag{3}$$

Finally, the efficiency of a cell (E_k) is computed as the sum of the processing times assigned to the cell divided by the cycle time, as shown in Equation (4).

$$E_k = \left(\frac{T_k}{C}\right) \times 100 \quad \forall \quad k = 1, \ldots, m \tag{4}$$

In the SALB-1 problem, the maximum number of cells m can be computed by dividing the sum of all tasks' processing time by the cycle time, rounded to the next nearest integer, as shown in Equation (5).

$$m = \left\lceil \frac{\sum_{i \in V} t_i}{C} \right\rceil \tag{5}$$

By setting the maximum number of open cells as in Equation (5), the objective function is rewritten using the finite set $K = \{1, 2, \ldots, m\}$ as min $\sum_{k \in K} y_k$ instead of using infinite indexes $k = 1, 2, \ldots$. As m cells are open; then, Equation (1b) is $\sum_{i=1} t_i x_{ik} \leq C$, throwing the term $\leq C y_k$, given that $y_1 = y_2 = \cdots = y_m = 1$, and Equation (1e) can be deleted. Therefore, the problem modelling is the same as in [7]. In our implementation [32], the user can set m to any number, and the proposed implementation can solve the problem.

2.2. Proposed Algorithm

The following notation indicates whether a parameter is deterministic or stochastic when defining the proposed three-part algorithm.

- $\mathbf{T}_k$ is the cell time content computed by simulation and T_k is defined in Equation (3).
- $\mathbf{E}_k$ is the cell efficiency computed by simulation and E_k is defined in Equation (4).
- $\mathbf{p}_r$ is the rth stochastic parameter under study and p_r is the rth deterministic parameter; e.g., p_r = the deterministic inter-arrival time or $\mathbf{p}_r$ = is the inter–arrival time $\sim Exp(\frac{1}{\lambda})$. Another example is p_r = the deterministic processing time or $\mathbf{p}_r$ = is stochastic processing time $\sim Exp(t_i)$.
- $P = \{\mathbf{p}_1, p_2, p_3, \mathbf{p}_4, \ldots\}$ is the set of all parameters of interest, both deterministic and stochastic, in no particular order.

The suggested method is shown in Algorithm 1, which requires the graph G representing the product's structure to be assembled, the processing times, and the desired cycle time. As a result, the output is the cells to be deployed and the values of the stochastic parameters to obtain the desired cycle time.

Algorithm 1 has three steps: (a) computing the number of cells using the MIP model, whereby each task has been assigned to a cell; (b) experimenting with the MIP model's

solution in a simulation model by including stochasticity; and (c) optimising the value of the stochastic parameters so that the user can obtain the required cycle time in each cell.

Algorithm 1: Solving the stochastic SALB–1 Problem.

Data: A graph $G = \{V, E\}$, C, and $t_i \; \forall \; i \in V$
Result: cells S_k (Equation (2), the values of the parameters set P
`/* Section 2.2.1 Cells formation */`
1 compute m (Equation (5);
2 set the cycle time C;
3 solve the model in Equation (1) using GLPK via Pyomo;
4 get the S_k (Equation (2), the T_k (Equation (3), and the E_k (Equation (4) for every cell;
`/* Section 2.2.2 Build the simulation model and verify it */`
5 define the deterministic parameters set of interest, $P = \{p_1, \ldots, p_r, \ldots\}$;
6 build the simulation model placing, in a scaled layout, the distribution of the S_k;
7 input all model information to the simulation model;
8 create `Controls` in SIMIO for the $\mathbf{T}_k$ and $\mathbf{E}_k$;
9 run the simulation model;
10 **if** $\mathbf{T}_k \approx T_k \wedge \mathbf{E}_k \approx E_k$ **then**
11 go to 14; `// model has been verified`
12 **else**
13 go to 7;
14 **foreach** $p_r \in P$ **do**
15 set $\mathbf{p}_r$;
16 run the simulation model with $P = \{p_1, \ldots, \mathbf{p}_r, p_{r+1}, \ldots\}$;
17 record the results of $\mathbf{p}_r$;
`/* Section 2.2.3 Optimising the Value of the Stochastic Parameters */`
18 duplicate the simulation model to solve it by OptQuest in SIMIO;
19 select the `Controls` to minimise and/or maximise;
20 according to the values in 17, constrain the parameters $\mathbf{p}_r$;
21 define the constraint $\mathbf{T}_k \leq C \; \forall \; k$;
22 report the solution $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_r, \ldots\}$

2.2.1. Cells Formation Achieved by Solving the MIP Model

The suggested method calculates the number of cells and the task to be carried out in each one. To achieve this, the maximum number of cells m is computed along with the cycle time C (lines 1–2). In line 3, the Python implementation solves the model; an excerpt from the source code is shown in Listing A1 in Appendix A. This requires inputting the data as a `DiGraph` using the Python package `NetworkX` [33].

GNU Linear Programming Kit (GLPK) is used to solve the MIP model in Equation (1), and it is implemented with Pyomo language. GLPK consists of a collection of ANSI C routines arranged as a callable library under the GNU General Public License [34]. Meanwhile, Pyomo is an open-source Python-based modelling language for mathematical optimisation. It provides a flexible framework for formulating and solving optimisation problems, allowing users to express complex mathematical models in a concise and intuitive manner [35].

The line 35 of Listing A1 is used to solving the MIP model in Equation (1). The user must input "glpk" to the function `SolverFactory` and the name of the model to be solved in this case "model" as in Listing 1.

Listing 1. Call `SolverFactory` in Pyomo

```
results = pyo.SolverFactory('glpk').solve(model)
```

Finally, the implementation returns a report for every cell, similar to the following one:

Cell k produces the tasks: $i, \ldots$ (i.e., S_k).
The time content in cell k is $\ldots$ (i.e., T_k).
The efficiency of cell k is $\ldots$ (i.e., E_k).

2.2.2. Build the Simulation Model

This part has two important objectives: (a) to verify the model by comparing the values of the T_k and E_k with the ones computed by the simulation model, and (b) to create scenarios to set a parameter as stochastic and the other as deterministic to find (if possible) the value of the parameter that returns the desired cycle time.

In order to accomplish this, the user must specify the parameters of interest (line 5 in Algorithm 1), such as the number of workers in a cell or the value of the inter-arrival time.

The cells' physical distribution is a key aspect of the method; therefore, the proposed approach considers the distances among the machines, and the distance a product must travel to reach the next station or cell (line 6).

In line 7, all the information required for the model is input. The following data must be set in the SIMIO model to verify it (independently of the parameters of interest defined by the user): (a) a task is performed in a SIMIO object called `server`; (b) each `server` cannot store any kind of buffer; (c) every cell k has a SIMIO object called `worker` that performs all the tasks in the cell, one at a time; (d) the `worker` transports the processing units from station to station, and travels the physical distance in no time (i.e., instantaneously); (e) all the processing times are deterministic (t_i); and (f) the inter-arrival time is deterministic with $\frac{1}{\lambda} = C$.

To record the value of the $\mathbf{T}_k$ and $\mathbf{E}_k$, SIMIO uses `Controls`. A `Control` in SIMIO software simulation is a variable that can be changed to test the effects on the simulation output [36].

Once the model is run, the user has to verify the model by comparing the simulation results with those computed when solving the MIP model (lines 9–13). If those values are not similar, the user has to check the input information and the entire model, given that an unverified model returns untrusted results.

Finally, after the model has been validated, some scenarios are produced by making one parameter stochastic $\mathbf{p}_k$ and the other deterministic $p_{k'}$ (lines 14–17).

2.2.3. Optimising the Value of the Stochastic Parameters

To carry out this section, the model created in Section 2.2.2 is duplicated; thus, the `Control` is retained (line 18).

This section requires us to create a model in which one or more `Controls` are minimised and/or maximised under a set of constraints, where the default constraint is that the cell time content must be less than or equal to the cycle time $\mathbf{T}_k \leq C \; \forall \; k \in K$. The decision variables are the stochastic parameters whose values are within the range in Equation (6) (lines 19–21):

$$\mathbf{p}_r = \alpha_r, \alpha_r + \Delta, \alpha_r + 2\Delta, \alpha_r + 3\Delta, \ldots, \beta_r \tag{6}$$

this is represented in the model as $\mathbf{p}_r \in (\alpha_r, \beta_r) + \Delta_r$. To determine the range of the parameter intervals, the results in line 17 can be used. The general model to be solved is depicted below:

Maximise/Minimise ($\texttt{Control}_1, \texttt{Control}_2, \ldots$)
Subject to
$\mathbf{T}_k \leq C \quad \forall \quad k \in K$

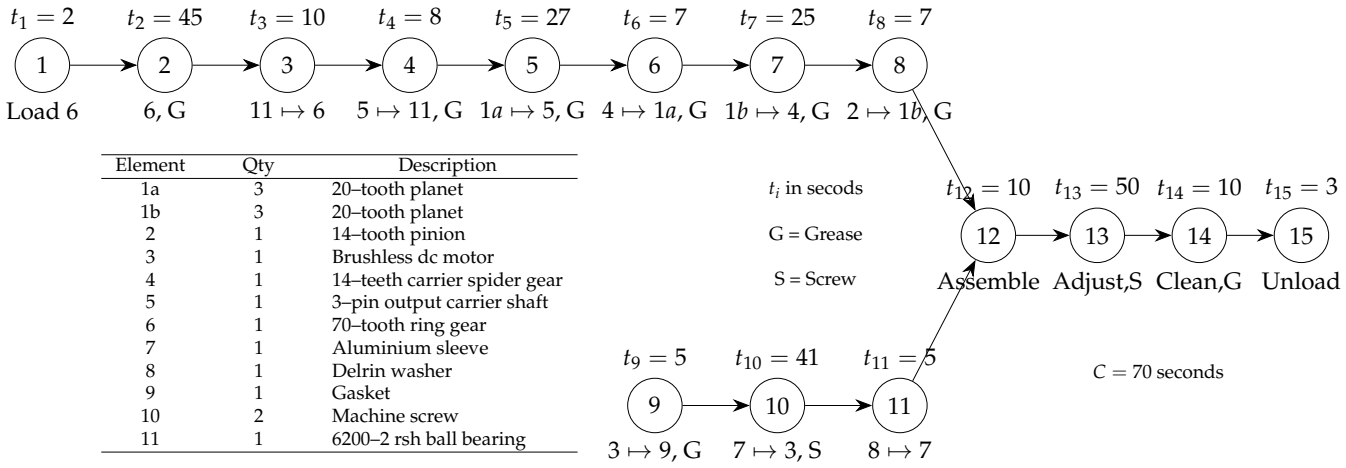
$$\vdots$$

$$\mathbf{p}_r \in (\alpha_r, \beta_r) + \Delta_r \; \forall \; r \in P$$

The solution is the value of the stochastic parameters as $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_r, \ldots\}$ (line 22), which guarantee that the time of each cell is less than or equal to the required cycle time.

2.3. Instance to Be Solved

Algorithm 1 is tested using a two–stage gearbox that appears in [31]. Figure 1 shows the graph $G = \{V, E\}$ with $V = \{1, 2, \ldots, 15\}$ (the maximum number of tasks is $n = 15$) and $E = \{(1,2), \ldots, (11,12), \ldots, (14,15)\}$ and a table with the description of each element (or raw material) and its quantities.



Element	Qty	Description
1a	3	20–tooth planet
1b	3	20–tooth planet
2	1	14–tooth pinion
3	1	Brushless dc motor
4	1	14–teeth carrier spider gear
5	1	3–pin output carrier shaft
6	1	70–tooth ring gear
7	1	Aluminium sleeve
8	1	Delrin washer
9	1	Gasket
10	2	Machine screw
11	1	6200–2 rsh ball bearing

Figure 1. Two-stage gearbox instance to be solved.

The upper label in each task (node) is the task's process time t_i in seconds, and the lower label is the activity that takes place in it. For example, the processing time of task (node) 12 in which an assembly is produced is $t_{12} = 10$ s. The worker in task 1 loads a 70-tooth ring gear in $t_1 = 2$ seconds. The 70-tooth ring gear is greased in $t_2 = 45$ s in task 2. Some nodes have a label below, such as the one in task 10 ($7 \mapsto 3$, S), which means that element 7 (aluminium sleeve) is screwed (S) inside element 3 (brushless DC motor). In another example, in task 5, 3 20–tooth planets (1a) are placed inside a three-pin output carrier shaft and then the assembly is greased (G). The required cycle time is $C = 70$ s.

The cells resulting from solving the MIP model will be placed over a 20×10 m^2 shop layout. Taking this into account is important because if workstations are located far apart, this may result in excessive material handling or movement time between stations, leading to inefficiencies and delays. As a result, the system will never achieve the required cycle time.

Table 1 shows the four parameters of interest (Algorithm 1 line 5) used to include variability in the resulting balancing.

Table 1. Parameter of interest to solve the instance in Figure 1.

r	**1**	**2**	**3**	**4**
Parameter	**Inter–Arrival Time**	**Speed of Workers**	**Number of Workers**	**Process Time**
Discrete value (p_r)	$p_1 = \frac{1}{\lambda}$	$p_2 = s$	$p_3 = w$	$p_4 = t_i$
Stochastic value ($\mathbf{p}_r$)	$\mathbf{p}_1 = Exp\left(\frac{1}{\lambda}\right)$	$\mathbf{p}_2 = (\alpha_2, \beta_2) + \Delta_2$	$\mathbf{p}_3 = (\alpha_3, \beta_3) + \Delta_3$	$\mathbf{p}_4 = Exp(t_i)$

1. Inter-arrival time (IAT) [s] is the time that passes between subsequent raw-material entities entering the first cell. To add variability to this parameter, it is assumed that IAT $\sim Exp\left(\frac{1}{\lambda}\right)$.
2. Speed of workers is the velocity [m/s] at which a worker is able to move inside the cell. A worker must walk inside the cell to work on each task. The worker's speed increases to see if the cell time content decreases.
3. Number of workers in each cell. There is just one worker to validate the model and this number is increased to reduce the cell time content. The workers move the work-in-progress (entities or pieces) among the workstation and transport them from station to station.
4. Process time t_i [s] refers to the duration of a task to be completed within the cell. The processing time is $\sim Exp(t_i)$.

To test the behaviour of the system, some experiments are created based on the nature of the parameters, e.g., the experiment u is run using $P_u = \{p_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4\}$, which means that the IAT is deterministic $p_1 = \frac{1}{\lambda}$ while the speed, the number of workers, and the processing time are stochastic; thus, $\mathbf{p}_2 = (\alpha_2, \beta_2) + \Delta_2$, $\mathbf{p}_3 = (\alpha_3, \beta_3) + \Delta_3$, and $\mathbf{p}_4 = Exp(t_i)$, respectively. Another example is $P_n = \{\mathbf{p}_1, \mathbf{p}_2, p_3, \mathbf{p}_4\}$; this means that the number of workers is deterministic, meanwhile, the others are stochastic; see Table 1 for the values of all the parameters.

3. Results

The findings of this work show that when all parameters except the inter-arrival time are stochastic, the cycle time is 68 s. The cycle time is around 72.04 s if all the parameters are stochastic. Regarding the three-part methodology, it is shown that the solution produced by the mathematical model cannot obtain a cycle time of less than or equal to 70 s in the presence of variability. Therefore, the simulation model finds the value of the stochastic parameters, resulting in a very close 70 s cycle time.

The detailed results of solving the instance depicted in Section 2.3 are shown below. In the first part of the proposed approach, the MIP model in Equation (1) is solved to determine the assignment of the task among the cells. The maximum number of cells is $m = \lceil 255/70 \rceil = 4$ (Equation (5). A summary of the results of lines 1–4 of Algorithm 1 is shown in Table 2. The parameters of interest are summarised in Table 1 (see line 5), the shop layout is plotted in Figrue 2 as required in line 6 of Algorithm 1, and the results of the model verification are presented in Table 3 lines 7–13.

Table 2. Results of the MIP model in Equation (1).

Cell k	Task Assigned Equation (2) S_k	Time Content Equation (3) T_k (s)	Cell Efficiency Equation (4) E_k (%)
1	$\{1,2,3,4\}$	65	92.86
2	$\{5,6,7,9\}$	64	91.43
3	$\{8,10,11,12\}$	63	90.00
4	$\{13,14,15\}$	63	90.00

In the second part of the proposed approach (Algorithm 1 lines 14–17), five experiments measure the effect of several stochastic parameters on the cycle time. The first one is used to verify the simulation model; thus, all the parameters are deterministic. In the second one, the effect of the inter-arrival time is measured by running the experiment with stochastic IAT with the rest of the parameters being deterministic. Experiments 3, 4, and 5 are run, setting the inter-arrival time as deterministic and stochastic as well as the stochastic processing time. In experiment 3, the IAT's value is between 50 and 80 s and four workers' speed at 2 m/s. The effect of two workers' speed is measured in experiment 4; it ranges

from 2 to 12 m/s, and the inter-arrival time is equal to 70 s. Finally, experiment 5 is run using a different number of workers in a cell when the IAT is 90 s, and their speed is 4 m/s.

The experiment $P_1 = \{p_1, p_2, p_3, p_4\} = \{70, \infty, 1, t_i\}$ is set to verify the model, i.e., the raw materials arrive at the cell at a pace of $\frac{1}{\lambda} = 70$ seconds, the workers' speed is high to avoid delays in travelling, there is one worker in each cell, and the processing times are deterministic for all tasks. The physical locations of the four cells in the shop layout are depicted in Figure 2. The simulation model is run for 200 h with a 20 h warm-up period. The output of Algorithm 1 from lines 5–13 is shown in Table 3. As expected, the cells' cycle time (C) is less than 70 s $(T_k \leq 70)$. As shown, the model is verified given that there is a percentage difference of less than 0.5% of the time contents and efficiency for all cells.

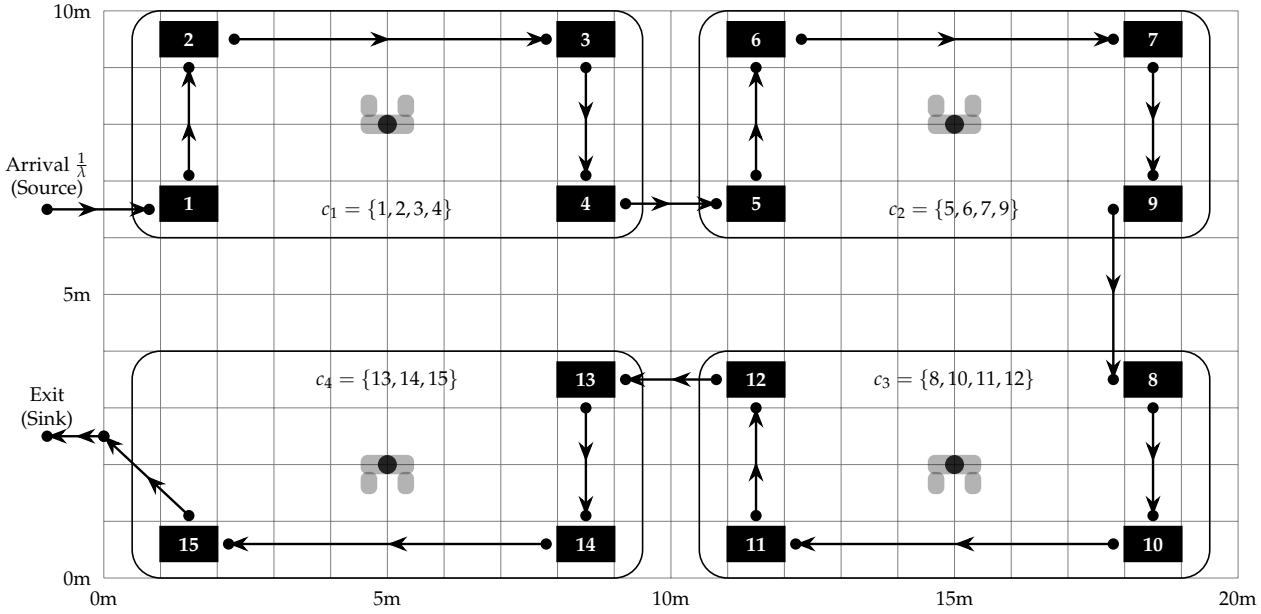


Figure 2. 10×20 m^2 shop layout to place the cells.

Table 3. Results of the simulation model with $P_1 = \{p_1, p_2, p_3, p_4\} = \{70, \infty, 1, t_i\}$.

Cell k	Time Content T_k (s)	Percentage Difference (%) [1]	Cell Efficiency E_k (%)	Percentage Difference (%) [2]
1	65.11	0.17	93.09	0.25
2	64.11	0.17	91.66	0.25
3	63.12	0.19	90.21	0.23
4	63.08	0.13	90.18	0.20

[1] $[(\mathbf{T}_k - T_k)/T_k] \times 100$ (see Table 2); [2] $[(\mathbf{E}_k - E_k)/E_k] \times 100$ (see Table 2).

To compute the results in Table 3, it is supposed that all the parameters of interest are deterministic. In a second experiment, the inter-arrival time is stochastic with IAT $\sim Exp(\frac{1}{\lambda})$, and the rest of the parameters are deterministic; thus, $P_2 = \{\mathbf{p}_1, p_2, p_3, p_4\} = \left\{Exp(\frac{1}{\lambda}), \infty, 1, t_i\right\}$. Since the cycle time duration exceeds the maximum of 70 s for all cells, as shown by the results in Figure 3, a variation in the inter-arrival time prevents on-time product delivery when deterministic balancing is used during the real process. As a result, the cycle time decreases as the inter-arrival time increases. The minimum cell time contents are $\mathbf{T}_1 = 97$, $\mathbf{T}_2 = 93$, $\mathbf{T}_3 = 104$, and $\mathbf{T}_4 = 100$ s when the IAT $\sim Exp(\frac{1}{80})$, but the efficiency is very low for the four cells; the highest one is 66.4% for cell 1, $E_1 = 66.4$. Figure 3 shows a clear correlation between cycle time and efficiency; the shorter the cycle time, the lower the efficiency. For this particular instance, the time content of cell 2 appears constant

(from 90 to 93 s) as the inter-arrival time increases; in addition, the efficiency in the four cells declines at a consistent rate as the inter-arrival time rises by increments of 5 s.

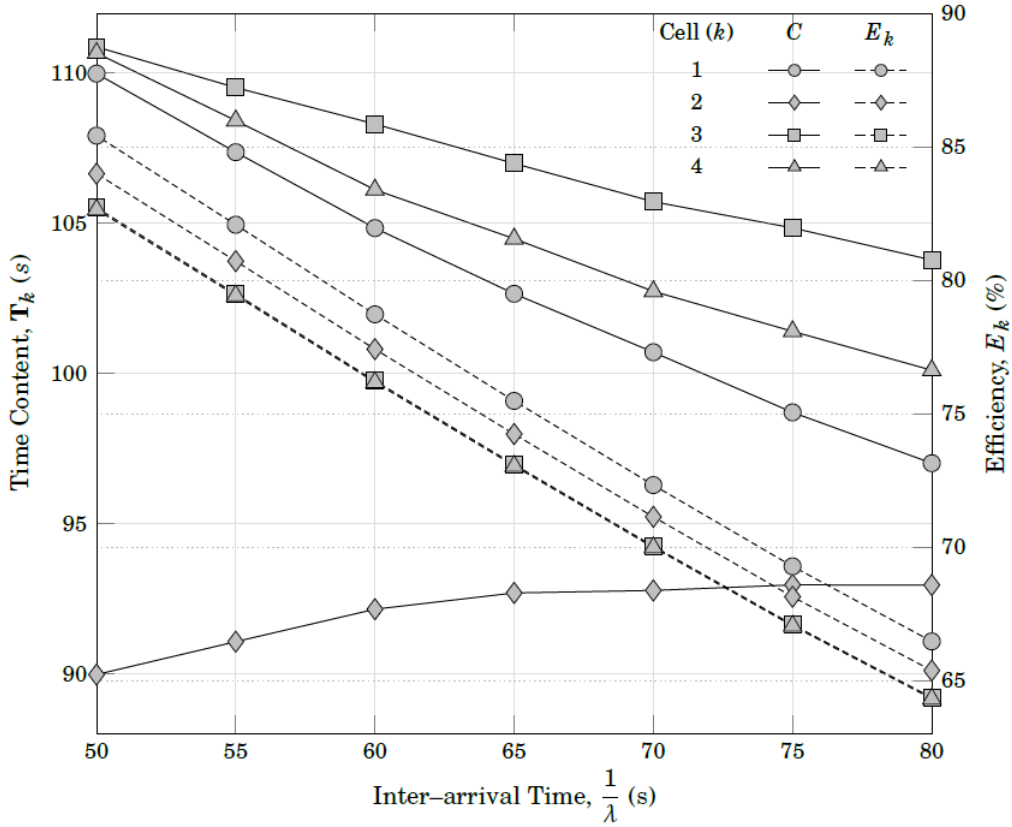
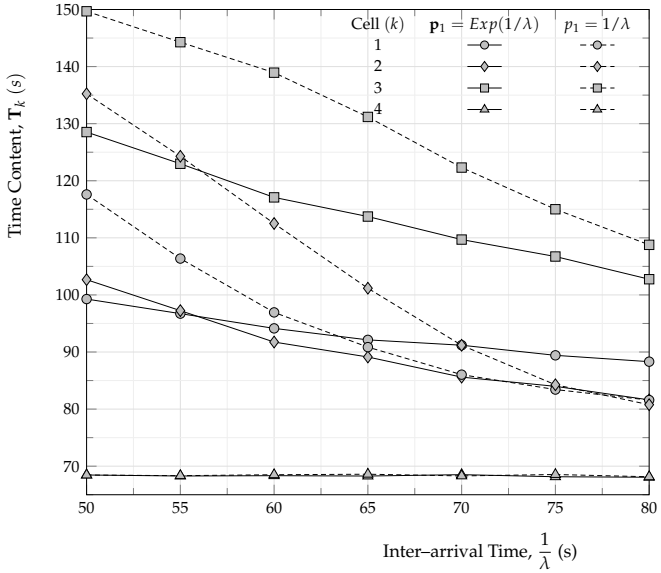


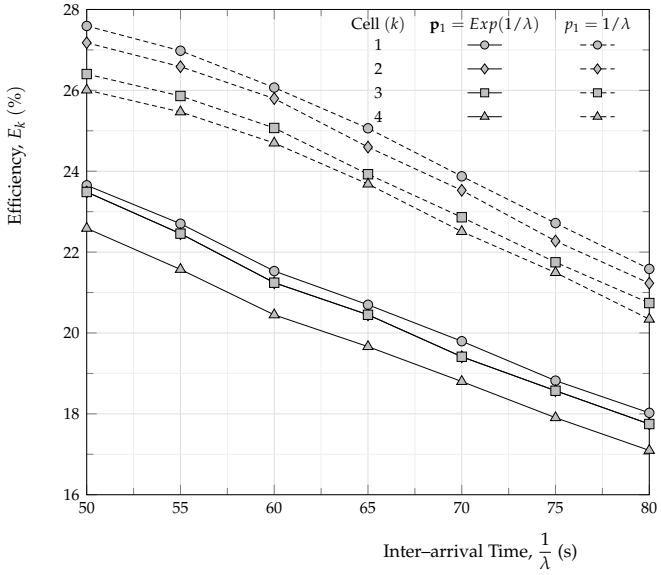
Figure 3. Experiment $P_2 = (\mathbf{p}_1, p_2, p_3, p_4) = \left(Exp\left(\frac{1}{\lambda}\right), \infty, 1, t_i\right)$.

The third experiment is run when the IAT is $\mathbf{p}_1 = Exp(\frac{1}{\lambda})$ and $p_1 = \frac{1}{\lambda}$, solid and dotted lines in Figure 4, respectively. The rest of the parameters are $p_2 = 2$ m/s, $p_3 = 4$ workers, and the process times stochastic $\mathbf{p}_4 = Exp(t_i)$. As in Figure 4b, the cells' time content and efficiency decrease with increments in the inter-arrival time (see Figure 4). As shown, when four workers are speeding at 2 m/s, the cells' time content is lower in the presence of stochastic inter-arrival time than when the IAT is deterministic. The efficiency is lower in the presence of stochastic IAT; see Figure 4b. Thus, the objective of maximum utilisation is reached by p_1, but the cells' time content $(\mathbf{T}_k)$ is less than 70 s when $\mathbf{p}_1$. In this experiment, the minimum time contents are observed when IAT$\sim Exp(\frac{1}{80})$ seconds; those are $\mathbf{T}_1 = 88$, $\mathbf{T}_2 = 82$, $\mathbf{T}_3 = 103$, and $\mathbf{T}_4 = 68$ s. Only cell 4 satisfies the required cycle time of 70 s, and the maximum efficiency is 27.5% $(E_1 = 27.5)$ when $p_1 = 50$ s. Unfortunately, experiment P_3 fails to meet both the necessary cycle time and the decision-maker's aim of decreasing the cycle time and maximising usage.

Experiment 4 (P_4) assesses the effect of the workers' velocity on the system. The speed is changed from 2 to 12 m/s with increments of 2. Figure 5a shows that while the speed increases, the value of the cells' time content is almost constant, e.g., the time content of cell 3 is 134 s with 2 m/s, while it is 115 with 12 m/s when $p_1 = 70$ s. Generally, if the workers' speed increases, the time content $(\mathbf{T}_k)$ should decrease. In the instance solved, the maximum decrement in cycle time is about 10% for cell 2 when comparing between 2 and 12 m/s using $p_1 = 70$ s. Concerning the efficiency (Figure 5b), it decreases fast between 2 and 4 m/s; after 4 m/s, the velocity decreases slowly; moreover, higher efficiency is observed when the interarrival is deterministic.

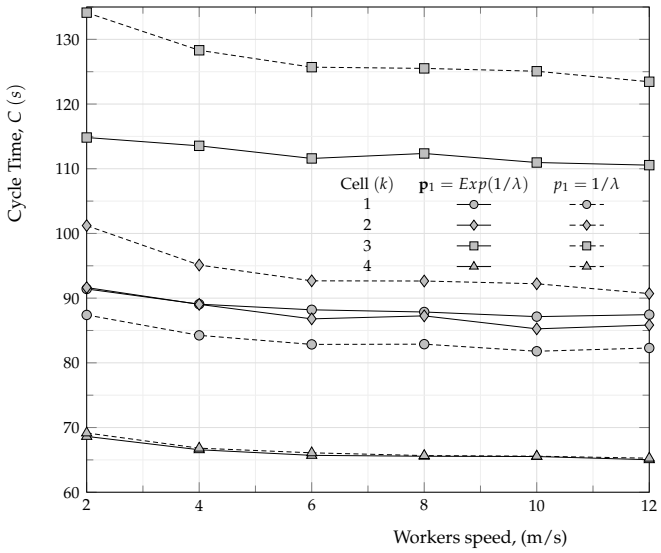


(**a**) Cycle time (C) when $\mathbf{p_1}$ and p_1

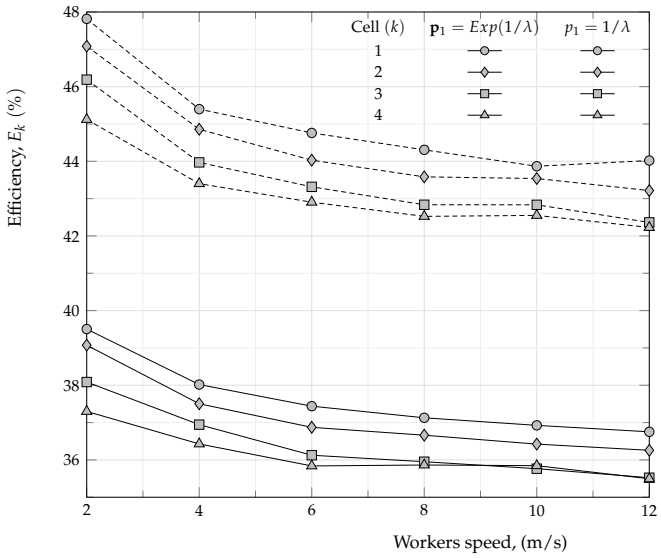


(**b**) Efficiency (E_k) when $\mathbf{p}_1$ and p_1

Figure 4. Effect of the inter-arrival time, $P_3 = \{[\mathbf{p}_1, p_1], p_2, p_3, \mathbf{p}_4\} = \left\{[Exp(\frac{1}{\lambda}), \frac{1}{\lambda}], 2, 4, Exp(t_i)\right\}$.



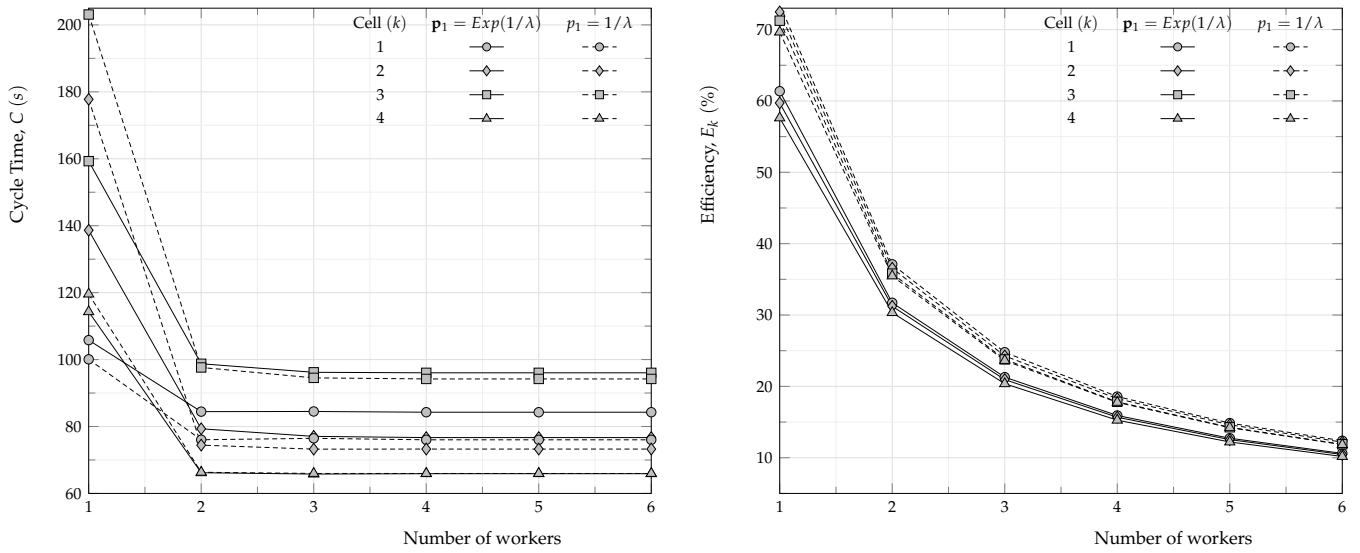
(**a**) Cycle time (C) when $\mathbf{p}_2 \in (2, 12) + 2$



(**b**) Efficiency (E_k) when $\mathbf{p}_2 \in (2, 12) + 2$

Figure 5. Effect of the workers speed, $P_4 = \{[\mathbf{p}_1, p_1], \mathbf{p}_2, p_3, \mathbf{p}_4\} = \{[Exp(70), 70], \mathbf{p}_2, 2, Exp(t_i)\}$.

Finally, in experiment five P_5, the number of workers varies from 1 to 6. From previous experiments, it looks like a long inter–arrival time and speed of 4 m/s returns short time contents; thus, those values are used in this experiment (P_5). The results in Figure 6 clearly show that using more than two workers in each cell returns the same time contents for all cells (Figure 6a), and the efficiency decreases (Figure 6b) when more workers are assigned to a cell. Note that the number of workers is exactly the same in each cell; e.g., if the number of workers is two, it means that two workers work in each open cell. Furthermore, if the speed is set to 4 m/s, every worker will have the same speed.



(**a**) Cycle time (C) when $\mathbf{p}_3 \in (1, 6) + 2$

(**b**) Efficiency (E_k) when $\mathbf{p}_3 \in (1, 6) + 2$

Figure 6. Effect of the number of workers, $P_5 = \{[\mathbf{p}_1, p_1], p_2, \mathbf{p}_3, \mathbf{p}_4\} = \{[Exp(90), 90], 4, \mathbf{p}_3, Exp(t_i)\}$.

In summary, when the IAT$\sim Exp(\frac{1}{\lambda})$ is the only stochastic parameter, the time content and the efficiency are minimal with long $\frac{1}{\lambda}$ (see Figure 3). From experiments 3 to 5 (Figures 4–6), it seems that the shortest cells' time content is observed when the inter-arrival time is stochastic. Still, the maximum efficiency is computed using deterministic IAT. None of the experiments return cells time content of 70 s or less. The time content of cell 4 reaches less than 70 s in experiments 3, 4, and 5.

The lowest cycle time for the five experiments is listed below:

- $P_1 = \{70, \infty, 1, t_i\}$ is $C_1 = \max_k\{65.11, 64.11, 63.12, 63.08\} = 65.11$ s which satisfy the required 70 s.
- $P_2 = \{Exp(80), \infty, 1, t_i\}$ is $C_2 = \max_k\{97.01, 92.95, 103.78, 100.11\} = 103.78$ s. It is longer than 70 s.
- $P_3 = \{Exp(80), 2, 4, Exp(t_i)\}$ returns $C_3 = \max_k\{88.31, 81.61, 102.75, 68.06\} = 102.75$ s which is not satisfactory.
- $P_4 = \{Exp(70), 12, 2, Exp(t_i)\}$ is $C_4 = \max_k\{87.45, 85.84, 110.56, 65.03\} = 110.56$ s, longer than 70 s.
- $P_5 = \{Exp(90), 4, 3, Exp(t_i)\}$ is $C_5 = \max_k\{84.49, 77.05, 96.19, 65.79\} = 96.19$ s, which does not satisfy the required 70 s.

In conclusion, the cells return time content of 70 s or less without stochastic parameters. Real-world applications of industrial processes include intrinsic variability; as a result, strategies for process engineering, like lean manufacturing, try to reduce it because total eradication is essentially unachievable [37]. Therefore, experiment P_1 is not achievable, and the other experiments (with stochastic factors) do not satisfy the constraint of a 70 s cycle time.

SIMIO OptQuest is used to find a likely scenario and search for a feasible solution that provides a cycle time of 70 s or less. OptQuest for SIMIO simulation software is an add-on module that provides strong optimisation capabilities (see [38]). It connects with SIMIO's simulation models to maximise or minimise one or more objectives while maximising or minimising the value of specific variables.

Two models in the third part of the proposed algorithm (Algorithm 1 lines 18–22) are implemented. In the first one, the efficiencies of the cells are maximised (Equation (7a)) given that the cells' time content is less than the required cycle time (Equation (7b)) using the following parameters: stochastic IAT between 240 and 300 s (Equation (7c)); all the

process times are stochastic (Equation (7d)); and the workers' speed ranges from 2 to 10 m/s and from 6 to 9 workers (Equation (7e)).

$$\max \quad (E_1, E_2, E_3, E_4) \tag{7a}$$

$$\text{s.t.} \quad \mathbf{T}_k \leq 70 \text{ s} \qquad \forall \quad k \in K, \tag{7b}$$

$$\mathbf{p}_1 = Exp\left(\frac{1}{\lambda}\right) \qquad \frac{1}{\lambda} \in (240, 300) + 10, \tag{7c}$$

$$\mathbf{p}_4 = Exp(t_i) \qquad \forall \quad i \in V, \tag{7d}$$

$$\mathbf{p}_2 \in (2, 10) + 2, \quad \mathbf{p}_3 \in (6, 9) + 1 \tag{7e}$$

The second implemented model is similar to the one in Equation (7), but the constraint in Equation (7c) is relaxed, as shown in Equation (8).

$$p_1 = \frac{1}{\lambda}, \quad \frac{1}{\lambda} \in (120, 180) + 5 \tag{8}$$

OptQuest finds a solution for model 1 with a cycle time equal to $C_{\text{model 1}} = 72.04$ s, as shown in Table 4, only two seconds above the required 70. To generate a 72 s cycle time, nine people speeding at 10 m/s must be placed in each cell, and IAT must be $Exp(300)$ s (i.e., an arrival every $Exp(5)$ min). Model 2 is model 1 with a constant IAT; thus, the cycle time of model 2 is $C_{\text{model 2}} = 68.92$ s, which is less than 70 s. As a result, the IAT time must not exhibit variability if the decision-maker intends to deliver items on time.

Table 4. Solution of the two implemented models using OptQuest .

Model	Efficiency (E_k) and Time Content (T_k) of Cell k								Parameters [a]		
	E_1	E_2	E_3	E_4	$\mathbf{T}_1$	$\mathbf{T}_2$	$\mathbf{T}_3$	$\mathbf{T}_4$	$\mathbf{p}_1$	$\mathbf{p}_2$	$\mathbf{p}_3$
1 [b]	2.4	2.36	2.32	2.33	72.04	67.4	70.45	64.56	300 [c]	10	9
2 [b]	6.71	6.62	6.50	6.51	67.22	66.05	68.62	64.39	165 [c]	6	8

[a] $\mathbf{p}_4 = Exp(t_i)$ for model 1 and 2. [b] Model 1 is Equation (7) and model 2 replaces Equation (7c) by Equation (8) in model 1. [c] $\mathbf{p}_1 = Exp(165)$ s to solve model 1 and $p_1 = 165$ s to solve model 2.

4. Discussion

This work investigates the effect of variability in the simple assembly line balancing the (SALB–1) problem, in which the cycle time is known and the number of cells and the tasks' assignations are known.

In this work, a three-part algorithm (Algorithm 1) is proposed in which commercial solvers solve a MIP model; then, a set of parameters of interest is chosen according to the decision-maker to experiment with the values of each parameter; finally, OptQuest is used to optimise some controls, constraining the cells' time content to the desired cycle time.

The proposed approach differs from those in [7,8] because it considers parameter variability; furthermore, variables (such as a physical shop layout, in which one or more workers speed to a certain velocity inside the cell) are taken into account; additionally, stochastic process times are also considered. Some works, such as [15], consider variability in the tasks' process time, and the solution method is constraint programming, but variables such as the inter-arrival time or others considered in this work are not taken into account.

The proposed method is much more flexible than those based on tailor- and meta-heuristic or MIP models and constraint programming (see [6–8,15,39]) because the decision–maker can set any number of stochastic parameters; moreover, he or she can test the resulting balancing on a scaled shop layout.

According to the results, the real-world instance is solved in no time [31] by implementing the MIP model in Pyomo and solving it using the GLPK solver. In the MIP model, all the parameters are deterministic; thus, a simulation model is developed in SIMIO software to include variability in the problem. Four stochastic parameters are set: IAT, workers'

speed, number of workers, and tasks' process times. Some experiments are created to find cycle times with less than 70 s (the required cycle time). Finally, the add-on OptQuest in SIMIO is utilised to locate cells' time contents with less than 70 s in the presence of variability in all the parameters.

In Table 2, the solution of the MIP model is presented; therefore, the SALB-1 problem is solved with deterministic parameters. Using the stochastic parameters shown in Table 1, five experiments were set to find the values of the parameters that return the desired cycle time. The results of experiment 1 (Table 3) were used to verify the model, comparing the results of the MIP model with those of the simulation model. After that, Experiment 2 included variability just in the IAT. If variability is observed in the IAT, the shortest time contents are obtained using a large IAT—in this case, about 80 s (Figure 3). From experiments 3 to 5 (Figures 4–6), it is concluded that in the presence of variability, shorter cycle times are obtained; meanwhile, the efficiency increases with the deterministic parameters.

Two models are solved in OptQuest (Equations (7) and (8)). As stated by the results (Table 4), the proposed approach can find a solution that results in a cycle time of 68.62 s when the IAT is deterministic and 72.04 s when the IAT is stochastic. According to this, it seems that when the inter-arrival parameter is constant, the desired cycle time is attainable in the solved instance, 70 s.

The results prove that when variability is observed, the results of the MIP model can achieve the required cycle time when implemented in a real shop layout. It is shown that using the add-on OptQuest in SIMIO, the desired solution is found. For the solved instance, the proposed approach finds a solution with 2.04 s more than the desired cycle time.

This work is limited to a 15-task instance with 14 precedence relationships and four parameters from which the IAT and the processing time fit an exponential distribution, and an interval limits the value of the other two. Although the MIP model balances the system, it is an open research area, as stated in [6], given the computational capacity nowadays and the state-of-the-art commercial solvers.

In future, the proposed approach can be tested using bigger problems and can be solved the SALB–2 problem and/or the General SALB problems. Another natural extension is to use a different set of parameters and different distribution functions.

5. Conclusions

In conclusion, this work addresses a simple line-balancing problem when the cycle time has already been determined (i.e., SALB-1 problem). Through simulations, it has been shown that the solution of the MIP model cannot deliver products on time, i.e., the cycle time is not satisfied. The three-part algorithm finds a solution in which, in the presence of variability, the system could deliver products on time (before the cycle time). Using the add-on OpQuest in SIMIO, the proposed approach finds a solution that satisfies the cycle time given some parameters of interest and variability.

Through extensive experimentation and analysis, it was observed that in the presence of variability, a long inter-arrival time is necessary, but this drastically decreases the cells' efficiency. Some scenarios are created to experiment with the values of a set of stochastic parameters. OptQuest determines the values of the stochastic parameter to satisfy the required cycle time in the presence of variability in the process.

The findings of this study have significant practical implications for industries that rely on assembly lines, particularly those seeking to improve productivity and reduce costs. By achieving a balanced production line in which the intrinsic variability is considered, the decision–maker delivers products on time; thus, the operation efficiency is enhanced.

In summary, this research contributes to the field of line balancing by presenting a comprehensive approach that combines mathematical programming models and simulation by first computing an optimal balancing line, then including variability via a set of stochastic parameters, and finally optimising the values of the parameters to satisfy the cycle time. The results obtained demonstrate the effectiveness of the proposed method in optimising line balance and provide a foundation for further advancements in stochastic balancing lines.

By addressing the line-balancing problem, industries can achieve improved productivity, streamlined operations, and increased competitiveness in the marketplace.

Author Contributions: Conceptualization, L.A.M.-M. and E.H.A.-N.; methodology, L.A.M.-M. and E.H.A.-N.; software, L.A.M.-M. and E.H.A.-N.; validation, L.A.M.-M. and E.H.A.-N.; data curation, L.A.M.-M.; writing—original draft preparation, L.A.M.-M. and E.H.A.-N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The reader can find the whole project at: https://github.com/LuisMoncayo/StochasticSALB1 (accessed on 1 June 2023).

Acknowledgments: Financial support from the Asociación Mexicana de Cultura (A.C.) is gratefully acknowledged. The authors sincerely appreciate the anonymous reviewers' insightful comments. The National Council of Science and Technology (CONACyT) from Mexico is acknowledged.

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

Abbreviations

The following abbreviations are used in this manuscript:

V	Set of tasks.
E	Set of precedence relationships.
K	Set of cells.
P_u	Set of parameters used in experiment. u
i	Task i.
(i, j)	Precedence between task i and task j.
k	Cell k.
m	Maximum number of cells.
n	Maximum number of tasks.
C	Cycle time.
t_i	Process time of task i.
S_k	Set of task i assigned to cell k.
T_k	Time content of the cell k.
$\mathbf{p}_r$	Stochastic parameter.
p_r	Deterministic parameter

Appendix A

Listing A1. Python implementation to solve the MIP model using GLPK.

```
model = pyo.ConcreteModel()

nu_cells = 4
cycle_time = 70
list_cells = list(range(1,nu_cells+1,1))

model.N = pyo.Set(initialize=list_nodes) # list of tasks (i.e., nodes); i=1,..,N
model.M = pyo.Set(initialize=list_cells) # list of cells; j=1,...,M
model.x = pyo.Var(model.N,model.M, domain=pyo.Binary)
model.y = pyo.Var(model.M, domain=pyo.Binary)

def objective_SALB1(model):
    return sum(model.y[i] for i in model.M)
model.obj = pyo.Objective(rule=objective_SALB1, sense=pyo.minimize)

def cycle_time_cell(model, j):
    return sum(process_time[i]*model.x[i,j] for i in model.N) <= \\ cycle_time*model.y[j]
model.cycle_cell = pyo.Constraint(model.M, rule=cycle_time_cell)

def task_assignation(model, i):
    return sum(model.x[i,j] for j in model.M) == 1
model.task_assig = pyo.Constraint(model.N, rule=task_assignation)

model.precedence_task = pyo.ConstraintList()
for edge in precedence_relationships:
    from_task = edge[0]
    to_task = edge[1]
    model.precedence_task.add(expr=sum(j*model.x[from_task,j]\\ for j in model.M) <=  sum(j*model.x[
    to_task,j] for j in model.M))

model.cells = pyo.ConstraintList()
for j in range(nu_cells-1):
        model.cells.add(expr=model.y[j+2] <= model.y[j+1])
#model.pprint()

results = pyo.SolverFactory('glpk').solve(model)

print("The minimum number of cells is ", pyo.value(model.obj) )

for j in range(nu_cells):
    cell_time = 0
    print()
    print("Cell",j+1, "produces the tasks: ", end="")
    for n in list_nodes:
        if(pyo.value(model.x[n,j+1]) == 1.0):
            cell_time += process_time[n]
            print(n, end=", ")
    print()
    print("The time content in cell",j+1," is",  cell_time, end="")
    print()
    print("Efficiency of cell",j+1," is",round((cell_time/cycle_time)*100,2),"%", end="")
    print()

```

References

1. Schniederjans, D.G.; Curado, C.; Khalajhedayati, M. Supply chain digitisation trends: An integration of knowledge management. *Int. J. Prod. Econ.* **2020**, *220*, 107439. [CrossRef]
2. Thomopoulos, N.T. *Assembly Line Planning and Control*; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; pp. 1–145.
3. Boysen, N.; Fliedner, M.; Scholl, A. Assembly line balancing: Which model to use when? *Int. J. Prod. Econ.* **2008**, *111*, 509–528. [CrossRef]
4. Cortés, P.; Onieva, L.; Guadix, J.; Corte´s, P.C.; Jose´guadix, J. Optimising and simulating the assembly line balancing problem in a motorcycle manufacturing company: A case study. *Int. J. Prod. Res.* **2010**, *48*, 3637–3656. [CrossRef]
5. Salveson, M.E. The assembly-line balancing problem. *Trans. Am. Soc. Mech. Eng.* **1955**, *77*, 939–947. [CrossRef]

6. Battaïa, O.; Dolgui, A. Hybridizations in line balancing problems: A comprehensive review on new trends and formulations. *Int. J. Prod. Econ.* **2022**, *250*, 108673. [CrossRef]
7. Boysen, N.; Schulze, P.; Scholl, A. Assembly line balancing: What happened in the last fifteen years? *Eur. J. Oper. Res.* **2022**, *301*, 797–814. [CrossRef]
8. Sivasankaran, P.; Shahabudeen, P. Literature review of assembly line balancing problems. *Int. J. Adv. Manuf. Technol.* **2014**, *73*, 1665–1694. [CrossRef]
9. Gurevsky, E.; Battaïa, O.; Dolgui, A. Balancing of simple assembly lines under variations of task processing times. *Ann. Oper. Res.* **2012**, *201*, 265–286. [CrossRef]
10. Hu, P.; Chu, F.; Fang, Y.; Wu, P. Novel distribution-free model and method for stochastic disassembly line balancing with limited distributional information. *J. Comb. Optim.* **2022**, *43*, 1423–1446. [CrossRef]
11. Hazir, Ö.; Dolgui, A. Assembly line balancing under uncertainty: Robust optimization models and exact solution method. *Comput. Ind. Eng.* **2013**, *65*, 261–267. [CrossRef]
12. Pereira, J.; Álvarez-Miranda, E. An exact approach for the robust assembly line balancing problem. *Omega* **2018**, *78*, 85–98. [CrossRef]
13. Zacharia, P.T.; Nearchou, A.C. A meta-heuristic algorithm for the fuzzy assembly line balancing type-E problem. *Comput. Oper. Res.* **2013**, *40*, 3033–3044. [CrossRef]
14. Li, Y.; Saldanha-da Gama, F.; Liu, M.; Yang, Z. A risk-averse two-stage stochastic programming model for a joint multi-item capacitated line balancing and lot-sizing problem. *Eur. J. Oper. Res.* **2023**, *304*, 353–365. [CrossRef]
15. Abidin Çil, Z.; Kizilay, D. Constraint programming model for multi-manned assembly line balancing problem. *Comput. Oper. Res.* **2020**, *124*, 105069. [CrossRef]
16. Nourmohammadi, A.; Fathi, M.; Ng, A.H. Choosing efficient meta-heuristics to solve the assembly line balancing problem: A landscape analysis approach. *Procedia CIRP* **2019**, *81*, 1248–1253. [CrossRef]
17. Pınarbaşı, M.; Yüzükırmızı, M.; Toklu, B. Variability modelling and balancing of stochastic assembly lines. *Int. J. Prod. Res.* **2016**, *54*, 5761–5782. [CrossRef]
18. Liu, S.B.; Ong, H.L.; Huang, H.C. A bidirectional heuristic for stochastic assembly line balancing type II problem. *Int. J. Adv. Manuf. Technol.* **2005**, *25*, 71–77. [CrossRef]
19. Zacharia, P.T.; Nearchou, A.C. Multi-objective fuzzy assembly line balancing using genetic algorithms. *J. Intell. Manuf.* **2012**, *23*, 615–627. [CrossRef]
20. Cakir, B.; Altiparmak, F.; Dengiz, B. Multi-objective optimization of a stochastic assembly line balancing: A hybrid simulated annealing algorithm. *Comput. Ind. Eng.* **2011**, *60*, 376–384. [CrossRef]
21. Hamta, N.; Fatemi Ghomi, S.M.; Jolai, F.; Akbarpour Shirazi, M. A hybrid PSO algorithm for a multi-objective assembly line balancing problem with flexible operation times, sequence-dependent setup times and learning effect. *Int. J. Prod. Econ.* **2013**, *141*, 99–111. [CrossRef]
22. Fan, W.; Gao, Z.; Xu, W.; Xiao, T. Balancing and simulating of assembly line with overlapped and stopped operation. *Simul. Model. Pract. Theory* **2010**, *18*, 1069–1079. [CrossRef]
23. Liu, C.M.; Chen, C.H. Multi-section electronic assembly line balancing problems: A case study. *Prod. Plan. Control* **2002**, *13*, 451–461. [CrossRef]
24. Sime, H.; Jana, P.K.; Panghal, D. Feasibility of Using Simulation Technique for Line Balancing In Apparel Industry. *Procedia Manuf.* **2019**, *30*, 300–307. [CrossRef]
25. Bahadir, S.K. Assembly Line Balancing in Garment Production by Simulation. In *Assembly Line*; Grzechca, W., Ed.; IntechOpen: Rijeka, Croatia, 2011; Chapter 4. [CrossRef]
26. Li, M.; Kuo, Y.; Qu, Z.Q.; Xu, Z. Simulation of Assembly Line Balancing in Automotive Component Manufacturing. *IOP Conf. Ser. Mater. Sci. Eng.* **2016**, *114*, 012049. [CrossRef]
27. Rane, A.B.; Sunnapwar, V.K.; Chari, N.R.; Sharma, M.R.; Jorapur, V.S. Improving performance of lock assembly line using lean and simulation approach. *Int. J. Bus. Perform. Manag.* **2017**, *18*, 101–124. [CrossRef]
28. Soroush, H.; Sajjadi, S.M.; Arabzad, S.M. Efficiency analysis and optimisation of a multi-product assembly line using simulation. *Int. J. Product. Qual. Manag.* **2014**, *13*, 89–104. [CrossRef]
29. Bongomin, O.; Mwasiagi, J.I.; Nganyi, E.O.; Nibikora, I. A complex garment assembly line balancing using simulation-based optimization. *Eng. Rep.* **2020**, *2*, e12258. [CrossRef]
30. Hossain, A. Assembly line balancing and sensitivity analysis of a single-model stochastic sewing line using arena simulation modelling. *Research Square* **2022**, *in press*. [CrossRef]
31. Hamza, R.M.A.; Al-Manaa, J.Y. Selection of balancing method for manual assembly line of two stages gearbox. *Glob. Perspect. Eng. Manag.* **2013**, *2*, 70–81.
32. Moncayo-Martínez, L.A. Solving the SALB-1 Problem in Process Variability. 2023. Available online: https://github.com/LuisMoncayo/StochasticSALB1 (accessed on 1 June 2023).
33. Hagberg, A.A.; Schult, D.A.; Swart, P.J. Exploring Network Structure, Dynamics, and Function using NetworkX. In Proceedings of the 7th Python in Science Conference; Pasadena, CA USA, 19–24 August 2008; Varoquaux, G., Vaught, T., Millman, J., Eds.; pp. 11–15.

34. Makhorin, A. GNU Linear Programming Kit Version 4.32. 2012. Available online: https://www.gnu.org/software/glpk/#introduction (accessed on 1 June 2023).
35. Bynum, M.L.; Hackebeil, G.A.; Hart, W.E.; Laird, C.D.; Nicholson, B.L.; Siirola, J.D.; Watson, J.P.; Woodruff, D.L. *Pyomo—Optimization Modeling in Python*; Springer Nature Switzerland AG: Cham, Switzerland, 2021; Volume 67. [CrossRef]
36. Smith, J.; Sturrock, D. *Simio and Simulation: Modeling, Analysis, Applications*, 6th ed.; Simio Forward Thinking: Pennsylvania, USA, 2012.
37. van Assen, M.F. Lean, process improvement and customer-focused performance. The moderating effect of perceived organisational context. *Total Qual. Manag. Bus. Excell.* **2021**, *32*, 57–75. [CrossRef]
38. Sadeghi, A.; Suer, G.; Sinaki, R.Y.; Wilson, D. Cellular manufacturing design and replenishment strategy in a capacitated supply chain system: A simulation-based analysis. *Comput. Ind. Eng.* **2020**, *141*, 106282. [CrossRef]
39. Özceylan, E.; Kalayci, C.B.; Güngör, A.; Gupta, S.M. Disassembly line balancing problem: A review of the state of the art and future directions. *Int. J. Prod. Res.* **2019**, *57*, 4805–4827. [CrossRef]