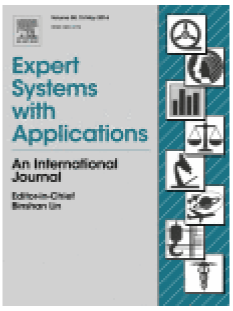


# Lexicographic bottleneck mixed-model assembly line balancing problem: Artificial bee colony and tabu search approaches with optimised parameters

**Kadir Buyukozkan [a,c], Ibrahim Kucukkoc [b,d*], Sule Itir Satoglu [a], David Z. Zhang [b]**

[a] Industrial Engineering Department, Istanbul Technical University, Istanbul, Turkey

[b] College of Engineering, Mathematics and Physical Sciences, University of Exeter, North Park Road, EX4 4QF Exeter, England, United Kingdom

[c] Department of Industrial Engineering, Faculty of Engineering, Karadeniz Technical University, Kanuni Campus, Trabzon, Turkey

[d] Department of Industrial Engineering, Faculty of Engineering and Architecture, Balikesir University, Cagis Campus, Balikesir, Turkey

** Corresponding author: Ibrahim Kucukkoc, Email: i.kucukkoc@exeter.ac.uk, Tel: +441392723613. K.B Email: kbuyukozkan@ktu.edu.tr, Tel: +904623772954; S.I.S Email: onbaslis@itu.edu.tr, Tel: +902122856801; D.Z.Z Email: d.z.zhang@exeter.ac.uk, Tel: +441392723641.*

**Abstract:** The lexicographic bottleneck assembly line balancing problem is a recently introduced problem which aims at obtaining a smooth workload distribution among workstations. This is achieved hierarchically. The workload of the most heavily loaded workstation is minimised, followed by the workload of the second most heavily loaded workstation and so on. This study contributes to knowledge by examining the application of the lexicographic bottleneck objective on mixed-model lines, where more than one product model is produced in an inter-mixed sequence. The main characteristics of the lexicographic bottleneck mixed-model assembly line balancing problem are described with numerical examples. Another contribution of the study is the methodology used to deal with the complex structure of the problem. Two effective meta-heuristic approaches, namely artificial bee colony and tabu search, are proposed. The parameters of the proposed meta-heuristics are optimised using response surface methodology, which is a well-known design of experiments technique, as a unique contribution to the expert and intelligent systems literature. Different from the common tendency in the literature (which aims to optimise one parameter at a time), all parameters are optimised simultaneously. Therefore, it is shown how a complex production planning problem can be solved using sophisticated artificial intelligence techniques with optimised parameters. The methodology used for parameter setting can be applied to other metaheuristics for solving complex problems in practice. The performances of both algorithms are assessed using well-known test problems and it is observed that both algorithms find promising solutions. Artificial bee colony algorithm outperforms tabu search in minimising the number of workstations while tabu search shows a better performance in minimising the value of lexicographic bottleneck objective function.

**Keywords:** mixed-model assembly line balancing; lexicographic bottleneck; artificial bee colony; tabu search, parameter optimisation; response surface methodology.

**Expert Systems with Applications**

# Lexicographic Bottleneck Mixed-model Assembly Line Balancing Problem: Artificial Bee Colony and Tabu Search Approaches with Optimised Parameters

Kadir Buyukozkan [ac], Ibrahim Kucukkoc [bd*], Sule Itir Satoglu [a], David Z. Zhang [b]

[a] Industrial Engineering Department, Istanbul Technical University, Istanbul, Turkey

[b] College of Engineering, Mathematics and Physical Sciences, University of Exeter, North Park Road, EX4 4QF Exeter, England, United Kingdom

[c] Department of Industrial Engineering, Faculty of Engineering, Karadeniz Technical University, Kanuni Campus, Trabzon, Turkey

[d] Department of Industrial Engineering, Faculty of Engineering and Architecture, Balikesir University, Cagis Campus, Balikesir, Turkey

## Abstract

The lexicographic bottleneck assembly line balancing problem is a recently introduced problem which aims at obtaining a smooth workload distribution among workstations. This is achieved hierarchically. The workload of the most heavily loaded workstation is minimised, followed by the workload of the second most heavily loaded workstation and so on. This study contributes to knowledge by examining the application of the lexicographic bottleneck objective on mixed-model lines, where more than one product model is produced in an inter-mixed sequence. The main characteristics of the lexicographic bottleneck mixed-model assembly line balancing problem are described with numerical examples. Another contribution of the study is the methodology used to deal with the complex structure of the problem. Two effective meta-heuristic approaches, namely artificial bee colony and tabu search, are proposed. The parameters of the proposed meta-heuristics are optimised using response surface methodology, which is a well-known design of experiments technique, as a unique contribution to the expert and intelligent systems literature. Different from the common tendency in the literature (which aims to optimise one parameter at a time), all parameters are optimised simultaneously. Therefore, it is shown how a complex production planning problem can be solved using sophisticated artificial intelligence techniques with optimised parameters. The methodology used for parameter setting can be applied to other metaheuristics for solving complex problems in practice. The performances of both algorithms are assessed using well-known test problems and it is observed that both algorithms find promising solutions. Artificial bee colony algorithm outperforms tabu search in minimising the number of workstations while tabu search shows a better performance in minimising the value of lexicographic bottleneck objective function.

**Keywords:** mixed-model assembly line balancing; lexicographic bottleneck; artificial bee colony; tabu search, parameter optimisation; response surface methodology.

* Corresponding author: Ibrahim Kucukkoc, Email: i.kucukkoc@exeter.ac.uk, Tel: +441392723613.

K.B Email: kbuyukozkan@ktu.edu.tr, Tel: +904623772954; S.I.S Email: onbaslis@itu.edu.tr, Tel: +902122856801; D.Z.Z Email: d.z.zhang@exeter.ac.uk, Tel: +441392723641.

## 1. Introduction

Six decades have passed since the idea of balancing assembly lines has been introduced by Bryton (1954). This idea was described systematically and formulated mathematically by Salveson (1955) and has received a great deal of attention from both academia and industry. With the change of the global market, product diversity has become one of the key parameters in attracting customers. Mass customisation has been adopted by companies as a flexible manufacturing tool to meet diversified customer demands in a timely manner (Goh & Zhang, 2003; I. Kucukkoc & D.Z. Zhang, 2014). In this context, companies converted existing single-model lines into mixed-model lines to enable producing more than one product model on the same line where set-up times between model changes are small enough (Zhang & Kucukkoc, 2013). Thus, similar product models could be produced on the same line, avoiding the cost of utilising a new line for each product model.

Battaïa and Dolgui (2013) presented a taxonomy of line balancing problems and their solution approaches, recently. In its traditional and simplest form, which is called *simple assembly line balancing problem*, the assembly line balancing problem is assigning tasks into a serially linked set of workstations by ensuring that capacity constraints and precedence relationship constraints are satisfied. The tasks belonging to a single commodity (or product model) are performed on the line and a decision is made to determine which task will be accommodated in which workstation (I. Kucukkoc & Zhang, 2013; I. Kucukkoc & D. Z. Zhang, 2015).

Obtaining a smooth workload distribution among the workstations is very important to have a well-balanced and reliable assembly line which has strengths against unforeseeable circumstances such as breakdowns and other tolerable small extra works that can be performed while the line is running. For this aim, Pastor *et al.* (2011; 2012) proposed a new approach to systematically distribute the total workload among the workstations utilised across the line. Also, they showed the advantage of the lexicographic bottleneck objective function (Buyukozkan, Kucukkoc, & Zhang, 2014). The main aim of this paper is to experiment the lexicographic bottleneck objective on mixed-model assembly line balancing problem and to address the *lexicographic bottleneck mixed-model assembly line balancing problem*. Two powerful solution approaches, namely artificial bee colony and tabu search, are also developed to solve the addressed problem efficiently. The parameters of the proposed algorithms are optimised using a robust methodology that can be applied to other metaheuristics.

The remainder of the paper is organised as follows. Section 2 provides a comprehensive review of the literature on mixed-model assembly line balancing problem as well as the applications of proposed techniques on various line balancing configurations. Section 3 describes the main characteristics of the lexicographic bottleneck mixed-model assembly line balancing problem and briefly presents its differences from the existing problems. Section 4 presents the artificial bee colony and tabu search algorithms developed to solve the addressed problem and gives numerical examples to describe their solution procedures stepwise. Section 5 explains the response surface methodology (called RSM hereafter) used for parameter setting, and reports the experimental test results. After a brief discussion on the strengths and weaknesses of the proposed research method in Section 6, Section 7 draws conclusions along with the future research directions.

## 2. Literature Review

The mixed-model assembly line balancing problem, which aims at finding an optimal assignment of tasks belonging to more than one product model produced on the same line, has been introduced by Thomopoulos (1967, 1970). Since then, the mixed-model assembly line balancing problem has attracted a vast number of researchers from both academia and industry. Minimising the number of workstations, which could also help reduce cost, has been considered as an ultimate goal in the majority of studies, *e.g.* Simaria and Vilarinho (2009), Kara and Tekin (2009), Ozcan and Toklu (2009), Xu and Xiao (2011), Yagmahan (2011), Akpinar and Bayhan (2011), Hamzadayi and Yildiz (2012), Rabbani *et al*. (2012), Chutima and Chimklai (2012), Liao *et al*. (2012), Akpinar *et al*. (2013), Kucukkoc *et al*. (2013), Manavizadeh *et al*. (2013), Kucukkoc *et al*. (2013) and Kucukkoc and Zhang (2014b). This problem is called type-1, as the number of workstations is minimised for a predefined cycle time value.

Minimising cycle time for the given number of workstations is another type of line balancing problem, called type-2. This problem has been studied by Simaria and Vilarinho (2004), Battini *et al.* (2007) and Ozcan *et al*. (2011) recently. Some studies considered both objectives (the minimisation of cycle time and the number of workstations) at the same time, such as Manavizadeh *et al*. (2012) and Kucukkoc and Zhang (2015). Manavizadeh *et al*. (2012) proposed a multi-objective genetic algorithm based approach to optimise both the number of workstations and the cycle time (called type-E) in a stochastic make-to-order environment. Kucukkoc and Zhang (2015) addressed the type-E problem on parallel two-sided assembly lines and proposed an ant colony optimisation algorithm where the parameters of ant colony optimisation algorithm are calibrated through RSM, which is a well-known design of experiment method proposed by Box and Wilson (1951).

Workload smoothness is an important criterion which also needs to be taken into account: *(i)* to maintain an even workload distribution among workstations and so workers, *(ii)* to improve the quality of product(s) assembled on the line, and *(iii)* to reduce the risk of incomplete tasks exceeding the cycle time due to some unforeseeable circumstances. This criterion has been taken into account as an additional performance measure in many researches, such as Simaria and Vilarinho (2009), Ozcan and Toklu (2009), Ozcan *et al*. (2010), Akpinar and Bayhan (2011), Yagmahan (2011), Hamzadayi and Yildiz (2012), Liao *et al*. (2012), Chutima and Chimklai (2012), Manavizadeh *et al*. (2013) and Kucukkoc *et al*. (2013).

In terms of the applied solution techniques, there is an increasing interest in the applications of population (or swarm intelligence) based and neighbourhood search based optimisation algorithms on a variety of line balancing problems. Specifically, Simaria and Vilarinho (2009), Yagmahan (2011), Kucukkoc *et al.* (2013) and Kucukkoc and Zhang (2014a, 2014b) developed different ant colony optimisation based approaches; Ozcan *et al*. (2011), Xu and Xiao (2011), Akpinar and Bayhan (2011), Hamzadayi and Yildiz (2012), Rabbani *et al*. (2012), Manavizadeh *et al*. (2012) and Kucukkoc *et al*. (2013) developed different genetic algorithm based techniques and Chutima and Chimklai (2012) proposed a particle swarm optimisation approach for the solution of mixed-model assembly line balancing problem. However, the applications of bee colony optimisation (D. Karaboga, 2005) and bees algorithm (Pham, et al., 2006) are quite scarce in the entire line balancing domain. Akpinar and Baykasoğlu (2014) applied the bee colony algorithm for solving the mixed-model assembly line balancing problem. Özbakir and Tapkan (2011) and Tapkan *et al*. (2012a, 2012b) used bee colony intelligence and bees algorithms, respectively, to solve zone constrained two-sided assembly line balancing problem.

Lapierre *et al.* (2006) applied tabu search algorithm for solving the simple assembly line balancing problem (with the aim of minimising the number of workstations) and non-standard versions of this problem coming from real life. Computational tests showed that the proposed tabu search method had advantages over existing priority based procedures. Ozcan and Toklu (2008) presented a tabu search algorithm for two-sided assembly line balancing problem, where workstations are located on both sides (left and right) of a straight line. The performance of the proposed method was compared to the existing methods and it was observed that the proposed method performed well. Özcan *et al.* (2009) used the tabu search to balance the parallel lines, where resources between two adjacent lines are shared, with the aim of maximising line efficiency. The proposed method was illustrated through a numerical example and its performance was tested through existing test problems. Özcan *et al.* (2010) introduced the parallel two-sided assembly line configuration, which is referred to as advantageous for producing large-sized items (such as buses and trucks) and developed a tabu search based approach. Two numerical examples were given and computational tests were performed to explain the solution building mechanism of the algorithm and to prove its performance. Esmaeilian *et al.* (2011) presented a tabu search approach, which incorporates a heuristic procedure to provide an initial solution, for solving the parallel assembly line balancing problem in a mixed-model production environment. The computational experiments showed that the proposed approach produces promising solutions.

The lexicographic bottleneck objective has been recently introduced for assembly line balancing domain. Pastor *et al*. (2011) presented and formalised the lexicographic bottleneck assembly line balancing problem; which aims at hierarchically minimising the workload of the most heavily loaded workstation, followed by the workload of the second most heavily loaded workstation and so on. Two mixed-integer linear programming models and three heuristic procedures were proposed for solving the problem. Pastor *et al*. (2011) have proven that the lexicographic bottleneck objective has advantages over traditional smoothness index objectives to obtain a more smoothly distributed workload across the workstations. In their latter study, Pastor *et al*. (2012) proposed and tested new algorithms, which were different combinations of a heuristic procedure and several local search procedures derived from the literature. The computational experiments showed that the heuristic procedure developed by Pastor *et al*. (2012) was an improvement upon the heuristic procedures (three heuristic procedures based on two mixed-integer linear programming models) published by Pastor *et al*. (2011).

The lexicographic bottleneck assembly line balancing problem, which is different from type-2 line balancing problem as it was exposed by Pastor *et al*. (2011; 2012), has not been studied properly for mixed-model lines in the literature. Based on this motivation, we apply the lexicographic bottleneck objective on mixed-model assembly line balancing problem and propose new solution techniques, namely an artificial bee colony algorithm and a tabu search algorithm, for the possible solution of the addressed problem. This paper is original in terms of both the addressed problem and the proposed solution methods. This paper addresses the lexicographic bottleneck mixed-model assembly line balancing problem, whose primitive version was introduced by Pastor *et al*. (2011) for the simple assembly line balancing problem, by building work on Kucukkoc *et al*. (2015). In addition to the developed artificial bee colony algorithm, the tabu search algorithm is applied for any type of lexicographic bottleneck assembly line balancing problem for the first time in the literature. Also, the parameters of both proposed techniques are optimised using RSM as a pioneering research in expert and intelligent systems domain. Therefore, this paper contributes to knowledge by not only addressing a newly introduced assembly line balancing problem type but also developing novel artificial bee colony and tabu search algorithms (with optimised parameters through RSM) for solving this problem.

## 3. Problem Statement

The workload time of a workstation is constituted by the summation of processing times of all tasks assigned to that workstation. In traditional type-2 assembly line balancing problems, the workload of the most heavily loaded workstation is minimised as it determines the cycle time of the entire production system in synchronised assembly lines. As only the workload of the workstation which has the largest workload time is minimised, the remaining workload times are ignored in type-2 assembly line balancing problems. However, as indicated by Pastor *et al*. (2012) and Boysen *et al*. (2007), there are some important factors that tightly depend on the workload distribution among all workstations: the reliability of the line, uniform (or equitable) distribution of the total workload among all operators, quality defects caused by stations with disproportionately large station times, *etc*. Therefore, it is important to consider the second-largest workload, the third-largest workload, *etc*., as the criticalness of the workstations and the reliability of the line are tightly interrelated to each other. The larger the difference between the total workload of a workstation and cycle time, the less critical becomes the workstation. That means the reliability of the entire system could be increased by reducing the criticalness (R. Pastor, 2011).

Pastor *et al*. (2011) have already proven with an example that the optimal solution based on the lexicographic bottleneck objective may be different from the optimal solution based on the 'Smoothness Index' objective, which was considered as an additional objective in many researchers. Although this situation has been proven for only a simple assembly line balancing problem, it is highly possible to observe the same situation for mixed-model assembly lines, as well.

The main aim of the introduced problem in this research, which is called lexicographic bottleneck mixed-model assembly line balancing problem, is to hierarchically minimise the weighted-workload of the most heavily loaded workstation, followed by the weighted-workload of the second most heavily loaded workstation and so on. More details on how to calculate the weighted-workload of a workstation will be specified below.

Let us consider the graph given in Figure 1 as the combined precedence relationships diagram of three models $(m_1, m_2, \text{and } m_3)$, where a model is symbolised with $m_j$ $(j = 1, \ldots, M)$, assembled on a mixed-model assembly line. Table 1 presents the processing times $(pt_{ji})$ of tasks, where a task is represented by $t_{ji}$ $(i = 1, \ldots, T_j)$, belonging to the product models. Demands $(D_j)$ are considered $16$, $24$ and $8$ for models $m_1, m_2, \text{and } m_3$ , respectively $(D_1 = 16,\ D_2 = 24 \text{ and } D_3 = 8)$. The task processing times given in this table are taken from Simaria (2006) except that of task-10. As cycle time $(CT)$ is assumed 12.5 time units, in such an environment where parallel workstations or feeding lines are not allowed, the processing times of all tasks must be smaller than the cycle time. For that reason, the processing time of task-10 is assumed 8.6 time units rather than its original value of 13.6 time units, which exceeds the cycle time.

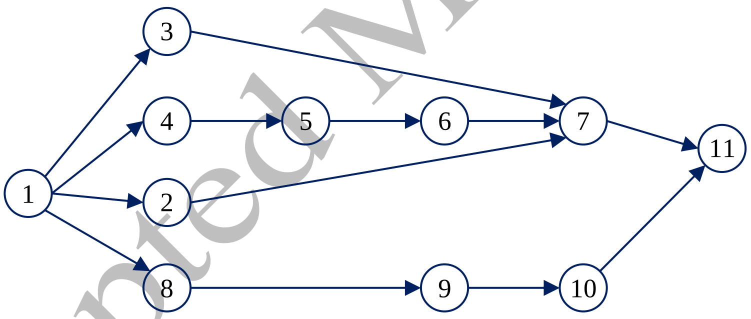


Figure 1. Combined precedence relationships diagram of the instance, adapted from Gökçen and Erel (1998)

Table 1. Task processing times of the models for the given instance

| **Task No / Model** | $m_1$ | $m_2$ | $m_3$ |
|---|---|---|---|
| 1 | 8.3 | 8.6 | 8.3 |
| 2 | 0.0 | 2.0 | 2.0 |
| 3 | 9.6 | 9.6 | 9.6 |
| 4 | 1.8 | 1.8 | 1.8 |
| 5 | 2.4 | 2.4 | 2.5 |
| 6 | 2.3 | 2.3 | 2.3 |
| 7 | 2.3 | 2.3 | 2.5 |
| 8 | 4.7 | 4.7 | 4.7 |
| 9 | 0.0 | 9.0 | 9.0 |
| 10 | 8.6 | 8.6 | 8.6 |
| 11 | 1.0 | 1.0 | 1.0 |

The two possible line balancing assignments of the example problem are given in Table 2 where upper bound of $CT$ is set to 12.5 time units. The table presents the task assignments to the workstations, symbolised with $k$ $(k = 1, \dots, K)$, and the workloads of workstations based on these assignments. Each workstation's workload time for individual models and also the total weighted-workloads $(WW)$ of the workstations are calculated and presented in the table.

Table 2. Two alternative assignment solutions of the given instance

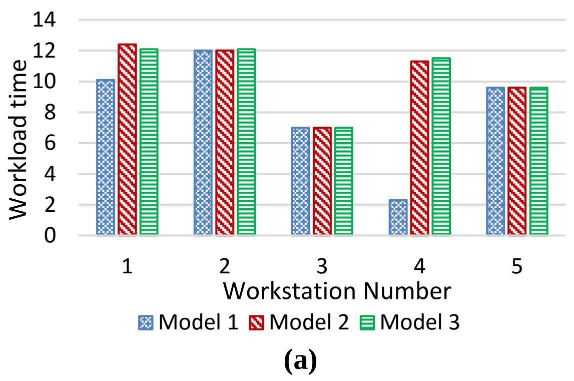
| **Alternative Assignment 1** | | | | | | **Alternative Assignment 2** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Station | Assigned Tasks | Workloads (Time Units) | | | Weighted-workload (WW) | Station | Assigned Tasks | Workloads (Time Units) | | | Weighted-workload (WW) |
| | | $m_1$ | $m_2$ | $m_3$ | | | | $m_1$ | $m_2$ | $m_3$ | |
| 1 | 1,2,4 | 10.10 | 12.40 | 12.10 | 11.58 | 1 | 1,4 | 10.10 | 10.40 | 10.10 | 10.25 |
| 2 | 3,5 | 12.00 | 12.00 | 12.10 | 12.02 | 2 | 2,3 | 9.60 | 11.60 | 11.60 | 10.93 |
| 3 | 6,8 | 7.00 | 7.00 | 7.00 | 7.00 | 3 | 5,6,8 | 9.40 | 9.40 | 9.50 | 9.42 |
| 4 | 9,7 | 2.30 | 11.30 | 11.50 | 8.33 | 4 | 7,9 | 2.30 | 11.30 | 11.50 | 8.33 |
| 5 | 10,11 | 9.60 | 9.60 | 9.60 | 9.60 | 5 | 10,11 | 9.60 | 9.60 | 9.60 | 9.60 |

The weighted-workload $(WW)$ of a workstation corresponds to the sum of weighted processing times of all tasks assigned to this workstation. The weighted processing time of a task is obtained by summing the multiplications of processing times belonging to different models by proportional demands of these models. Also, proportional demand of a model corresponds to division of this model's demand by the total demand of the models produced on the same line $(D_j/\sum_{j=1}^{M} D_j)$. Thus, the weighted-workload of station $k$, represented by $WW_k$, could be calculated as in Equation (1).

$$WW_k = \sum_{j=1}^{M}\left(\frac{D_j}{\sum_{j=1}^{M} D_j} \cdot \sum_{t_{ji} \in S_k} pt_{ji}\right), \quad (1)$$

where $S_k$ denotes the set of tasks assigned to workstation $k$. $D_j$ and $pt_{ji}$ are demand for model $m_j$ and processing time of task $t_{ji}$, respectively.

As seen from Table 2, five workstations are needed to perform a total of 11 tasks for both situations. However, there are differences in the *workloads* and *weighted-workload* columns of these two alternative solutions. Figure 2 comparatively depicts the workload distributions among workstations for both solutions. Apparently, the second solution has a more uniform workload distribution than the first solution. It should be noted here that there could be more fluctuation in the distribution of workloads between workstations if there were tens of tasks (not five) to be assigned. However, even this small-sized numerical example shows the dependency of workload times to model types. For example, the workload of workstation 4 equals to 2.30 time units when model 1 is being operated in this workstation. However, this will change when model 2 and model 3 will be assembled in workstation 4. Specifically, the workload will dramatically increase from 2.30 time units to 11.30 and 11.50 time units when model 2 and model 3 are assembled, respectively. On one hand, huge idle times will occur when producing model 1 in workstation 4. On the other hand, the workstation will be loaded almost full when producing model 2 and model 3. This clearly shows the importance of having a smooth workload across workstations on mixed-model assembly lines.



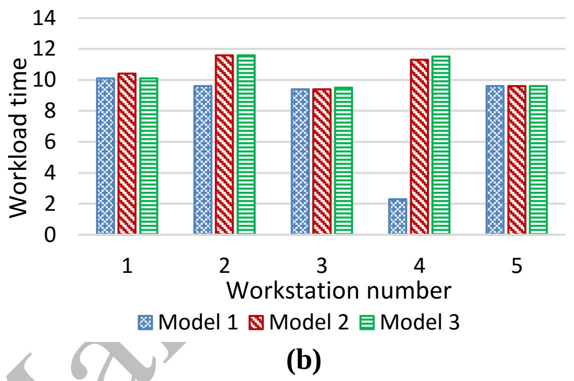


Figure 2. Workload distributions among workstations: (a) Solution-1, (b) Solution-2

Figure 3 exhibits the weighted-workload of each workstation for both alternative solutions. As seen from the figure, distributing the weighted-workload among five workstations as 11.58, 12.02, 7.00, 8.33 and 9.60 is not equivalent to distributing it as 10.25, 10.93, 9.42, 8.33 and 9.60. Although both solutions are obtained under the same upper bound for the cycle time $(CT = 12.5)$, the second solution has less critical workstations and it is more reliable.

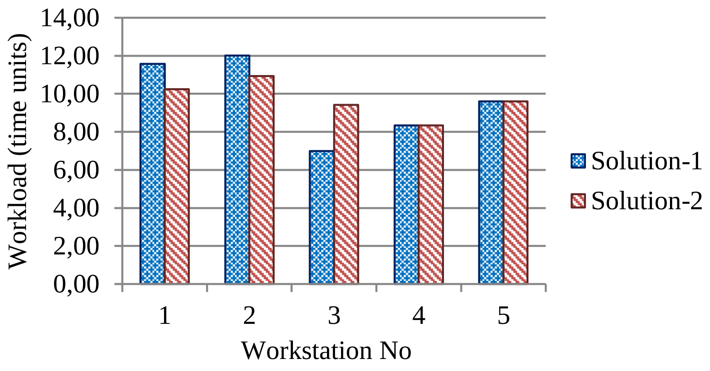


Figure 3. Weighted-workload of each workstation for both alternative solutions

Following assumptions are made for the solution of the lexicographic bottleneck mixed-model assembly line balancing problem:

- Two or more models of a product are assembled on a paced (synchronous) mixed-model assembly line.

- Model demands are deterministic and known for a pre-defined planning horizon.
- Task processing times are deterministic and known for each product model produced on the line. When a specific task is not necessary for a certain model, then its processing time is considered zero.
- A common (or joint) precedence diagram, which satisfies all precedence relationships between tasks belonging to different product models, is used in order to maximise the resource utilisation by assigning common tasks between similar models to the same workstation.
- Each task for each product model must be assigned to exactly one workstation. Tasks cannot be split into two or more workstations.
- A task can only be assigned if all of its predecessors have been assigned and completed.
- Operators are multi-skilled and any task can be performed at any workstation with no change in its processing time.
- Parallel workstations and buffers between workstations (or work in progress inventory) are not allowed.
- Setup operations are not required between model changes.
- Operator travel times are ignored.

## 4. Proposed Algorithms

This section presents the artificial bee colony and tabu search algorithms proposed for solving the lexicographic bottleneck assembly line balancing problem. Both artificial bee colony and tabu search approaches have been recognised as powerful and flexible optimisation algorithms, which have the capability of robustly solving global optimisation problems with linear and nonlinear objective functions. As shown by Karaboga and Akay (2009) through a set of comprehensive experimental tests, artificial bee colony is better than or similar to those of other population-based algorithms with the advantage of employing fewer control parameters. It is flexible and has strengths in both local and global searches. This reduces the possibility of being trapped at a local optimum. In comparison with evolutionary algorithms, the fast convergence feature of artificial bee colony algorithm – which can be considered an advantage in a timely manner – may result in premature convergence as a possible disadvantage of the algorithm.

Different from most metaheuristics, including tabu search, the basic artificial bee colony approach employs multiple random starting points for initialisation. This attribute helps to explore the search space more effectively. On the other hand, as emphasised by Glover and Marti (2006) a bad strategic choice may often yield better results than a good random choice. Therefore, by taking the advantage of keeping search history, this strategic choice builds a base point for the progressive improvement of tabu search. Both tabu search and artificial bee colony algorithms can be easily implemented for a wide range of optimisation problems. Although some promising areas of the search space may be missed when only one solution is used and the algorithm is purely progressed from its neighbourhood solutions, tabu search still produces promising results as will be shown in the following subsections. Tabu search needs relatively less number of parameters in comparison with most of the metaheuristics, including ant colony, genetic and artificial bee colony algorithms. However, as in the majority of other metaheuristics, both algorithms need parameter tuning. For this reason, a novel RSM based parameter optimisation approach is adopted, as will be explained in Section 5.

## 4.1. Artificial bee colony algorithm

Artificial bee colony algorithm, which is proposed by Karaboga (2005), is a swarm intelligence method. It was inspired from the foraging behaviour of bees in nature. In social life, foraging begins with random food search. The *scout bees* find new food sources in their neighbourhood and perform waggle-dance in front of the hive. This dance movement gives information about *(i)* the distance from the food source to the hive, *(ii)* the nectar quality of the source and *(iii)* the nectar quantity of the source. The onlooker bees in the hive watch this dance movement and choose the scout bees to follow. When an onlooker bee follows a scout bee, it starts foraging and is called a *follower bee*. Each bee, which collects food, performs waggle-dance to give information about the food source. By this way, the bees meet the food need of the hive. When a food source is exhausted, this source is abandoned and scout bees continue to seek new sources.

When the behaviour of bees is adapted to an optimisation problem, each bee represents a solution and the algorithm starts with randomly generated initial solutions. These solutions are considered as scout bees and the algorithm continues with the neighbourhood search with the help of the follower bees around these solutions. The general structure of the proposed artificial bee colony algorithm is presented in Figure 4.

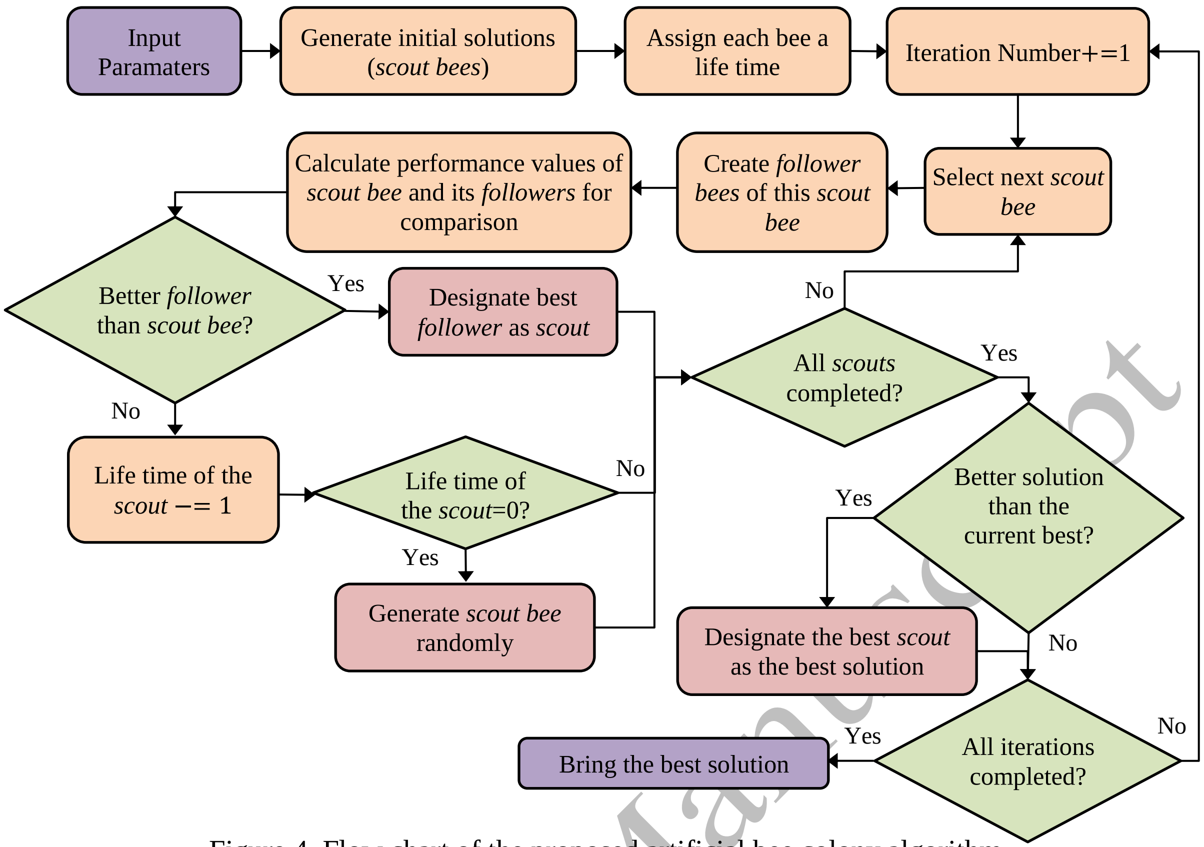

Input Paramaters
Generate initial solutions (scout bees)
Assign each bee a life time
Iteration Number+=1
Select next scout bee
Create follower bees of this scout bee
Calculate performance values of scout bee and its followers for comparison
Better follower than scout bee?
Yes
Designate best follower as scout
No
Life time of the scout −= 1
Life time of the scout=0?
No
Yes
Generate scout bee randomly
All scouts completed?
No
Yes
Better solution than the current best?
Yes
Designate the best scout as the best solution
No
All iterations completed?
Yes
No
Bring the best solution

Figure 4. Flow chart of the proposed artificial bee colony algorithm

The parameters of the algorithm (namely, the number of scout bees ($S$), the number of follower bees ($F$), the maximum number of iterations ($MaxIter$), life time ($LF$), cycle time ($CT$) and hierarchy parameter ($\beta$)) are initialised. Afterwards, $S$ number of solutions are generated considering the precedence relationships. While generating these solutions, the tasks are sequenced in random orders ensuring that the precedence relations are not violated. The workstations are formed by splitting the ordered tasks into groups (which is equivalent to allocating the tasks into the workstations) considering the $CT$ value. Hence, it is ensured that the initial solutions (scout bees) are produced very quickly. Each scout bee is assigned an $LF$ value, which corresponds to the maximum number of trials that can be passed without improvement. $F$ number of follower bees are directed to each scout bee for the neighbourhood search mechanism.

The neighbourhoods are searched using insert method. In doing so, two random numbers, *i.e.* $r_1$ and $r_2$, are generated and the task located at $r_1$th position is relocated to $r_2$th position by checking the precedence relationships to ensure feasibility. If this move is considered to violate the precedence relationships, a new point ($r_2$), which satisfies the precedence relationships is determined and the task is allocated to this position. After completing neighbourhood searches for all followers in such a way, the performance values ($\delta$) of scout bees and their followers are calculated using Equation (2), which is modified from Pastor (2011). This is equivalent to using Equation (3) when comparing two different solutions. By this way, it is endeavoured to hierarchically minimise the weighted-workload of the most heavily loaded workstation.

$$\delta = \frac{\sum_{k=1}^{K}\left(\beta^{K-k+1} \cdot WW_k\right)}{CT \cdot \beta^{K-1}}. \tag{2}$$

$$\delta_{diff} = \frac{\sum_{k=1}^{K}\left(\beta^{K-k+1} \cdot \sum_{j=1}^{J} d_j \cdot \Delta_{kj}\right)}{CT_{best} \cdot \beta^{K-1}}, \tag{3}$$

where $\Delta_{kj}$ is the positive, null, or negative workload difference in terms of model $m_j$ in the $k^{th}$ most heavily loaded workstation between the worst and best solutions compared. $CT_{best}$ is the best cycle time of the two solutions compared. $\beta$ is a parameter whose value must guarantee the hierarchy of the objectives ($\beta > \max(\Delta_{kj} - \Delta_{(k+1)j})$) and $d_j$ is the proportional demand of model $m_j$, $\left(d_j = \frac{D_j}{\sum_{j=1}^{M} D_j}\right)$. Please refer to Pastor (2011) for more details about $\beta$ parameter.

The best performance value among the followers is compared with the performance value of its scout bee and the scout bee is replaced with that follower if the follower is better. If not, $LF$ value of the scout bee is decreased by one. When the $LF$ value gets zero, scout bee is replaced with a randomly generated feasible solution. The global best solution is updated if any of the scout bees has a better performance value than the global best. The same procedures are carried out for all scout bees and this cycle continues until all iterations are completed. Note that better performance values could be obtained with solutions which require larger numbers of workstations. However, in practice, designs which require fewer number of workstations are preferred by line managers. For that reason, when two solutions which require different numbers of workstations are subject to comparison, the solution which requires the fewer number of workstations is favoured regardless of its performance value.

For a better understanding of the steps of the algorithm, a numerical example is given below. The precedence relationships and task processing times are given in Figure 1 and Table 1, respectively. The parameters considered are as follows: $S = 5$, $F = 10$, $MaxIter = 25$, $LF = 5$, $CT = 12.5$ and $\beta = 100$.

Table 3 shows the initial solutions (scout bees) generated randomly. Let us consider the first scout bee in Table 3. Three follower bees generated from this scout bee are given in Table 4. The follower bees search neighbourhood solutions around the scout bee using the randomly determined two numbers. It could be seen that the random numbers are 2 and 5 for the first follower ($r_1 = 2, r_2 = 5$). So the job located in the $2^{nd}$ position is relocated to the $5^{th}$ position for this follower. The random numbers for the second and the third followers are 3 and 9, and 7 and 5, respectively. The performance values of the followers are also given in Table 4. To calculate the performance value of a solution, the tasks are assigned to the workstations as in the same order they are sequenced by bees and Equation (2) is used for calculation.

Table 3. Initial solutions generated by the algorithm

| **#** | **Sequence of Tasks** | | | | | | | | | | | | **Workstations** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | **1** | **2** | **3** | **4** | **5** | **6** |
| ***1*** | 1 | 2 | 3 | 4 | 8 | 9 | 5 | 10 | 6 | 7 | 11 | Assignments | 1,2 | 3,4 | 8 | 9,5 | 10,6 | 7,11 |
| | | | | | | | | | | | | Weighted Stat. Workloads | 9.790 | 11.400 | 4.700 | 8.450 | 10.900 | 3.340 |
| ***2*** | 1 | 2 | 4 | 3 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | Assignments | 1,2,4 | 3,5 | 6,7,8 | 9 | 10,11 | - |
| | | | | | | | | | | | | Weighted Stat. Workloads | 11.590 | 12.070 | 9.330 | 6.030 | 9.600 | - |
| ***3*** | 1 | 2 | 3 | 4 | 8 | 5 | 6 | 9 | 10 | 7 | 11 | Assignments | 1,2 | 3,4 | 8,5,6 | 9 | 10,7,11 | - |
| | | | | | | | | | | | | Weighted Stat. Workloads | 9.790 | 11.400 | 9.420 | 6.030 | 11.930 | - |
| ***4*** | 1 | 4 | 5 | 6 | 8 | 3 | 9 | 2 | 10 | 7 | 11 | Assignments | 1,4 | 5,6,8 | 3 | 9,2 | 10,7,11 | - |
| | | | | | | | | | | | | Weighted Stat. Workloads | 10,250 | 9,417 | 9,600 | 7,370 | 11,934 | - |
| ***5*** | 1 | 4 | 8 | 5 | 6 | 2 | 9 | 3 | 7 | 10 | 11 | Assignments | 1,4 | 8,5,6 | 2,9 | 3,7 | 10,11 | - |
| | | | | | | | | | | | | Weighted Stat. Workloads | 10,250 | 9,417 | 6,030 | 11,934 | 9,600 | - |

Table 4. Followers of the first scout bee

| | **Task Sequence** | | | | | | | | | | | **Weighted-workload Times ($WW$)** | | | | | | **Performance Value ($\delta$)** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | **1** | **2** | **3** | **4** | **5** | **6** | |
| **Scout Bee** | 1 | 2 | 3 | 4 | 8 | 9 | 5 | 10 | 6 | 7 | 11 | 9.79 | 11.40 | 4.70 | 8.45 | 10.90 | 3.34 | 92.080 |
| **Follower-1** | 1 | 3 | 4 | 8 | 2 | 9 | 5 | 10 | 6 | 7 | 11 | 8.45 | 11.39 | 6.04 | 8.45 | 10.90 | 3.34 | 91.999 |
| **Follower-2** | 1 | 2 | 4 | 8 | 9 | 5 | 10 | 6 | 3 | 7 | 11 | 11.59 | 4.70 | 8.45 | 10.90 | 11.93 | 1.00 | 96.376 |
| **Follower-3** | 1 | 2 | 3 | 4 | 5 | 8 | 9 | 10 | 6 | 7 | 11 | 9.79 | 11.39 | 7.12 | 6.03 | 10.90 | 3.34 | 92.000 |

The overall survival of this scout bee through 25 iterations is presented in Table 5. As could be seen from the table, the scout bee is regenerated randomly in the tenth and twentieth iterations to avoid local minima. Better solutions are sought by local search procedures within the neighbourhoods of all scout bees at different locations of the global search space. Figure 5 presents the movements of five scout bees. As seen, these five scouts find solutions with different performance values, which prove the algorithm's effective search capability.

Table 5. Survival of the first scout bee through 25 iterations

| **Iteration** | **Task Sequence** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Scout Bee** | 1 | 2 | 3 | 4 | 8 | 9 | 5 | 10 | 6 | 7 | 11 |
| **Iteration-1** | 1 | 3 | 4 | 8 | 9 | 5 | 2 | 10 | 6 | 7 | 11 |
| **Iteration-2** | 1 | 4 | 8 | 9 | 5 | 2 | 3 | 10 | 6 | 7 | 11 |
| **Iteration-3** | 1 | 4 | 8 | 9 | 5 | 3 | 10 | 2 | 6 | 7 | 11 |
| **Iteration-4** | 1 | 8 | 4 | 9 | 5 | 3 | 10 | 2 | 6 | 7 | 11 |
| **Iteration-5** | 1 | 8 | 4 | 9 | 3 | 10 | 2 | 5 | 6 | 7 | 11 |
| **Iteration-6** | 1 | 8 | 4 | 9 | 3 | 10 | 2 | 5 | 6 | 7 | 11 |
| **Iteration-7** | 1 | 8 | 4 | 9 | 3 | 10 | 2 | 5 | 6 | 7 | 11 |
| **Iteration-8** | 1 | 8 | 4 | 9 | 3 | 10 | 2 | 5 | 6 | 7 | 11 |
| **Iteration-9** | 1 | 8 | 4 | 9 | 3 | 10 | 2 | 5 | 6 | 7 | 11 |
| **Iteration-10** | 1 | 2 | 8 | 3 | 4 | 9 | 10 | 5 | 6 | 7 | 11 |
| **Iteration-11** | 1 | 2 | 8 | 3 | 9 | 10 | 4 | 5 | 6 | 7 | 11 |
| **Iteration-12** | 1 | 8 | 3 | 9 | 2 | 10 | 4 | 5 | 6 | 7 | 11 |
| **Iteration 13** | 1 | 8 | 2 | 3 | 9 | 10 | 4 | 5 | 6 | 7 | 11 |
| **Iteration-14** | 1 | 8 | 2 | 3 | 9 | 10 | 4 | 5 | 6 | 7 | 11 |
| **Iteration-15** | 1 | 8 | 2 | 3 | 9 | 4 | 5 | 6 | 7 | 10 | 11 |
| **Iteration-16** | 1 | 8 | 2 | 3 | 9 | 4 | 5 | 6 | 7 | 10 | 11 |
| **Iteration-17** | 1 | 8 | 2 | 3 | 9 | 4 | 5 | 6 | 7 | 10 | 11 |
| **Iteration-18** | 1 | 8 | 2 | 3 | 9 | 4 | 5 | 6 | 7 | 10 | 11 |
| **Iteration-19** | 1 | 8 | 2 | 3 | 9 | 4 | 5 | 6 | 7 | 10 | 11 |
| **Iteration-20** | 1 | 8 | 4 | 2 | 5 | 9 | 3 | 10 | 6 | 7 | 11 |
| **Iteration-21** | 1 | 8 | 4 | 2 | 9 | 5 | 3 | 10 | 6 | 7 | 11 |
| **Iteration-22** | 1 | 8 | 4 | 2 | 9 | 5 | 3 | 6 | 7 | 10 | 11 |
| **Iteration-23** | 1 | 8 | 4 | 2 | 5 | 3 | 6 | 7 | 9 | 10 | 11 |
| **Iteration-24** | 1 | 4 | 8 | 2 | 5 | 3 | 6 | 7 | 9 | 10 | 11 |
| **Iteration-25** | 1 | 4 | 8 | 2 | 5 | 3 | 6 | 7 | 9 | 10 | 11 |

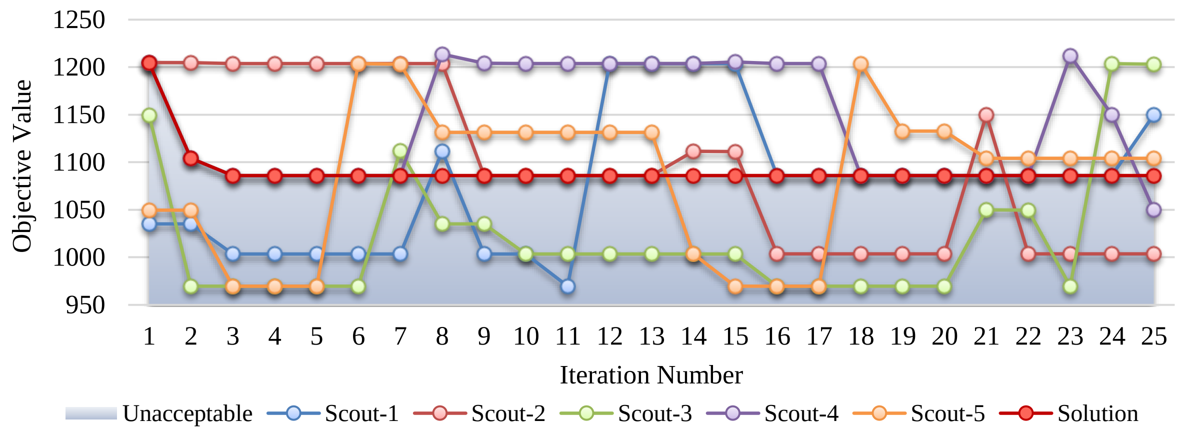


Figure 5. The movements of the scout bees

The convergence of the artificial bee colony algorithm for solving the given instance is also depicted in Figure 5 (please see '*Solution*' curve). As could be seen from the figure, the algorithm finds the best solution in only three iterations consuming a CPU time of 0.1976 s. Please note that the solution curve does not follow the minimum values for all iterations. This is why scouts find solutions with larger number of workstations, which yield lower performance values (see the shaded area in the figure). However, such solutions are discarded by the algorithm as the solutions with fewer number of workstations are favourable in real world implementations. The task assignments, the weighted-workload times, the number of workstations and the performance value (calculated using Equation (2)) of the best balancing solution are given in Table 6 along with the CPU time consumed. Five workstations are needed to perform 11 tasks under the predefined cycle time constraint.

Table 6. Best balancing solution obtained using artificial bee colony algorithm

| Task Assignment | | | | | | Number of Stations | Performance Value | CPU (s) |
|---|---|---|---|---|---|---|---|---|
| **Workstations** | Station-1 | Station-2 | Station-3 | Station-4 | Station-5 | | | |
| **Assigned Tasks** | 1,4 | 5,6,2,8 | 3 | 9,7 | 10,11 | 5 | 86.883 | 0.1976 |
| **Weighted-workloads** | 10.25 | 10.76 | 9.60 | 8.36 | 9.60 | | | |

## 4.2. Tabu search algorithm

Tabu search, defined and developed primarily by Glover (1989, 1990), is a neighbourhood search algorithm which uses effective local search procedures. It uses some taboos in neighbourhood search process to escape local optimality and has been used widely for solving complex combinatorial optimisation problems (Özcan, et al., 2010). Please refer to Glover and Laguna (1993, 1997) and Gendreau (2003) for more details on tabu search.

With motivation from successful applications of tabu search in assembly line balancing domain, a tabu search algorithm is also developed in this research as well as the artificial bee colony algorithm proposed. The steps of tabu search procedure proposed in this research are explained below through a numerical example.

Step 1. The algorithm starts by determining the algorithmic parameters and initialising the tabu lists. The *tabu size* and *the maximum number of iterations* are determined as 10 and 300, respectively, for this example.

Step 2. An initial solution is generated randomly using the same procedure to produce scout bees for artificial bee colony algorithm in the previous subsection (please see Section 0). To give an example, let us assume that the initial solution has the same task sequence (1-2-3-4-8-9-5-10-6-7-11) given in the first row of Table 3.

Step 3. A neighbourhood solution is generated. To do this, as in artificial bee colony algorithm, two random numbers ($r_1$ and $r_2$) are generated and a new neighbourhood solution is built by moving the task located at $r_1$th position to $r_2$th position. For example, let $r_1$ and $r_2$ be 2 and 5, respectively. Then, the new neighbourhood solution will have the task sequence of 1-3-4-8-2-9-5-10-6-7-11.

Step 4. The feasibility of newly generated neighbourhood solution is ensured by checking the precedence relationships matrix of the problem as well as the tabu tables used by the algorithm. The proposed tabu search algorithm incorporates two tabu tables, which are given in Table 7. The first table, given in Table 7a, is used to hold the number of iteration values during which the tasks cannot be moved to the same position again. The second table, given in Table 7b, forbids moving the tasks into their previous locations until a predefined iteration number.

Step 5. If the newly generated neighbourhood solution is not feasible, new random values are determined for $r_1$ and $r_2$ until the feasibility is maintained. An unfeasible solution becomes feasible upon the sequence of tasks in this solution satisfy the precedence relationships matrix.

Step 6. The values in the tabu tables are updated based on the move performed. In our example, the neighbourhood solution is built by moving the task placed in the second position to fifth position.

- In tabu table-1, the value of cell corresponding to the second row of fifth column is updated as $Cell\_Value = Current\_Iteration\_Number + Tabu\_Size = 1 + 10 = 11$. The cell that will be updated is determined based on the task's number (in our case it is task 2 as it is located at the second position in the task sequence) and its new location (it is five as the task is relocated to fifth position in the task sequence). Thus, it is not allowed to move task 2 into fifth position again until iteration 11.
- In tabu table-2, the value of cell on fifth row and second column is updated as $Cell\_Value = Current\_Iteration\_Number + 2 = 3$. By this way, it is prohibited to relocate task 2 in its previous position in the next iteration. Thus, the algorithm avoids getting stuck in local optima. Table 7 depicts the tabu tables after these operations.

Table 7. Tabu tables employed by the tabu search algorithm

**a) Tabu table-1**

| **Task No** | **Position** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | **11** | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **8** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **9** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **10** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **11** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**b) Tabu table-2**

| **Position** | **Position** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **8** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **9** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **10** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **11** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Step 7. The current solution and the new neighbourhood solution are compared to each other based on their performance values, which are computed as in the artificial bee colony algorithm. If the neighbourhood solution has a better performance value than the current solution, the neighbourhood solution replaces the current solution.

Step 8. This cycle continues until the maximum iteration number and the best solution is taken. Table 8 shows the best solution found by tabu search algorithm.

Table 8. The best balancing solution obtained by the tabu search algorithm

| **Task Assignment** | | | | | | **Number of Stations** | **Performance Value** | **CPU (s)** |
|---|---|---|---|---|---|---|---|---|
| **Workstations** | Station-1 | Station-2 | Station-3 | Station-4 | Station-5 | | | |
| **Assigned Tasks** | 1, 4 | 3 | 8,2,5 | 9,6 | 7,10,11 | 5 | 82.818 | 0.0175 |
| **Weighted-workloads** | 10.250 | 9.600 | 8.457 | 8.330 | 10.134 | | | |

As can be seen from the table, a total of five workstations are required to perform 11 tasks, which is the same as the artificial bee colony algorithm. However, tabu search finds a better configuration of tasks, which has a better performance value (82.818) than the performance value of the solution (86.883) obtained by the artificial bee colony algorithm.

## 5. Experimental Study

### 5.1. Test data

Twenty test problems, whose main characteristics are given in Table 9, are taken from Simaria and Vilarinho (2002) and solved using the proposed artificial bee colony algorithm and tabu search algorithm. Information regarding the precedence diagrams used for the problem set is shown in the second column. $T_j$ and $CT$ columns denote the number of tasks of the combined precedence diagram and the cycle time of the assembly line, respectively. Models demands are given in $D_j$ column. The minimum, maximum and average processing times of tasks for the considered test problems are also presented in columns $pt_{min}$, $pt_{max}$ and $pt_{avg}$.

Please note that if the processing time of a task belonging to any product model exceeds the cycle time, processing times of that task are divided into two for all product models. The reason for this modification is that parallel workstations are not allowed in the current work, different from the original study belonging to Simaria and Vilarinho (2002).

Table 9. Data for computational tests

| # | Test Problem | $T_j$ | $CT$ | $m_j$ | $D_j$ | $pt_{min}$ | $pt_{max}$ | $pt_{avg}$ |
|---|---|---|---|---|---|---|---|---|
| 1 | Bowman | 8 | 10 | A | 20 | 1.8 | 7.8 | 3.73 |
| | | | | B | 28 | 1.8 | 7.9 | 3.57 |
| 2 | Bowman | 8 | 10 | A | 16 | 0 | 10 | 3.54 |
| | | | | B | 24 | 0 | 7 | 4.40 |
| | | | | C | 8 | 0 | 10 | 5.87 |
| 3 | Gökçen and Erel (1998) | 11 | 10 | A | 20 | 1.9 | 8.8 | 4.97 |
| | | | | B | 28 | 0 | 8.7 | 3.37 |
| 4 | Gökçen and Erel (1998) | 11 | 12.5 | A | 16 | 0 | 9.6 | 3.56 |
| | | | | B | 24 | 1 | 9.6 | 4.59 |
| | | | | C | 8 | 1 | 9.6 | 4.59 |
| 5 | Mitchel | 21 | 10 | A | 20 | 0 | 9.6 | 5.29 |
| | | | | B | 28 | 0 | 9.6 | 4.46 |
| 6 | Mitchel | 21 | 10 | A | 16 | 0 | 7.5 | 4.06 |
| | | | | B | 24 | 0 | 7.5 | 4.19 |
| | | | | C | 8 | 0 | 10 | 4.68 |
| 7 | Simaria and Vilarinho | 25 | 10 | A | 20 | 0 | 9.6 | 4.74 |
| | | | | B | 28 | 0 | 9.4 | 4.35 |
| 8 | Simaria and Vilarinho | 25 | 10 | A | 16 | 0 | 9.9 | 4.37 |
| | | | | B | 24 | 1 | 10 | 4.85 |
| | | | | C | 8 | 1 | 10 | 4.87 |
| 9 | Heskiaoff | 28 | 10 | A | 20 | 0 | 10 | 5.46 |
| | | | | B | 28 | 0 | 10 | 5.75 |
| 10 | Heskiaoff | 28 | 10 | A | 16 | 0 | 10 | 5.39 |
| | | | | B | 24 | 0 | 10 | 5.53 |
| | | | | C | 8 | 0 | 10 | 5.78 |
| 11 | Sawyer | 30 | 10 | A | 20 | 0 | 9.9 | 4.49 |
| | | | | B | 28 | 0 | 9.9 | 4.46 |
| 12 | Sawyer | 30 | 10 | A | 16 | 0 | 9.9 | 4.65 |
| | | | | B | 24 | 0 | 9.9 | 4.40 |
| | | | | C | 8 | 0 | 9.9 | 4.79 |
| 13 | Lutz1 | 32 | 10 | A | 20 | 0 | 9.5 | 3.85 |
| | | | | B | 28 | 0 | 10 | 4.21 |
| 14 | Lutz1 | 32 | 10 | A | 16 | 0 | 9.7 | 4.60 |
| | | | | B | 24 | 0 | 9.5 | 4.40 |
| | | | | C | 8 | 0 | 9.7 | 4.61 |
| 15 | Gunther | 35 | 10 | A | 20 | 0 | 8.2 | 4.97 |
| | | | | B | 28 | 0 | 9 | 4.88 |
| 16 | Gunther | 35 | 10 | A | 16 | 0 | 8.7 | 4.66 |
| | | | | B | 24 | 0 | 8.7 | 4.84 |
| | | | | C | 8 | 0 | 8.8 | 4.90 |
| 17 | Kilbridge and Wester | 45 | 10 | A | 20 | 0 | 10 | 4.63 |
| | | | | B | 28 | 0 | 10 | 4.53 |
| 18 | Kilbridge and Wester | 45 | 10 | A | 16 | 0 | 9.3 | 4.64 |
| | | | | B | 24 | 0 | 9.3 | 4.61 |
| | | | | C | 8 | 0 | 9.3 | 4.24 |
| 19 | Tonge | 70 | 10 | A | 20 | 0 | 9.9 | 4.85 |
| | | | | B | 28 | 0 | 10 | 4.99 |
| 20 | Tonge | 70 | 10 | A | 16 | 0 | 9.7 | 5.02 |
| | | | | B | 24 | 0 | 9.7 | 5.02 |
| | | | | C | 8 | 0 | 9.6 | 5.03 |

## 5.2. Parameter setting

The proposed artificial bee colony algorithm and tabu search algorithm have been coded in C# environment and run on a workstation with the specifications of Intel Xeon CPU E5-2643 3.30GHz (2 processors) with 128GB RAM. The parameters of the algorithm were determined through a well-known design of experiments method, response surface methodology (RSM) to get high quality solutions (where $\beta = 100$ for all test problems).

RSM is a combination of statistical and mathematical techniques and has been used extensively to examine and characterise problems and/or processes in which the input variables (called factors) influence the outputs (called response) of the process (Bicakci, Akdas, & Deniz Karaoglan, 2014). It was proposed by Box and Wilson (1951) to determine the best combination of input parameters that minimise the output of a real non-simulated system. The main advantage of RSM is its capability to provide process optimisation by simultaneous testing of numerous factors in a limited number of experiments. This consumes less time and effort in comparison to experimenting all possible combinations of parameters one-by-one. Another advantage of RSM is that RSM provides a mathematical relation between the inputs and outputs of the system, including the interactions between the factors. Equation (4) shows the general second-order polynomial response surface model (full quadratic model) used for the experimental design (Demirtas & Karaoglan, 2012).

$$Y_u = \beta_0 + \sum_{i=1}^{n} \beta_i X_{iu} + \sum_{i=1}^{n} \beta_{ii} X_{iu}^2 + \sum_{i<j}^{n} \beta_{ij} X_{iu} X_{ju} + e_u \quad (4)$$

where $Y_u$ is the corresponding response; $\beta_0$, $\beta_i$, $\beta_{ii}$ and $\beta_{ij}$ represent the regression coefficients; $X_{iu}$ and $X_{ju}$ are coded values of the $i^{th}$ and $j^{th}$ input parameters $(i < j)$ respectively, and $e_u$ is the residual experimental error of the $u^{th}$ observation.

The model in terms of the observations may be written in matrix notation as $Y = \beta X + \varepsilon$, where $X$ and $Y$ represent input and output matrices, respectively; and $\varepsilon$ is the matrix of residuals (error term) (Montgomery, 2001). The least square estimator of $\beta$ matrix that is composed of coefficients of the regression equation is calculated as $\beta = (X'X)^{-1}X'Y$ (I. Kucukkoc, Karaoglan, et al., 2013; Yalcinkaya & Bayhan, 2009). The fitted regression models with the fitness value coefficients are formulated in the next section.

### 5.2.1. Optimisation of artificial bee colony algorithm parameters

Experiments have been conducted on a randomly selected large-sized test problem (#15) given in Section 5.1. The factor levels of artificial bee colony parameters for the experiments are listed in Table 10.

Table 10. Levels and values of parameters belonging to artificial bee colony algorithm

| Parameter | Symbol | Level | | |
|---|---|---|---|---|
| | | -1 | 0 | 1 |
| The number of scout bees | *S* | 5 | 20 | 35 |
| The number of follower bees | *F* | 5 | 15 | 25 |
| The maximum number of iterations | *Maxiter* | 50 | 175 | 300 |
| Life time | *LF* | 10 | 25 | 40 |

Table 11 shows the experimental design, detailing the experiment run order and un-coded values of the algorithm parameters. To have a more consistent analysis, the artificial bee colony algorithm was run for 5 times with the designated factor levels for each experiment. The average values of the responses (*Number of Stations* and *LB Fitness*, where *LB* means lexicographic bottleneck) are reported in Table 11.

A commercial statistical software package (Minitab-17) was used to find the coefficients matrix and establish the mathematical models for predicting the responses (namely, *Number of Stations* and *LB Fitness*). The regression equations, which depict the RSM based mathematical models representing the relations between the responses and the factors based on the results observed, are given in Equations (5) and (6) in un-coded units. Please see Figure A1 (in Appendices) for residual plots.

Table 11. Design of experiments matrix showing un-coded values and observed responses

| **Experiment No** | **Run Order** | **Factors (Un-coded Values)** | | | | **Responses (Average)** | |
|---|---|---|---|---|---|---|---|
| | | *S* | *F* | *Maxiter* | *LF* | *Number of Stations* | *LB Fitness* |
| 1 | 1 | 5 | 5 | 50 | 10 | 22.90 | 95.96 |
| 2 | 2 | 35 | 5 | 50 | 10 | 22.20 | 96.95 |
| 3 | 3 | 5 | 25 | 50 | 10 | 22.40 | 95.93 |
| 4 | 4 | 35 | 25 | 50 | 10 | 21.90 | 95.95 |
| 5 | 5 | 5 | 5 | 300 | 10 | 22.30 | 97.14 |
| 6 | 6 | 35 | 5 | 300 | 10 | 22.00 | 95.04 |
| 7 | 7 | 5 | 25 | 300 | 10 | 22.10 | 94.94 |
| 8 | 8 | 35 | 25 | 300 | 10 | 22.00 | 94.86 |
| 9 | 9 | 5 | 5 | 50 | 40 | 23.20 | 95.05 |
| 10 | 10 | 35 | 5 | 50 | 40 | 22.50 | 95.83 |
| 11 | 11 | 5 | 25 | 50 | 40 | 23.20 | 93.42 |
| 12 | 12 | 35 | 25 | 50 | 40 | 22.10 | 96.26 |
| 13 | 13 | 5 | 5 | 300 | 40 | 22.80 | 94.42 |
| 14 | 14 | 35 | 5 | 300 | 40 | 21.80 | 96.66 |
| 15 | 15 | 5 | 25 | 300 | 40 | 22.30 | 95.46 |
| 16 | 16 | 35 | 25 | 300 | 40 | 22.00 | 94.94 |
| 17 | 17 | 5 | 15 | 175 | 25 | 22.40 | 95.64 |
| 18 | 18 | 35 | 15 | 175 | 25 | 21.90 | 95.55 |
| 19 | 19 | 20 | 5 | 175 | 25 | 22.10 | 96.14 |
| 20 | 20 | 20 | 25 | 175 | 25 | 21.90 | 96.07 |
| 21 | 21 | 20 | 15 | 50 | 25 | 22.30 | 95.35 |
| 22 | 22 | 20 | 15 | 300 | 25 | 22.00 | 94.94 |
| 23 | 23 | 20 | 15 | 175 | 10 | 21.90 | 95.65 |
| 24 | 24 | 20 | 15 | 175 | 40 | 22.10 | 95.94 |
| 25 | 25 | 20 | 15 | 175 | 25 | 22.00 | 95.34 |
| 26 | 26 | 20 | 15 | 175 | 25 | 22.10 | 95.84 |
| 27 | 27 | 20 | 15 | 175 | 25 | 22.10 | 94.94 |
| 28 | 28 | 20 | 15 | 175 | 25 | 22.10 | 95.64 |
| 29 | 29 | 20 | 15 | 175 | 25 | 22.00 | 95.05 |
| 30 | 30 | 20 | 15 | 175 | 25 | 22.00 | 94.94 |
| 31 | 31 | 20 | 15 | 175 | 25 | 22.00 | 95.86 |

$$\begin{aligned} Number\ of\ Stations \\ &= 23.258 - 00490 * S - 0.0282 * F - 0.00553 * Maxiter + 0.0195 * LF \\ &+ 0.000704 * S^2 + 0.000084 * F^2 + 0.000010 * Maxiter^2 + 0.000037 * LF^2 \\ &+ 0.000292 * S * F + 0.000043 * S * Maxiter - 0.000417 * S * LF + 0.000035 \\ &* F * Maxiter + 0.000125 * F * LF - 0.000037 * Maxiter * LF \end{aligned} \tag{5}$$

$$\begin{aligned} LB\ Fitness = 97.68 &+ 0.0061 * S - 0.180 * F + 0.01096 * Maxiter - 0.1182 * LF - 0.00022 \\ &* S^2 + 0.00462 * F^2 - 0.000032 * Maxiter^2 + 0.00068 * LF^2 + 0.00015 * S * F \\ &- 0.000170 * S * Maxiter + 0.001812 * S * LF - 0.000041 * F * Maxiter \\ &+ 0.00065 * F * LF + 0.000124 * Maxiter * LF \end{aligned} \tag{6}$$

The parameter optimisation was performed with the aim of minimising *Number of Stations* and *LB Fitness* values, where the importance of responses were set to 2 and 1, respectively. The optimal un-coded process parameter setting of artificial bee colony algorithm was found as $S = 28.03$, $F = 23.18$, $Maxiter = 277.27$ and $LF = 10$ with a composite desirability of $d = 0.80$. The optimisation plot is given in Figure 6.

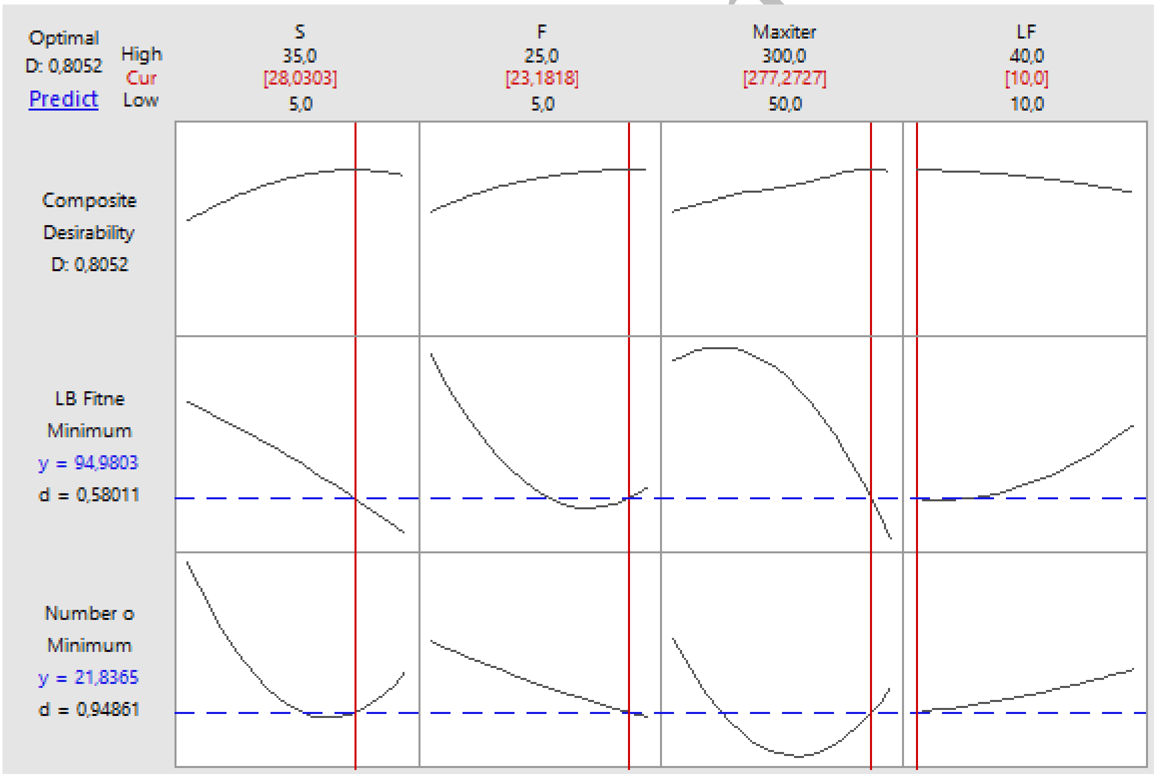


Figure 6. Optimisation results for the parameters of artificial bee colony algorithm

### 5.2.2. Optimisation of the tabu search algorithm parameters

The optimised parameters of tabu search algorithm were the *number of iterations (Maxiter)* and *tabu size*. Experiments have been conducted on the same test problem (#15) used for optimising artificial bee colony parameters. Table 12 presents the factor levels for *Maxiter* and *tabu size* while Table 13 provides experimental design, detailing the experiment run order and un-coded values of the algorithm parameters, as well as observed responses.

Table 12. Levels and values of parameters belonging to tabu search algorithm

| Parameter | Symbol | Level | | |
|---|---|---|---|---|
| | | -1 | 0 | 1 |
| The maximum number of iterations | *Maxiter* | 100 | 550 | 1000 |
| Tabu size | *Tabu size* | 10 | 20 | 30 |

As in the previous subsection, the tabu search algorithm was run for 5 times for each experiment using the designated factor values and the average values of the observed responses are reported. The regression equations, which depict the RSM based mathematical models that represent the relations between the responses and the factors based on the observed results, are given in Equations (7) and (8) in un-coded units. Please see Figure A2 (in Appendices) for residual plots.

Table 13. Design of experiments matrix showing un-coded values and observed responses

| **Experiment No** | **Run Order** | **Factors (Un-coded Units)** | | **Responses (Average)** | |
|---|---|---|---|---|---|
| | | *Maxiter* | *Tabu size* | *Number of Stations* | *LB Fitness* |
| 1 | 1 | 100 | 10 | 23.40 | 97.74 |
| 2 | 2 | 1000 | 10 | 23.00 | 94.71 |
| 3 | 3 | 100 | 30 | 23.80 | 94.72 |
| 4 | 4 | 1000 | 30 | 23.00 | 94.10 |
| 5 | 5 | 100 | 20 | 24.00 | 94.18 |
| 6 | 6 | 1000 | 20 | 22.80 | 97.36 |
| 7 | 7 | 550 | 10 | 24.20 | 89.74 |
| 8 | 8 | 550 | 30 | 23.80 | 89.50 |
| 9 | 9 | 550 | 20 | 23.00 | 94.52 |
| 10 | 10 | 550 | 20 | 23.00 | 93.90 |
| 11 | 11 | 550 | 20 | 23.00 | 95.52 |
| 12 | 12 | 550 | 20 | 22.60 | 94.52 |
| 13 | 13 | 550 | 20 | 23.40 | 94.49 |

$$\begin{aligned} Number\ of\ Stations &= 25.08 + 0.00027 * Maxiter - 0.175 * TabuSize - 0.000001 * Maxiter^2 \\ &+ 0.00469 * TabuSize^2 - 0.000022 * Maxiter * TabuSize \end{aligned} \quad (7)$$

$$\begin{aligned} LBFitness = 90.68 - 0.02111 * Maxiter + 0.978 * TabuSize + 0.000017 * Maxiter^2 - 0.0279 \\ * TabuSize^2 + 0.000134 * Maxiter * TabuSize \end{aligned} \quad (8)$$

A multi-objective optimisation analysis was performed to minimise the *Number of Stations* and *LB Fitness* value based on the developed mathematical formulation in Equations (7) and

(8). The Minitab-17 software was utilised for multi-objective optimisation with the weights of 2 and 1 for *Number of Stations* and *LB Fitness*, respectively. The optimum parameter setting was obtained as $Maxiter = 881.81$ and $Tabu\ Size = 24.54$ with a composite desirability of $d = 0.54$ (see Figure 7).

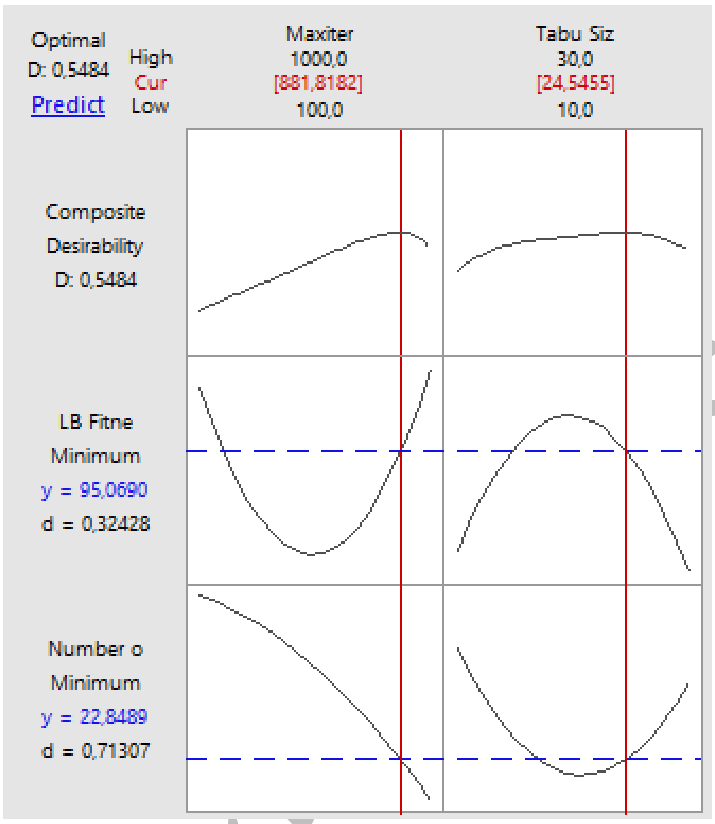


Figure 7. Optimisation results for tabu search parameters

## 5.3. Experimental test results

This section presents the experimental tests, which were conducted using the optimised values of the algorithm parameters for each solution method, namely artificial bee colony algorithm and tabu search algorithm. The decimals obtained from the RSM for the artificial bee colony and tabu search parameters have been rounded to the nearest integer value. Thus, optimum parameters were considered as $S = 28$, $F = 23$, $Maxiter = 277$ and $LF = 10$ for artificial bee colony algorithm and $Maxiter = 882$ and $Tabu\ Size = 25$ for tabu search algorithm and the test problems have been solved.

As mentioned earlier, a tabu search algorithm, as well as artificial bee colony algorithm, was also developed to provide a comprehensive experimental study. Table 14 reports the best results obtained for each test problem using artificial bee colony and tabu search algorithms. The columns $K$ and $IT$ provide the total number of workstations and the iteration number in which the best solution is found, respectively. Furthermore, the CPU time consumed to solve the relevant test problem by each algorithm is presented in the *CPU* column. The overall performance measures (*δ*) of the best solutions are also provided in Table 14 to provide comparable results for future studies.

Table 14. Computational test results

| **Problem #** | **Artificial Bee Colony** | | | | **Tabu Search** | | | |
|---|---|---|---|---|---|---|---|---|
| | ***K*** | ***δ*** | **IT** | **CPU** | ***K*** | ***δ*** | **IT** | **CPU** |
| 1 | 4 | 79.34 | 1 | 39.92 | 4 | 79.34 | 17 | 0.01 |
| 2 | 8 | 78.87 | 1 | 45.83 | 8 | 78.87 | 32 | 0.01 |
| 3 | 7 | 87.85 | 1 | 44.21 | 7 | 87.85 | 50 | 0.01 |
| 4 | 5 | 81.86 | 1 | 44.45 | 5 | 81.86 | 118 | 0.00 |
| 5 | 13 | 97.98 | 4 | 45.86 | 13 | 97.98 | 232 | 0.01 |
| 6 | 13 | 82.52 | 5 | 53.09 | 13 | 84.32 | 49 | 0.01 |
| 7 | 15 | 94.89 | 38 | 53.09 | 16 | 94.89 | 258 | 0.01 |
| 8 | 14 | 99.98 | 8 | 62.89 | 15 | 99.88 | 353 | 0.01 |
| 9 | 19 | 100.99 | 30 | 52.80 | 20 | 101.00 | 171 | 0.01 |
| 10 | 19 | 101.00 | 92 | 57.62 | 19 | 101.00 | 106 | 0.01 |
| 11 | 16 | 99.96 | 39 | 65.11 | 17 | 99.96 | 487 | 0.01 |
| 12 | 18 | 99.98 | 62 | 47.92 | 18 | 99.98 | 153 | 0.02 |
| 13 | 17 | 95.94 | 8 | 48.15 | 17 | 95.94 | 496 | 0.02 |
| 14 | 18 | 97.46 | 7 | 47.53 | 18 | 97.46 | 89 | 0.02 |
| 15 | 22 | 94.94 | 9 | 47.24 | 23 | 93.91 | 281 | 0.02 |
| 16 | 21 | 90.41 | 24 | 47.27 | 22 | 87.88 | 487 | 0.02 |
| 17 | 25 | 97.71 | 79 | 47.22 | 25 | 97.65 | 483 | 0.02 |
| 18 | 26 | 97.92 | 96 | 46.80 | 27 | 93.93 | 400 | 0.02 |
| 19 | 45 | 99.97 | 66 | 56.36 | 44 | 99.97 | 476 | 0.03 |
| 20 | 44 | 96.97 | 80 | 74.51 | 46 | 96.96 | 413 | 0.04 |

As can be seen from Table 14, artificial bee colony algorithm found solutions with fewer numbers of workstations for test problems 7-9, 11, 15, 16, 18 and 20. That is why tabu search algorithm found solutions with lower *δ* values for the same instances, except test problem 9. In this particular case, artificial bee colony algorithm outperformed tabu search in terms of the number of workstations value as well as the *δ* value. In only one test problem (test problem 19), tabu search obtained a solution with one lower workstation than the one found by artificial bee colony algorithm (where *δ* values were the same). For test problems 7 and 11, the solution gathered by artificial bee colony algorithm requires one lower workstation than the one gathered by tabu search (while *δ* values were the same for both methods). It can be said that the artificial bee colony algorithm provided more promising results in comparison with tabu search in terms of the number of workstations. On the other hand, tabu search performed better in obtaining smoother workload distributions with better *δ* values. Therefore, it can be concluded that neither of the algorithms outperformed the other one in terms of the sought performance measures.

The weighted-workloads of workstations for the solutions obtained are given in descending order in the Appendices section for both methods. As seen from the table, the weighted-workloads of workstations are quite close to each other in both solution strategies. Please note that the difference between the weighted-workloads of workstations is a bit higher in some solutions. This situation occurs if there are tasks, which require zero processing time for some product models. Another reason could be the processing times of tasks, which exceed the cycle time were divided into "two" to make the dataset suitable for the experimental study. Therefore, the compatibility of the total workload of the test problems for predefined cycle times may have been impaired.

## 6. Discussion

The methodology used in this research adopted an RSM based parameter optimisation strategy for the proposed artificial bee colony and tabu search algorithms. The variants of both artificial bee colony and tabu search algorithms have been applied successfully in solving many assembly line balancing problems in the literature. However, as known, there is no algorithm, which works perfectly for all optimisation problems.

Although artificial bee colony and tabu search algorithms are recognised with employing fewer control parameters (Dervis Karaboga & Akay, 2009), both algorithms need some preliminary work to determine the values of control parameters, as in the majority of heuristic and meta-heuristic approaches. The proposed RSM based parameter optimisation technique aims to overcome this issue and determine the best combinations of different parameter values for both algorithms. As it concurrently determines all input parameters (or factors) of the system for an optimised output (or response), RSM is more economical than conventional experimental or optimisation methods in which one input parameter at a time is optimised. Also, the regression equation produced by the RSM represents the effects for binary combinations of input parameters on the output. Furthermore, in comparison to other design of experiments techniques (including Taguchi and $2^k$ factorial design), RSM has the strength of providing optimised parameter values between the parameter levels determined by the user.

One could argue that the artificial bee colony and tabu search parameters could be optimised for each test problem individually. However, due to the page limit, the parameters have been optimised for a large-sized test problem (*i.e.* #15) and the same parameter sets have been used for solving all test problems. The truth lying behind this idea was that the parameters used for solving the large-sized problems will have the ability to solve smaller sized ones. Although this may yield extra computational time in terms of solving the small and medium-sized test problems, the CPU times reported together with the experimental results are quite desirable due to the efficient and compact structure of the developed algorithms.

## 7. Conclusions and Future Work

Different from existing traditional mostly studied types of line balancing problems in the literature, a new type of problem, called *lexicographic bottleneck assembly line balancing problem*, has been defined for the simple (straight) assembly line configuration, recently. Moving from that base point, the lexicographic bottleneck assembly line balancing problem has been handled in a mixed-model production environment, which is a more flexible but complicated version of the simple assembly line balancing system tackled in the former study.

One of the major contributions of this paper is addressing the *lexicographic bottleneck mixed-model assembly line balancing problem*. The weighted-workloads of workstations have been minimised hierarchically starting from the most heavily loaded workstation, followed by the second heavily loaded workstation and so on. Illustrative examples were given to describe the problem and explore its main characteristics. The artificial bee colony and tabu search algorithms developed for efficiently solving the problem by dealing with the variations in processing times of tasks among different product models are of major contributions of this research. The solution procedures of the algorithms were depicted using numerical examples. Furthermore, RSM has been applied to systematically optimise the parameters of the proposed algorithms for the first time in the literature. In accordance with the experimental test results, artificial bee colony algorithm found slightly better solutions than tabu search in terms of the number of workstations. However, tabu search performed better in obtaining smoother workload distributions in terms of the lexicographic bottleneck objective. It was observed that CPU times consumed by the algorithms were quite reasonable especially for the large-sized test problems.

The lexicographic bottleneck model has a wide range of application area in the assembly line balancing domain. Therefore, the proposed lexicographic bottleneck model can be applied to other types of line configurations, *e.g.* two-sided lines, U-shaped lines, parallel lines, *etc.* in future studies. New solution methods can also be developed and their performances can be compared to the performances of the algorithms developed in the current work. Developing robust models considering dynamic model demands in a lexicographic bottleneck environment may also be of interest for researchers.

## Appendices

Figure A1. Residual plots for artificial bee colony parameter optimisation

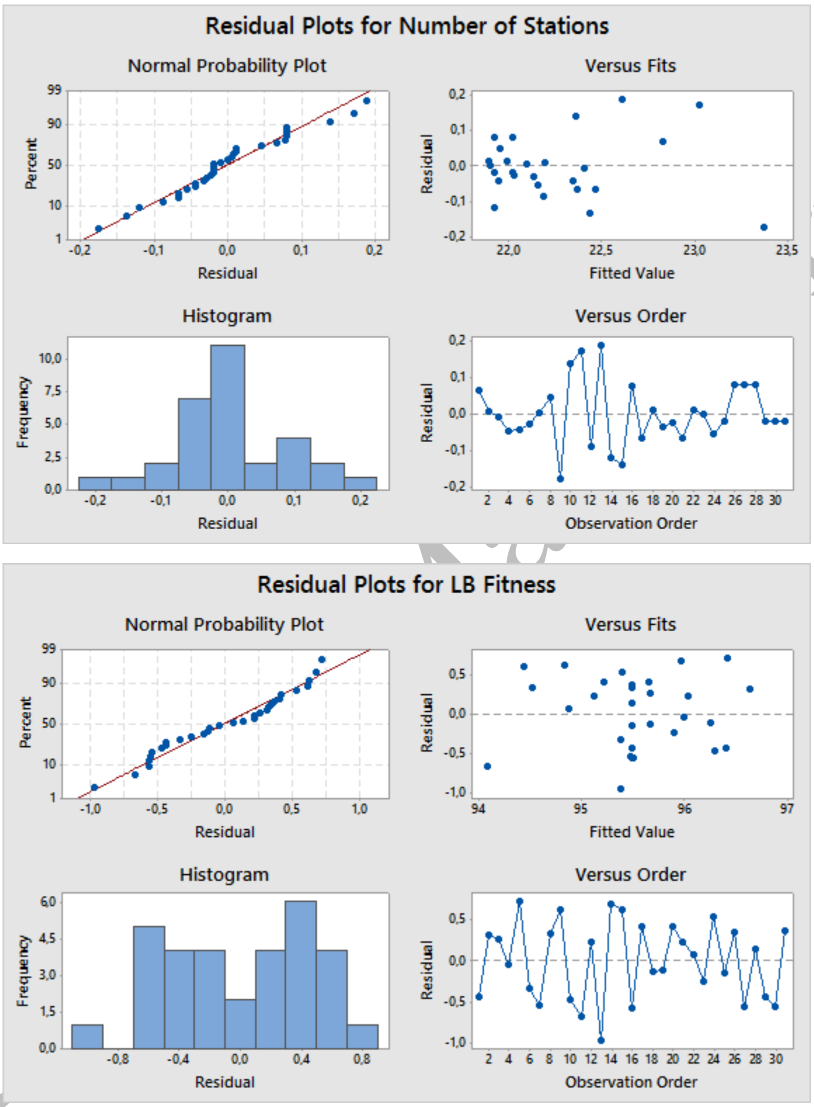
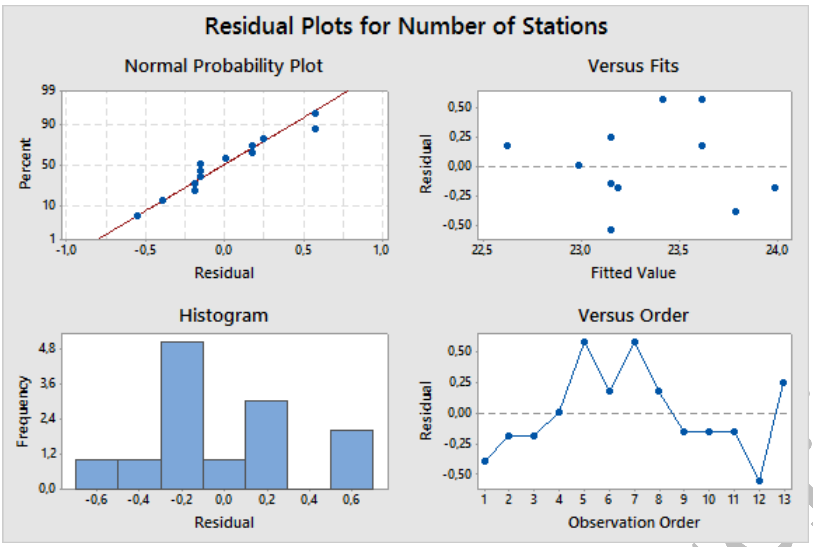


Figure A2. Residual plots for tabu search parameter optimisation



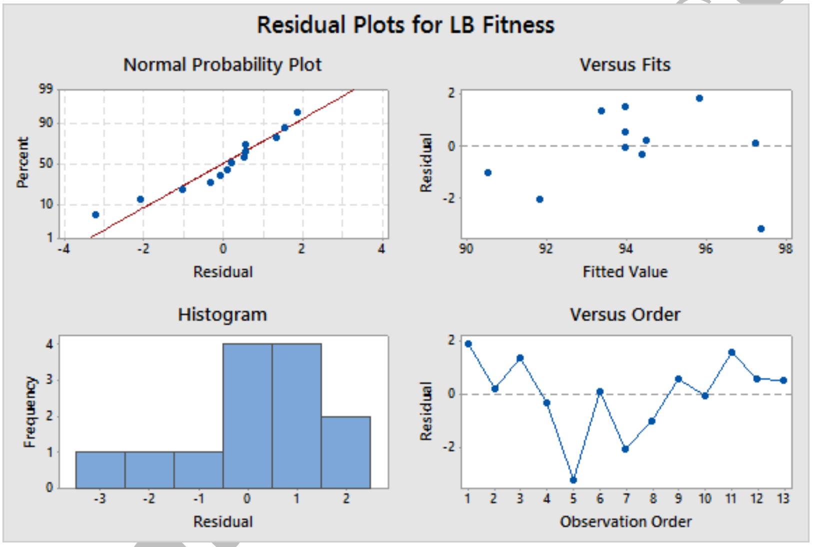


Table A1. The weighted-workloads of workstations when the problems are solved using artificial bee colony algorithm

| # | **The Weighted-workloads of Workstations in Descending Order** |
|---|---|
| 1 | 7.858 - 7.5 - 7.259 - 6.458 |
| 2 | 7.825 - 6.15 - 4.69 - 4.02 - 3.9 - 3.652 - 3 - 1.7 |
| 3 | 8.7 - 8.4 - 6.584 - 6.26 - 5.6 - 5.223 - 3.696 |
| 4 | 10.134 - 9.79 - 9.6 - 8.917 - 8.33 |
| 5 | 9.7 - 9.7 - 9.6 - 8.936 - 8.5 - 8.038 - 7.7 - 7.482 - 7.438 - 6.474 - 6.4 - 6.1 - 4.998 |
| 6 | 8.17 - 8.083 - 8 - 7.65 - 7.3 - 7 - 6.85 - 6.653 - 6.599 - 5.9 - 5.76 - 5.67 - 5.217 |
| 7 | 9.4 - 8.8 - 8.788 - 8.568 - 8.46 - 8.4 - 7.858 - 7.7 - 7.5 - 7.5 - 7.32 - 6.66 - 6.6 - 5.146 - 4.116 |
| 8 | 9.9 - 9.75 - 9.566 - 9.383 - 9.14 - 8.8 - 8.7 - 8.687 - 8.617 - 7.8 - 7.2 - 6.8 - 6.7 - 6.299 |
| 9 | 10 - 9.8 - 9.8 - 9.732 - 9.7 - 9.6 - 9.2 - 9.162 - 9.158 - 9.1 - 8.8 - 8.6 - 7.9 - 7.7 - 7.6 - 7.348 - 5.22 - 5.22 - 4.158 |

| | |
|---|---|
| 10 | 10 - 9.9 - 9.585 - 9.3 - 9.25 - 9.2 - 8.972 - 8.6 - 8.6 - 8.6 - 8.4 - 8.398 - 8 - 8 - 7 - 6.685 - 6.03 - 5.45 - 4.933 |
| 11 | 9.9 - 9.5 - 9.1 - 8.96 - 8.77 - 8.7 - 8.7 - 8.6 - 8.5 - 8.374 - 8.3 - 8.3 - 7.868 - 7.6 - 7 - 6.048 |
| 12 | 9.9 - 9.7 - 9.1 - 9.068 - 8.784 - 7.94 - 7.791 - 7.75 - 7.7 - 7.6 - 7.2 - 6.9 - 6.806 - 6.75 - 6.485 - 6.4 - 5.5 - 5.24 |
| 13 | 9.5 - 9.3 - 9.142 - 9.068 - 8.574 - 8.248 - 8.242 - 8.006 - 8 - 8 - 7.516 - 6.626 - 6.6 - 6.55 - 5.8 - 5.5 - 5.298 |
| 14 | 9.65 - 9.5 - 9.3 - 9.2 - 9.15 - 8.95 - 8.6 - 8.4 - 8.25 - 8.15 - 8.149 - 8.05 - 7.76 - 7.7 - 6.832 - 6.1 - 5.342 - 5.084 |
| 15 | 9.4 - 9.316 - 9.3 - 9 - 8.8 - 8.3 - 8.24 - 8.2 - 8.2 - 8 - 7.98 - 7.944 - 7.6 - 7.572 - 7.362 - 7.2 - 7.064 - 7 - 6.9 - 6.8 - 6.5 - 5.624 |
| 16 | 8.951 - 8.867 - 8.7 - 8.619 - 8.58 - 8.532 - 8.4 - 8.385 - 8.316 - 8.3 - 8.25 - 8.2 - 8.052 - 8 - 7.7 - 7.65 - 7.4 - 7.1 - 6.9 - 6.46 - 6.3 |
| 17 | 9.674 - 9.6 - 9.4 - 9.4 - 9.362 - 9.178 - 9.048 - 9.042 - 9 - 8.9 - 8.8 - 8.8 - 8.74 - 8.6 - 8.434 - 8.384 - 8.306 - 8.3 - 7.9 - 7.6 - 7.206 - 6.68 - 6.344 - 5.7 - 3.36 |
| 18 | 9.698 - 9.3 - 9.2 - 9 - 9 - 8.8 - 8.7 - 8.6 - 8.517 - 8.496 - 8.478 - 7.926 - 7.9 - 7.878 - 7.85 - 7.75 - 7.719 - 7.6 - 7.583 - 7.47 - 6.85 - 6.6 - 6.45 - 6.424 - 6.251 - 4.9 |
| 19 | 9.9 - 9.6 - 9.6 - 9.3 - 9.2 - 8.8 - 8.736 - 8.7 - 8.7 - 8.632 - 8.5 - 8.5 - 8.4 - 8.4 - 8.4 - 8.3 - 8.218 - 8.2 - 8.106 - 8.1 - 8.016 - 7.9 - 7.766 - 7.7 - 7.666 - 7.5 - 7.5 - 7.4 - 7.2 - 7.18 - 7.162 - 7.064 - 7.042 - 6.848 - 6.8 - 6.7 - 6.7 - 6.7 - 6.6 - 6.526 - 6.4 - 5.8 - 5.14 - 5.058 - 4.64 |
| 20 | 9.6 - 9.6 - 9.5 - 9.3 - 9.25 - 9.231 - 9.2 - 9.1 - 8.985 - 8.94 - 8.817 - 8.8 - 8.7 - 8.633 - 8.516 - 8.5 - 8.5 - 8.45 - 8.4 - 8.4 - 8.4 - 8.4 - 8.383 - 8.3 - 8.2 - 8.15 - 8.1 - 8.1 - 8.051 - 8 - 8 - 7.984 - 7.958 - 7.92 - 7.9 - 7.551 - 7.4 - 7.15 - 6.797 - 6.5 - 4.35 - 4.02 - 3.85 - 3.5 |

Table A2. The weighted-workloads of workstations when the problems are solved using tabu search algorithm

| # | **The Weighted-workloads of Workstations in Descending Order** |
|---|---|
| 1 | 7.858 - 7.5 - 7.259 - 6.458 |
| 2 | 7.825 - 6.15 - 4.69 - 4.02 - 3.9 - 3.652 - 3 - 1.7 |
| 3 | 8.7 - 8.4 - 6.584 - 6.26 - 5.6 - 5.223 - 3.696 |
| 4 | 10.134 - 9.79 - 9.6 - 8.917 - 8.33 |
| 5 | 9.7 - 9.7 - 9.6 - 8.936 - 8.5 - 8.038 - 7.738 - 7.7 - 6.682 - 6.6 - 6.474 - 6.4 - 4.998 |
| 6 | 8.35 - 8.17 - 7.498 - 7.3 - 7.25 - 7.034 - 7 - 6.85 - 6.653 - 5.9 - 5.87 - 5.76 - 5.217 |
| 7 | 9.4 - 8.8 - 8.788 - 8.46 - 8 - 7.7 - 7.5 - 7.458 - 7.146 - 7.1 - 6.774 - 5.9 - 5.5 - 5.276 - 5.146 - 3.868 |
| 8 | 9.9 - 8.7 - 8.687 - 8.583 - 8.266 - 8.14 - 7.8 - 7.8 - 7.7 - 7.45 - 7.317 - 7.2 - 6.8 - 6.7 - 6.299 |
| 9 | 10 - 9.858 - 9.8 - 9.732 - 9.2 - 9.162 - 8.9 - 8.8 - 8.7 - 8.2 - 8.1 - 7.7 - 7.6 - 7.6 - 7.3 - 6.348 - 6.2 - 5.22 - 5.22 - 4.158 |
| 10 | 10 - 9.9 - 9.585 - 9.3 - 9.25 - 9.2 - 9.048 - 8.972 - 8.6 - 8.6 - 8.6 - 8.4 - 8.217 - 8.203 - 8 - 7.968 - 6.03 - 5.5 - 1.53 |
| 11 | 9.9 - 9.5 - 9.1 - 8.7 - 8.7 - 8.568 - 8.53 - 8.5 - 8.3 - 8.3 - 7.8 - 7.716 - 7 - 6.758 - 6.616 - 6.4 - 3.832 |
| 12 | 9.9 - 9.7 - 9.251 - 9.1 - 7.734 - 7.7 - 7.64 - 7.535 - 7.518 - 7.44 - 7.2 - 7.04 - 6.9 - 6.806 - 6.75 - 6.7 - 6.4 - 5.3 |
| 13 | 9.5 - 9.3 - 9.142 - 9.068 - 9.048 - 8.574 - 8.242 - 8 - 8 - 7.516 - 7.206 - 6.626 - 6.6 - 6.55 - 5.8 - 5.5 - 5.298 |
| 14 | 9.65 - 9.5 - 9.3 - 9.2 - 9.15 - 8.95 - 8.6 - 8.4 - 8.25 - 8.166 - 8.15 - 7.76 - 7.742 - 7.7 - 6.749 - 6.1 - 5.65 - 5.15 |
| 15 | 9.3 - 9 - 8.7 - 8.3 - 8.2 - 8.2 - 7.944 - 7.756 - 7.6 - 7.58 - 7.572 - 7.5 - 7.5 - 7.4 - 7.362 - 7.2 - 7.064 - 6.9 - 6.8 - 6.5 - 6.5 - 6.126 - 5.298 |
| 16 | 8.7 - 8.7 - 8.385 - 8.369 - 8.332 - 8.316 - 8.3 - 8.28 - 8.25 - 8.2 - 8 - 7.95 - 7.703 - 7.7 - 7.6 - 7.4 - 7.1 - 6.9 - 6.3 - 5.917 - 5.9 - 5.36 |
| 17 | 9.668 - 9.6 - 9.4 - 9.4 - 9.362 - 9.178 - 9.048 - 8.9 - 8.824 - 8.8 - 8.8 - 8.706 - 8.6 - 8.442 - 8.384 - 8.314 - 8.306 - 8.3 - 8.2 - 7.9 - 7.5 - 7.4 - 6.26 - 5.106 - 3.36 |
| 18 | 9.3 - 9.2 - 9.2 - 9 - 8.909 - 8.8 - 8.7 - 8.5 - 8.498 - 8.3 - 7.9 - 7.85 - 7.75 - 7.719 - 7.6 - 7.574 - 7.5 - 7.5 - 7.5 - 7.47 - 7.128 - 6.51 - 6.424 - 6.191 - 6.017 - 4.9 - 3 |

| | |
|---|---|
| 19 | 9.9 - 9.6 - 9.6 - 9.3 - 9.2 - 8.9 - 8.862 - 8.8 - 8.758 - 8.736 - 8.7 - 8.6 - 8.516 - 8.5 - 8.5 - 8.402 - 8.4 - 8.316 - 8.216 - 8.2 - 8.148 - 8.1 - 8 - 7.9 - 7.666 - 7.58 - 7.5 - 7.5 - 7.5 - 7.48 - 7.4 - 7.366 - 7.042 - 6.864 - 6.8 - 6.8 - 6.7 - 6.606 - 6.6 - 6.526 - 6.316 - 5.8 - 5.8 - 5.3 |
| 20 | 9.6 - 9.5 - 9.25 - 9.2 - 8.9 - 8.9 - 8.9 - 8.9 - 8.885 - 8.808 - 8.8 - 8.751 - 8.516 - 8.4 - 8.4 - 8.4 - 8.383 - 8.25 - 8.2 - 8.2 - 8.173 - 8.15 - 8.1 - 8.051 - 8 - 7.9 - 7.8 - 7.6 - 7.554 - 7.4 - 7.4 - 7.221 - 7.15 - 6.797 - 6.75 - 6.7 - 6.5 - 6.5 - 6.05 - 6.03 - 5.917 - 5.8 - 5.3 - 5.2 - 4.35 - 3.85 |

## References

Akpinar, S., & Bayhan, G. M. (2011). A hybrid genetic algorithm for mixed model assembly line balancing problem with parallel workstations and zoning constraints. *Engineering Applications of Artificial Intelligence, 24,* 449-457.

Akpinar, S., Bayhan, G. M., & Baykasoglu, A. (2013). Hybridizing ant colony optimization via genetic algorithm for mixed-model assembly line balancing problem with sequence dependent setup times between tasks. *Applied Soft Computing, 13*, 574-589.

Akpinar, S., & Baykasoglu, A. (2014). Modeling and solving mixed-model assembly line balancing problem with setups. Part I: A mixed integer linear programming model. *Journal of Manufacturing Systems*, *33*, 177-187.

Battaïa, O., & Dolgui, A. (2013). A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, *142*, 259-277.

Battini, D., Faccio, M., Ferrari, E., Persona, A., & Sgarbossa, F. (2007). Design configuration for a mixed-model assembly system in case of low product demand. *International Journal of Advanced Manufacturing Technology, 34*, 188-200.

Bicakci, S., Akdas, D., & Deniz Karaoglan, A. (2014). Optimizing Karnopp friction model parameters of a pendulum using RSM. *European Journal of Control, 20*, 180-187.

Box, G. E. P., & Wilson, K. B. (1951). On the Experimental Attainment of Optimum Conditions. *Journal of the Royal Statistical Society Series B-Statistical Methodology, 13*, 1-45.

Boysen, N., Fliedner, M., & Scholl, A. (2007). A classification of assembly line balancing problems. *European Journal of Operational Research, 183*, 674-693.

Bryton, B. (1954). *Balancing of a Continuous Production Line.* M.S. Thesis, Northwestern University, Evanston, IL.

Buyukozkan, K., Kucukkoc, I., & Zhang, D. Z. (2014). Lexicographic Bottleneck Mixed-model Assembly Line Balancing Problem: an Artificial Bee Colony Approach. In *Proceedings of the 44th International Conference on Computers and Industrial Engineering (CIE44), October 14-16* (pp. 1213-1227). Istanbul.

Chutima, P., & Chimklai, P. (2012). Multi-objective two-sided mixed-model assembly line balancing using particle swarm optimisation with negative knowledge. *Computers & Industrial Engineering*, *62*, 39-55.

Demirtas, M., & Karaoglan, A. D. (2012). Optimization of PI parameters for DSP-based permanent magnet brushless motor drive using response surface methodology. *Energy Conversion and Management*, *56*, 104-111.

Esmaeilian, G. R., Sulaiman, S., Ismail, N., Hamedi, M., & Ahmad, M. M. H. M. (2011). A tabu search approach for mixed-model parallel assembly line balancing problem (type II). *International Journal of Industrial and Systems Engineering, 8*, 407.

Gendreau, M. (2003). *An Introduction to Tabu Search*: Kluwer Academic Publishers.

Glover, F. (1989). Tabu Search – Part I. *ORSA Journal on Computing,, 1*, 190-206.

Glover, F. (1990). Tabu Search – Part II. *ORSA Journal on Computing*, *2*, 4-32.

Glover, F., & Laguna, M. (1993). "Tabu Search," Modern Heuristic Techniques for Combinatorial Problems. In C. Reeves (Ed.), (pp. 70-150). Oxford Blackwell Scientific Publishing.

Glover, F., & Laguna, M. (1997). *Tabu Search*. Boston: Kluwer Academic Publishers.

Glover, F., & Marti, R. (2006). Tabu Search. In E. Alba & R. Martí (Eds.), *Metaheuristic Procedures for Training Neutral Networks* (Vol. 36, pp. 53-69): Springer US.

Goh, W. T., & Zhang, Z. (2003). An intelligent and adaptive modelling and configuration approach to manufacturing systems control. *Journal of Materials Processing Technology, 139*, 103-109.

Gökçen, H., & Erel, E. (1998). Binary integer formulation for mixed-model assembly line balancing problem. *Computers & Industrial Engineering, 34*, 451-461.

Hamzadayi, A., & Yildiz, G. (2012). A genetic algorithm based approach for simultaneously balancing and sequencing of mixed-model U-lines with parallel workstations and zoning constraints. *Computers & Industrial Engineering, 62*, 206-215.

Kara, Y., & Tekin, M. (2009). A mixed integer linear programming formulation for optimal balancing of mixed-model U-lines. *International Journal of ProductionResearch, 47*, 4201-4233.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. Technical Report TR06. In. Computer Engineering Department, Engineering Faculty, Erciyes University, Turkey.

Karaboga, D., & Akay, B. (2009). A comparative study of Artificial Bee Colony algorithm. *Applied Mathematics and Computation, 214*, 108-132.

Kucukkoc, I., Buyukozkan, K., Satoglu, S. I., & Zhang, D. Z. (2015). A mathematical model and artificial bee colony algorithm for the lexicographic bottleneck mixed-model assembly line balancing problem. *Journal of Intelligent Manufacturing*.

Kucukkoc, I., Karaoglan, A. D., & Yaman, R. (2013). Using response surface design to determine the optimal parameters of genetic algorithm and a case study. *International Journal of Production Research, 51*, 5039-5054, doi: http://dx.doi.org/5010.1080/00207543.00202013.00784411.

Kucukkoc, I., & Yaman, R. (2013). A New Hybrid Genetic Algorithm to Solve More Realistic Mixed-Model Assembly Line Balancing Problem. *International Journal of Logistics Systems and Management, 14*, 405-425.

Kucukkoc, I., & Zhang, D. Z. (2013). Balancing Parallel Two-Sided Assembly Lines via a Genetic Algorithm Based Approach. In *Proceedings of the 43rd International Conference on Computers and Industrial Engineering (CIE43)* (pp. 1-16). The University of Hong Kong, Hong Kong.

Kucukkoc, I., & Zhang, D. Z. (2014). An Agent Based Ant Colony Optimisation Approach for Mixed-Model Parallel Two-Sided Assembly Line Balancing Problem. In R. W. Grubbström & H. H. Hinterhuber (Eds.), *Pre-Prints of the Eighteenth International Working Seminar on Production Economics* (Vol. 3, pp. 313-328). Innsbruck, Austria: Congress Innsbruck.

Kucukkoc, I., & Zhang, D. Z. (2014a). Mathematical Model and Agent Based Solution Approach for the Simultaneous Balancing and Sequencing of Mixed-Model Parallel Two-Sided Assembly Lines. *International Journal of Production Economics, 158*, 314-333, doi: http://dx.doi.org/310.1016/j.ijpe.2014.1008.1010.

Kucukkoc, I., & Zhang, D. Z. (2014b). Simultaneous balancing and sequencing of mixed-model parallel two-sided assembly lines. *International Journal of Production Research, 52*, 3665-3687, doi: http://dx.doi.org/3610.1080/00207543.00202013.00879618.

Kucukkoc, I., & Zhang, D. Z. (2015). A Mathematical Model and Genetic Algorithm based Approach for Parallel Two-Sided Assembly Line Balancing Problem. *Production Planning & Control, DOI: 10.1080/09537287.2014.994685.*

Kucukkoc, I., & Zhang, D. Z. (2015). Type-E Parallel Two-Sided Assembly Line Balancing Problem: Mathematical Model and Ant Colony Optimisation based Approach with Optimised Parameters. *Computers & Industrial Engineering*.

Kucukkoc, I., Zhang, D. Z., & Keedwell, E. C. (2013). Balancing Parallel Two-Sided Assembly Lines with Ant Colony Optimisation Algorithm. In *Proceedings of the 2nd Symposium on Nature-Inspired*

*Computing and Applications (NICA) at Artificial Intelligence and the Simulation of Behaviour (AISB) 2013 Convention* (pp. 21-28). University of Exeter, Exeter, UK.

Lapierre, S. D., Ruiz, A., & Soriano, P. (2006). Balancing assembly lines with tabu search. *European Journal of Operational Research, 168*, 826-837.

Liao, L. M., Huang, C. J., & Huang, J. H. (2012). Applying Multi-agent Approach to Mixed-model Assembly Line Balancing. In *Proceedings of the IEEE ICMIT*.

Manavizadeh, N., Hosseini, N. S., Rabbani, M., & Jolai, F. (2013). A Simulated Annealing algorithm for a mixed model assembly U-line balancing type-I problem considering human efficiency and Just-In-Time approach. *Computers & Industrial Engineering, 64*, 669-685.

Manavizadeh, N., Rabbani, M., Moshtaghi, D., & Jolai, F. (2012). Mixed-model assembly line balancing in the make-to-order and stochastic environment using multi-objective evolutionary algorithms. *Expert Systems with Applications, 39*, 12026-12031.

Montgomery, D. C. (2001). *Design and Analysis of Experiments, 5th ed.* . New York: John Wiley.

Ozbakir, L., & Tapkan, P. (2011). Bee colony intelligence in zone constrained two-sided assembly line balancing problem. *Expert Systems with Applications, 38*, 11947-11957.

Ozcan, U., Cercioglu, H., Gokcen, H., & Toklu, B. (2009). A Tabu Search Algorithm for the Parallel Assembly Line Balancing Problem. *Gazi University Journal of Science, 22*, 313-323.

Ozcan, U., Cercioglu, H., Gokcen, H., & Toklu, B. (2010). Balancing and sequencing of parallel mixed-model assembly lines. *International Journal of Production Research, 48*, 5089-5113.

Ozcan, U., Kellegoz, T., & Toklu, B. (2011). A genetic algorithm for the stochastic mixed-model U-line balancing and sequencing problem. *International Journal of Production Research, 49*, 1605-1626.

Ozcan, U., & Toklu, B. (2009). Balancing of mixed-model two-sided assembly lines. *Computers & Industrial Engineering, 57*, 217-227.

Özcan, U., Gökçen, H., & Toklu, B. (2010). Balancing parallel two-sided assembly lines. *International Journal of Production Research, 48*, 4767-4784.

Özcan, U., & Toklu, B. (2008). A tabu search algorithm for two-sided assembly line balancing. *The International Journal of Advanced Manufacturing Technology, 43*, 822-829.

Pastor, R. (2011). LB-ALBP: the lexicographic bottleneck assembly line balancing problem. *International Journal of Production Research, 49*, 2425-2442.

Pastor, R., Chueca, I., & García-Villoria, A. (2012). A heuristic procedure for solving the Lexicographic Bottleneck Assembly Line Balancing Problem (LB-ALBP). *International Journal of Production Research, 50*, 1862-1876.

Pham, D. T., Koc, E., Ghanbarzadeh, A., Otri, S., Rahim, S., & Zaidi, M. (2006). The bees algorithm – a novel tool for complex optimisation problems. In *Proceedings of Innovative Production Machines and Systems Virtual Conference* (pp. 454-461).

Rabbani, M., Moghaddam, M., & Manavizadeh, N. (2012). Balancing of mixed-model two-sided assembly lines with multiple U-shaped layout. *International Journal of Advanced Manufacturing Technology, 59*, 1191-1210.

Salveson, M. E. (1955). The assembly line balancing problem. *Journal of Industrial Engineering, 6*, 18-25.

Simaria, A. S. (2006). *Assembly Line Balancing - New Perspectives and Procedures.* Universidade de Aveiro.

Simaria, A. S., & Vilarinho, P. M. (2004). A genetic algorithm based approach to the mixed-model assembly line balancing problem of type II. *Computers & Industrial Engineering, 47*, 391-407.

Simaria, A. S., & Vilarinho, P. M. (2009). 2-ANTBAL: An ant colony optimisation algorithm for balancing two-sided assembly lines. *Computers & Industrial Engineering, 56*, 489-506.

Tapkan, P., Ozbakir, L., & Baykasoglu, A. (2012a). Bees Algorithm for constrained fuzzy multi-objective two-sided assembly line balancing problem. *Optimization Letters, 6*, 1039-1049.

Tapkan, P., Ozbakir, L., & Baykasoglu, A. (2012b). Modeling and solving constrained two-sided assembly line balancing problem via bee algorithms. *Applied Soft Computing, 12*, 3343-3355.

Thomopoulos, N. T. (1967). Line Balancing-Sequencing for Mixed-Model Assembly. *Management Science, 14*, 59-75.

Thomopoulos, N. T. (1970). Mixed Model Line Balancing with Smoothed Station Assignments. *Management Science, 16*, 593-603.

Vilarinho, P. M., & Simaria, A. S. (2002). A two-stage heuristic method for balancing mixed-model assembly lines with parallel workstations. *International Journal of Production Research, 40*, 1405-1420.

Xu, W., & Xiao, T. (2011). Strategic Robust Mixed Model Assembly Line Balancing Based on Scenario Planning. *Tsinghua Science and Technology, 16*, 308-314.

Yagmahan, B. (2011). Mixed-model assembly line balancing using a multi-objective ant colony optimization approach. *Expert Systems with Applications, 38*, 12453-12461.

Yalcinkaya, O., & Bayhan, G. M. (2009). Modelling and optimization of average travel time for a metro line by simulation and response surface methodology. *European Journal of Operational Research, 196*, 225-233.

Zhang, D. Z., & Kucukkoc, I. (2013). Balancing Mixed-Model Parallel Two-Sided Assembly Lines. In L. Amodeo, A. Dolgui & F. Yalaoui (Eds.), *Proceedings of the International Conference on Industrial Engineering and Systems Management (IEEE-IESM'2013)* (pp. 391-401). Rabat, Morocco.