Scientia Iranica E (2014) 21(3), 963–979

SCIENTIA IRANICA

Sharif University of Technology
**Scientia Iranica**
*Transactions E: Industrial Engineering*
www.scientiairanica.com

# A hybrid meta-heuristic for balancing and scheduling assembly lines with sequence-independent setup times by considering deterioration tasks and learning effect

**N. Hamta[a], S.M.T. Fatemi Ghomi[a,*] R. Tavakkoli-Moghaddam[b] and F. Jolai[b]**

a. *Department of Industrial Engineering, Amirkabir University of Technology, 424 Hafez Avenue, 1591634311, Tehran, Iran.*
b. *Department of Industrial Engineering, College of Engineering, University of Tehran, Tehran, Iran.*

Received 18 April 2012; received in revised form 25 April 2013; accepted 13 August 2013

**KEYWORDS**
Assembly line balancing;
Scheduling;
Deterioration tasks;
Learning effect;
Hybrid meta-heuristic;
Sequence-independent setup times.

**Abstract.** This paper addresses the Simple Assembly Line Balancing Problem of type II (SALBP-II), with simultaneous effects of deterioration and learning in which there are sequence-independent setup times relating to each task. In many real industrial environments, although the actual task processing times are defined as a function of their starting times due to deterioration effects, workstations improve continuously as a result of repeating the same activities by worker(s) or machine(s). In this paper, a mathematical model is developed for this novel problem, attempting to minimize the cycle time for a given number of workstations. In addition to the balancing of the assembly line, the developed model presents the execution scheduling of tasks assigned to each workstation. Moreover, a hybrid meta-heuristic method is proposed to solve such an NP-hard problem. This robust and simply structured solution approach uses the tabu search within the Variable Neighbourhood Search (VNS/TS). The computational experiments and comparison with a Differential Evolution Algorithm (DEA) reflect the high efficiency of our proposed algorithm for a number of well-known instances.



## 1. Introduction

The Assembly Line Balancing Problem (ALBP) was first introduced by Bryton [1] and the first scientific study was published by Salveson [2] in this field. An assembly line consists of a set of tasks, each having a certain processing time, in the presence of a precedence relationship diagram, which designates the order of the tasks. The purpose of the assembly line balancing problem is to assign these tasks to workstations in such a way that the precedence relations are not violated and some effectiveness measures (such as cycle time, number of workstations, line efficiency or idle time) are optimized. The most well-known objective functions are to minimize the number of workstations for a given cycle time (SALBP-I) and to minimize the cycle time for a pre-defined number of workstations (SALBP-II) [3,4]. In addition to straight assembly lines, other types in the literature are U-shaped and parallel lines, which, depending on the final product, are divided into single and multi-product lines.

ALBP has been studied by many researchers for more than 50 years, and various types of line and different objective functions have been presented [5-7]. In addition, there are different surveys on ALBP in the literature [8-13]. Versions of ALBP are not only summarized in the types mentioned, and the main problem has been widely enriched under various

*. *Corresponding author. Tel.: +98 21 64545381;*
*Fax: +98 21 66954569*
*E-mail addresses: nima.hamta@aut.ac.ir (N. Hamta); fatemi@aut.ac.ir (S.M.T. Fatemi Ghomi); tavakoli@ut.ac.ir (R. Tavakkoli-Moghaddam); fjolai@ut.ac.ir (F. Jolai)*

working conditions, such as dealing with uncertainty in the parameters using the fuzzy set theory [7,13], probability theory [14], allowable task deterioration [15,16], and negligible setup times [17-19].

The SALBP has been extended considering different types of setup time, namely, SUALBSP [20], sequence dependent setup times [21,22], allowable learning effects [15,23], and general resource-constrained cases [24]. Recently, Hazır and Dolgui [25] dealt with line balancing under uncertainty. They assumed interval uncertainty for operation times and presented two robust optimization models.

The other version of SALBP is named ARALBP-I. This problem is the extended form of SALBP-I and considers the various assignment restrictions [19]. Moreover, several heuristic/meta-heuristic algorithms have been developed to obtain optimal or near optimal solutions in reasonable computational time. These algorithms could be classified as the genetic algorithm [5,17,23,26], ant colony optimization [27-29], tabu search [30], simulated annealing [14,31-34], particle swarm optimization [35], the differential evolution algorithm [36-37] and the GRASP algorithm [21]. Recently, Michalos et al. [38] introduced an intelligent search algorithm that uses three control factors to guide the search to derive assembly line design alternatives.

In the huge body of literature in the ALBP area, most studies have assumed that the task times are independent of realistic effects, such as the learning of worker(s) for repetitious tasks and deterioration. However, in many real industrial environments, the actual task processing times are a function of their starting times, due to the deterioration effect. On the other hand, workstations improve continuously because of repeating the same activities by worker(s) or machine(s). In order to reflect the real-world situation adequately, this paper adds the simultaneous effects of learning and linear deterioration to the classical SALBP. The scheduling of the execution of tasks assigned to every workstation following the balancing of the assembly line is also considered. In addition, we suppose there is a setup time relating to each task, which corresponds only to the task to be performed. This type of setup time is known as sequence-independent in related literature [39]. Due to the existence of learning and deterioration effects, setup times cannot be included in processing times. This issue is clarified in the relations that are presented later.

To the best of the authors' knowledge, there is no paper that addresses SALBP-II with the mentioned considerations. The most similar study to our work belongs to Toksarı et al. [15]. They proposed a model for SALBP-I with the effects of learning and linear deterioration without considering the scheduling of tasks and setup times. They have also adapted the COMSOAL approach for solving large-scale assembly line balancing problems with deterioration tasks and learning effects.

In this paper, we propose a hybrid meta-heuristic method to solve our NP-hard problem. Our approach utilizes the Tabu Search (TS) algorithm within the Variable Neighborhood Search (VNS). VNS is a robust solution technique that has shown an excellent performance for solving combinatorial and global optimization problems [40]. On the other hand, TS is an extremely popular method among other meta-heuristics in finding good solutions to large-scale problems. Lapierre et al. [30] presented a tabu search algorithm and evaluated its performance on the type I assembly line balancing problem from a real industrial data set, with 162 tasks and 264 precedence constraints. Liao and Cheng [41] used the combination of VNS and TS for minimizing total weighted earliness and tardiness in a single machine scheduling problem. They showed that their proposed algorithm has acceptable performance in solution quality and computation time in comparison with results obtained from the literature.

The performance of our proposed algorithm is examined over benchmark instances that are available at http://www.assembly-line-balancing.de/. The obtained results are also compared with a recently published Differential Evolution Algorithm (DEA). It is noted that DEA has shown superior performance in comparison with other evolutionary algorithms for solving SALBP-II [36-37].

The remainder of this paper has the following structure. Section 2 provides the problem description and the formulation of the model under study. Section 3 introduces the proposed hybrid algorithm in detail. Section 4 presents an experimental design in which the results obtained by the proposed hybrid algorithm are compared with those obtained by the differential evolution method. Finally, Section 5 is devoted to conclusions and some directions for future research.

## 2. Mathematical modelling of the problem

### *2.1. SALBP with sequence-independent setup times and effects of deterioration and learning*

In a Simple Assembly Line Balancing Problem (SALBP), a straight line is assumed, in which specific operations are performed on products. Several tasks allowed to process successively constitute a workstation. The maximum time of workstations is defined as Cycle Time (CT). Most existing organizations desire to maximize their production rate and optimize their assembly lines without adding new workstations.

Therefore, we prefer to use the assembly line balancing type II problem in our study, which aims to minimize the cycle time for a given number of workstations.

We assume that the processing time of a task is related to its starting time via a linear function. This concept, first presented by Browne and Yechiali [42], is known as the deterioration effect. By the effect of task deterioration, we mean that any delay in processing is penalized by incurring extra time for carrying out the task. For example, a drop in the temperature of an ingot, while waiting to be processed by a rolling machine, requires the ingot to be reheated before rolling, or the time needed to control a fire will increase, if there is a delay in fire-fighting activities. The general deterioration model is $p_i - p_0 + \alpha \times \mathrm{ST}_i$, where $p_0$ is the common basic processing time, $\alpha (\alpha > 0)$ is the deterioration rate, which is the growth rate in the task time per unit delay in its starting time, and $\mathrm{ST}_i$ is the starting time of task $i$. In addition, the log_linear curve form of the learning effect is utilized. Therefore, by considering the effects of deterioration and learning simultaneously, the actual processing time, $\hat{p}_r$, is formulated as follows:

$$\hat{p}_r = [p_r + (\alpha \times \mathrm{ST}_r)] \times r^{\beta}. \tag{1}$$

In a problem with $n$ tasks, if task $i (i = 1, 2, ..., n)$ is assigned to the $r$th sequence, $\hat{p}_r$ is its actual task time. $p_r$ and $\mathrm{ST}_r$ are the basic processing time and starting time of the task assigned to the $r$th sequence, respectively. $\beta (\beta \leq 0)$ is the learning index and is equal to the (base 2) logarithm of learning rate (i.e., $\log_2 s$) [43]. Since the deterioration effect is based on the starting time of a task, and the learning effect is based on its sequence position, Relation (1) is provided by combining these two effects. Defining $\hat{C}_{r-1}$ as the completion time of the task scheduled in the $(r-1)$th sequence, and $\mathrm{sut}_r$ as the setup time of the task in the $r$th sequence position, actual processing time, $\hat{p}_r$, can be formulated as follows:

$$\hat{p}_r = \left[p_r + \alpha \times (\hat{C}_{r-1} + \mathrm{sut}_r)\right] \times r^{\beta}. \tag{2}$$

Now, we can express $\hat{C}_{r-1}$ with the actual times and setup times of all assigning tasks. The actual completion time of the task assigned to each position is stated as:

$$\begin{aligned}
\hat{p}_1 &= \left[p_1 + \alpha \times (\hat{C}_0 + \mathrm{sut}_1)\right] \times 1^{\beta} \\
&= [p_1 + \alpha \times \mathrm{sut}_1] \times 1^{\beta},
\end{aligned}$$

$$\hat{C}_1 = \mathrm{sut}_1 + \hat{p}_1 = \mathrm{sut}_1 + [p_1 + \alpha \times \ \mathrm{sut}_1] \times 1^{\beta},$$

$$\begin{aligned}
\hat{p}_2 &= \left[p_2 + \alpha \times (\hat{C}_1 + \mathrm{sut}_2)\right] \times 2^{\beta} \\
&= \Bigg[p_2 + \alpha \times \Bigg([p_1 + \alpha \times \mathrm{sut}_1] \times 1^{\beta} \\
&\quad + \mathrm{sut}_1 + \mathrm{sut}_2\Bigg)\Bigg] \times 2^{\beta} = (p_2 \times 2^{\beta}) \\
&\quad + (\alpha \times p_1 \times 1^{\beta} \times 2^{\beta}) + (\alpha^2 \times \mathrm{sut}_1 \times 1^{\beta} \times 2^{\beta}) \\
&\quad + (\alpha \times \mathrm{sut}_1 \times 2^{\beta}) + (\alpha \times \mathrm{sut}_2 \times 2^{\beta}),
\end{aligned}$$

$$\begin{aligned}
\hat{C}_2 =& \hat{C}_1 + \mathrm{sut}_2 + \hat{p}_2 = [p_1 + \alpha \times \mathrm{sut}_1] \times 1^{\beta} + \mathrm{sut}_1 \\
&+ \mathrm{sut}_2 + (p_2 \times 2^{\beta}) + (\alpha \times p_1 \times 1^{\beta} \times 2^{\beta}) \\
&+ (\alpha^2 \times \mathrm{sut}_1 \times 1^{\beta} \times 2^{\beta}) + (\alpha \times \mathrm{sut}_1 \times 2^{\beta}) \\
&+ (\alpha \times \mathrm{sut}_2 \times 2^{\beta}) = (p_1 \times 1^{\beta}) + (p_2 \times 2^{\beta}) \\
&+ (\alpha \times p_1 \times 1^{\beta} \times 2^{\beta}) + (\alpha \times \mathrm{sut}_1 \times 1^{\beta}) \\
&+ (\alpha \times \mathrm{sut}_1 \times 2^{\beta}) + (\alpha \times \mathrm{sut}_2 \times 2^{\beta}) \\
&+ (\alpha^2 \times \mathrm{sut}_1 \times 1^{\beta} \times 2^{\beta}) + \mathrm{sut}_1 + \mathrm{sut}_2,
\end{aligned}$$

$$\begin{aligned}
\hat{p}_3 &= \left[p_3 + \alpha \times (\hat{C}_2 + \mathrm{sut}_3)\right] \times 3^{\beta} = \Bigg[p_3 + \alpha \\
&\times \Bigg(\Bigg((p_1 \times 1^{\beta}) + (p_2 \times 2^{\beta}) + (\alpha \times p_1 \times 1^{\beta} \times 2^{\beta}) \\
&\quad + (\alpha \times \mathrm{sut}_1 \times 1^{\beta}) + (\alpha \times \mathrm{sut}_1 \times 2^{\beta}) \\
&\quad + (\alpha \times \mathrm{sut}_2 \times 2^{\beta}) + (\alpha^2 \times \mathrm{sut}_1 \times 1^{\beta} \times 2^{\beta}) \\
&\quad + \mathrm{sut}_1 + \mathrm{sut}_2\Bigg) + \mathrm{sut}_3\Bigg)\Bigg] \times 3^{\beta},
\end{aligned}$$

then:

$$\begin{aligned}
\hat{p}_3 =& (p_3 \times 3^{\beta}) + (\alpha \times p_1 \times 1^{\beta} \times 3^{\beta}) \\
&+ (\alpha \times p_2 \times 2^{\beta} \times 3^{\beta}) \\
&+ (\alpha^2 \times p_1 \times 1^{\beta} \times 2^{\beta} \times 3^{\beta}) \\
&+ \left(\alpha + (\mathrm{sut}_1 + \mathrm{sut}_2 + \mathrm{sut}_3) \times 3^{\beta}\right) \\
&+ (\alpha^2 \times \mathrm{sut}_1 \times 1^{\beta} \times 3^{\beta}) \\
&+ \left(\alpha^2 \times (\mathrm{sut}_1 + \mathrm{sut}_2) \times 2^{\beta} \times 3^{\beta}\right) \\
&+ (\alpha^3 \times \mathrm{sut}_1 \times 1^{\beta} \times 2^{\beta} \times 3^{\beta}),
\end{aligned}$$

$$
\begin{aligned}
\hat{C}_3 =&\hat{C}_2 + \mathrm{sut}_3 + \hat{p}_3 = (p_1 \times 1^\beta) \\
&+ (p_2 \times 2^\beta) + (p_3 \times 3^\beta) \\
&+ (\alpha \times p_1 \times 1^\beta \times 2^\beta) \\
&+ (\alpha \times p_1 \times 1^\beta \times 3^\beta) \\
&+ (\alpha \times p_2 \times 2^\beta \times 3^\beta) \\
&+ (\alpha^2 \times p_1 \times 1^\beta \times 2^\beta \times 3^\beta) \\
&+ \mathrm{sut}_1 + \mathrm{sut}_2 + \mathrm{sut}_3 \\
&+ (\alpha \times \mathrm{sut}_1 \times 1^\beta) + (\alpha \times \mathrm{sut}_1 \times 2^\beta) \\
&+ (\alpha \times \mathrm{sut}_2 \times 2^\beta) + (\alpha \times \mathrm{sut}_3 \times 3^\beta) \\
&+ (\alpha^2 \times \mathrm{sut}_1 \times 1^\beta \times 2^\beta) \\
&+ (\alpha^2 \times \mathrm{sut}_1 \times 1^\beta \times 3^\beta) \\
&+ (\alpha^2 \times \mathrm{sut}_2 \times 2^\beta \times 3^\beta) \\
&+ (\alpha^3 \times \mathrm{sut}_1 \times 1^\beta \times 2^\beta \times 3^\beta).
\end{aligned}
$$

Therefore, continuing this procedure leads us to the following relation:

$$
\begin{aligned}
\hat{p}_r =&\left[\left(p_r + \alpha \times \sum_{v=1}^{r} \mathrm{sut}_v\right) + \sum_{h=1}^{r-1}\left(\alpha \times (p_h + \alpha \right.\right. \\
&\left.\left. \times \sum_{w=1}^{h} \mathrm{sut}_w) \times h^\beta \Pi_{q=h+1}^{r-1}(1+\alpha \times q^\beta)\right)\right] \times r^\beta.
\end{aligned} \tag{3}
$$

### 2.2. Mathematical model

Mathematical modeling is a well-known process to develop and describe a problem using mathematical concepts and languages. In this subsection, a Mixed Integer Non-Linear Programming (MINLP) model for the addressed problem is developed. To support the presentation of the proposed mathematical model, we first provide a list of notations in Table 1.

In terms of defined notations, the SALBP with sequence-independent setup times and deterioration and learning effects can be formulated as follows:

Objective function:

$$\mathrm{Min} Z = \mathrm{CT}. \tag{4}$$

Subject to:

$$\sum_{j=1}^{m}\sum_{r=1}^{\mathrm{Mn}} x_{ijr} = 1 \quad (i = 1, ..., n), \tag{5}$$

**Table 1.** List of notations.

| Notation | Definition |
|---|---|
| $i, k, s$ | Task |
| $j$ | Workstation |
| $m$ | Number of workstations |
| $r$ | Sequence position inside a workstation |
| $n$ | Number of tasks |
| $p_i$ | Basic processing time of task $i(i = 1, ..., n)$ |
| $\hat{p}_{jr}$ | Actual 1processing time of task assigned to the $r$th sequence at workstation $j$ |
| CT | Cycle time |
| $\alpha$ | Deterioration effect |
| $\beta$ | Learning effect |
| $P$ | Set of tasks that precedes a task, i.e. set of couples of tasks $(i, k)$ in which $i$ is the immediate predecessor of $k$ |
| $\mathrm{AP}_i$ | Set of all predecessors of task $i$ including non-immediate predecessors |
| Mn | Maximum number of tasks that can be assigned to any workstation |
| $\mathrm{sut}_i$ | Setup time required for task $i$ |
| $x_{ijr} \in \{0, 1\}$ | 1 if task $i$ is assigned to $r$th sequence position at workstation $j$, 0 otherwise $(i = 1, ..., n; j = 1, ..., m; r = 1, ..., \mathrm{Mn})$ |
| $y_{ikj} \in \{0, 1\}$ | 1 if task $i$ is performed immediately before task $k$ at workstation $j$ in the same or in the next cycle $(\forall j; \forall (i, l)\|(i \neq k))$ |
| $z_{ij} \in \{0, 1\}$ | 1 if task $i$ is in the last sequence position of tasks assigned to workstation $j$ $(\forall i; \forall j)$ |

$$\sum_{i=1}^{n}\sum_{r=1}^{\mathrm{Mn}} x_{ijr} \geq 1 \quad (j=1,...,m), \tag{6}$$

$$\sum_{i=1}^{n} x_{ijr} \leq 1 \quad (j=1,...,m; r=1,...,\mathrm{Mn}), \tag{7}$$

$$\sum_{i=1}^{n} x_{ij,r+1} - \sum_{i=1}^{n} x_{ijr} \leq 0$$

$$(j=1,...,m; r=1,...,\mathrm{Mn}-1), \tag{8}$$

$$\sum_{j=1}^{m}\sum_{r=1}^{\mathrm{Mn}} (\mathrm{Mn}\times(j-1)+r)\times x_{ijr}$$

$$-\sum_{j=1}^{m}\sum_{r=1}^{\mathrm{Mn}} (\mathrm{Mn}\times(j-1)+r)\times x_{kjr} \leq 0$$

$$(\forall(i,k)\in P), \tag{9}$$

$$\hat{p}_{jr} = \sum_{i=1}^{n}\Bigg[\Bigg(\Big(p_i + \alpha \times \sum_{i=1}^{n}\sum_{v=1}^{r} \mathrm{sut}_i \times x_{ijv}\Big)$$

$$+\sum_{h=1}^{r-1}\Big(\alpha\times\Big(\sum_{\tau=1}^{n}(p_\tau \times x_{\tau jr}\Big)+\alpha$$

$$\times \sum_{i=1}^{n}\sum_{w=1}^{h}(\mathrm{sut}_i \times x_{ijw})\Big)$$

$$\times h^{\beta}\Pi_{q=h+1}^{r-1}(1+\alpha\times q^{\beta})\Big)\Bigg)\times r^{\beta}\times x_{ijr}\Bigg]$$

$$(j=1,...,m; r=1,...,\mathrm{Mn}), \tag{10}$$

$$\sum_{r=1}^{\mathrm{Mn}} \hat{p}_{jr} + \sum_{i=1}^{n}\sum_{r=1}^{\mathrm{Mn}} (\mathrm{sut}_i \times x_{ijr}) \leq \mathrm{CT} \quad (j=1,...,m), \tag{11}$$

$$x_{ijr} + x_{kj,r+1} \leq 1 + y_{ikj}$$

$$(j=1,...,m; r=1,...,\mathrm{Mn}-1; \forall(i,k)|$$

$$(i\neq k)\wedge(k\notin \mathrm{AP}_i)), \tag{12}$$

$$x_{ijr} - \sum_{\substack{k=1\\(i\neq k)\wedge(k\notin \mathrm{AP}_i)}}^{n} x_{kj,r+1} \leq z_{ij}$$

$$(i=1,...,n; j=1,...,m; r=1,...,\mathrm{Mn}-1), \tag{13}$$

$$z_{ij} + x_{kj1} \leq 1 + y_{ikj}$$

$$(j=1,...,m; \forall(i,k)|(i\neq k)\wedge(i\notin \mathrm{AP}_k)). \tag{14}$$

Relation (4) presents the objective function that minimizes the cycle time. Constraint (5) shows that every task must be assigned to only one sequence in only one workstation. Constraint (6) ensures that existing workstations must be occupied with at least one task. With Constraint (7), in each sequence, inside every workstation, there will be, at most, one task assigned. Constraint (8) states that tasks should be assigned in ascending order of position in the scheduling of each workstation. Constraint (9) guarantees that the precedence relations between the tasks are not violated, regarding the assignment to different workstations and also the sequence positions inside the same workstation. Constraint (10) is the actual time of the task assigned to the $r$th sequence at workstation $j$. Constraint (11) shows that the sum of actual task times and the corresponding setup times in each workstation does not exceed the cycle time, which is going to be minimized. In other words, the total actual task times assigned to every workstation, plus the total corresponding setup times, will be equal to, or less than, cycle time. Constraint (12) assures that the binary variable, $y_{ikj}$, is equal to 1, whenever task $i$ is assigned to sequence position $r$, and task $k$ is placed in position $r+1$ in the scheduling workstation, $j$. In other words, according to Relation (12), when task $i$ and task $k$ are placed in position $r$ and $r+1$ of workstation $j$, respectively, $y_{ikj}$ must be equal to 1. Constraint (13) guarantees that the binary variable, $z_{ij}$, is 1 when task $i$ is assigned to the last position in workstation $j$, because, in this case, the summation available in the relation would be equal to 0 and, consequently, $z_{ij}$ must be set to 1. Lastly, Constraint (14) implies that the variable, $y_{ikj}$, is 1, whenever task $i$ is placed in the last position (i.e. $z_{ij}=1$) and task $k$ is assigned to the first sequence position (i.e. $x_{kj1}=1$) in the scheduling workstation, $j$.

### 2.3. A numerical example

In this subsection, the proposed MINLP model is tested using Mansoor 11, which is available in the SALBP literature. Table 2 provides processing time, predecessors and randomly generated setup times for each task. Table 3 demonstrates that considering 3 workstations, no deterioration and learning effects and using the same setup times presented in Table 2, minimum cycle time for Mansoor 11 is equal to 81.

Then, the problem under study was solved using the simultaneous effects of learning and linear deterioration. Table 4 shows that, in this case, cycle time is 81.916 when three workstations exist, learning takes place by a 70% learning curve, i.e., $\beta = log_2 0.7 = -0.515$, and the deteriorating effect ($\alpha$) is 0.15. In Tables 3 and 4, workstation time is the sum of operation and setup times. Note that we used the optimization software, LINGO 11.0, to test our proposed formulation.

**Table 2.** Task times, predecessors and setup times for Mansoor 11.

| Task no. | Task time | Predecessors | Setup time |
|---|---|---|---|
| 1 | 4 | - | 2 |
| 2 | 38 | - | 7 |
| 3 | 45 | - | 5 |
| 4 | 12 | 1, 2 | 2 |
| 5 | 10 | 2 | 3 |
| 6 | 8 | 4 | 8 |
| 7 | 12 | 5 | 8 |
| 8 | 10 | 6 | 7 |
| 9 | 2 | 7 | 4 |
| 10 | 10 | 8, 9 | 3 |
| 11 | 34 | 3, 11 | 1 |

**Table 3.** Balancing and scheduling Mansoor 11 without learning and deteriorating effects ($m = 3$).

| Workstation no. | Task | Task time + setup time | Workstation time |
|---|---|---|---|
| 1 | 1 | 6 | 78 |
| | 2 | 45 | |
| | 4 | 14 | |
| | 5 | 13 | |
| 2 | 3 | 50 | 76 |
| | 7 | 20 | |
| | 9 | 6 | |
| 3 | 6 | 16 | 81 |
| | 8 | 17 | |
| | 10 | 13 | |
| | 11 | 35 | |

**Table 4.** Balancing and scheduling Mansoor 11 with proposed model with learning and deterioration effects ($m = 3$, learning effect = 70% and deterioration effect = 0.15).

| Workstation no. | Task | Actual task time | Workstation time |
|---|---|---|---|
| 1 | 2 | 39.050 | 81.916 |
| | 5 | 12.150 | |
| | 7 | 12.716 | |
| 2 | 1 | 4.300 | 65.151 |
| | 3 | 32.687 | |
| | 9 | 5.226 | |
| | 4 | 9.938 | |
| 3 | 6 | 9.200 | 66.636 |
| | 8 | 9.541 | |
| | 10 | 8.813 | |
| | 11 | 20.082 | |

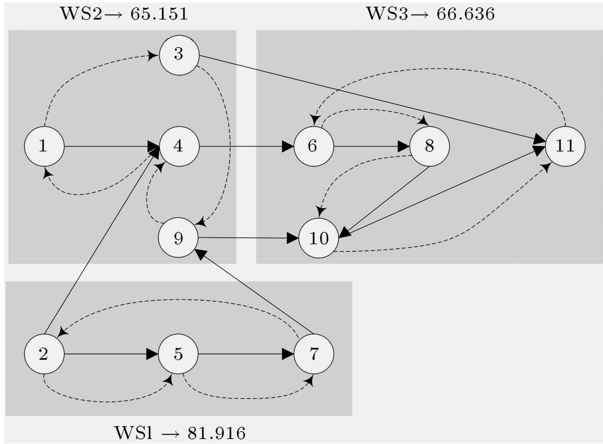


**Figure 1.** Precedence diagram for Mansoor 11 and scheduling of tasks inside each workstation.

Figure 1 presents the precedence diagram and the solution obtained for Mansoor 11, where the original precedence relationships between tasks are represented by solid arcs, and the work schedules inside three workstations are shown with discontinuous lines. As is obvious, the first workstation with the largest workstation time becomes the bottleneck in this assembly line example, with the effects of learning and deterioration and setup times.

## 3. The proposed hybrid algorithm

Gutjahr and Nemhauser [44] proved that SALBP is an NP-hard problem. They showed that ALBP can be formulated as a shortest path problem. In their approach, a network of nodes and arcs is represented in which each path corresponds to a feasible solution, and each shortest path corresponds to an optimal solution of SALBP. Since the number of nodes grows exponentially by increasing the number of tasks, it is commonly not possible to construct the complete graph. Therefore, Easton et al. [45] incorporate lower and upper bounds, as well as dominance rules, to reduce the size of the graph.

Since ALBP, even as a simple version, falls into the class of NP-hard optimization problems, more complicated versions of this problem are also known to be NP-hard, and a solution methodology should be used to solve the larger instances in reasonable computation time. In related literature, effective exact and heuristic/meta-heuristic procedures are available that solve medium-sized instances in a quality sufficient for use in real-world situations. In this regard, Scholl and Becker [11] provide a comprehensive survey of solution procedures in the SALBP field. However, further algorithmic improvement is necessary for solving the new problem raised, especially in large-scale instances. In this section, a hybrid meta-heuristic algorithm is proposed that employs the tabu search within the Variable Neighborhood Search (VNS/TS). The components of the hybrid VNS/TS algorithm are stated below.

### 3.1. Encoding scheme

There are two different representation mechanisms for the ALBPs in related literature: Task-oriented and station-oriented representations. In both of them, a string of integers is assumed with a length equal to the number of tasks to be proceeded in the assembly line. Task $i$ in position $j$ of the string, using the task-oriented representation, will be assigned to a workstation before the task in position $(j + 1)$ of the string. While, in the station-oriented scheme, task $i$ will be assigned to workstation $j$ if the $i$th location of the string has the amount of $j$. Nearchou [37] found the task-oriented scheme superior after experimentation for the SALBP-2. Because of some similarities between Nearchou's problem and our case, we decided to employ a task-oriented representation within the hybrid VNS/TS algorithm.

Our proposed solution scheme is represented by a permutation of tasks, which shows the sequence of tasks. After determining the number of tasks in each workstation, the tasks are assigned to the corresponding workstations from the left of the permutation to the right in order. Figure 2 clearly exhibits an example for this representation.

### 3.2. Initial solution

Generating a suitable solution has a significant effect on the quality of solutions and computation time. Due to the existence of precedence constraints among the tasks in ALB problems, providing an initial solution in a random manner may lead us to an infeasible solution. Checking and eliminating an infeasible permutation of tasks and substituting them for a new one will be time-consuming. So, in this paper, a simple method is used to generate a feasible initial solution that satisfies the precedence relations. The procedure is as follows:

**Step 1.** Sort the tasks based on the number of their Immediate Predecessors (IPs) in ascending order.

**Step 2.** Select a task with maximum task time plus setup time from the ones that have no predecessors (free tasks whose associated IPs are zero) and assign it to the first empty position of a string with size $n$. In case of a tie, choose a task from candidate tasks randomly.

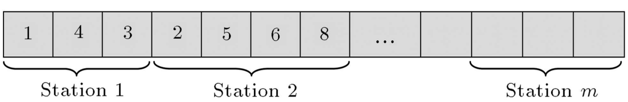


**Figure 2.** A solution representation.

**Step 3.** Delete the selected task in Step 2 from the set of tasks and update the corresponding IPs for the remaining tasks.

**Step 4.** If the set of tasks is not empty, go to Step 2.

**Step 5.** Return a permutation of $n$ tasks as a feasible initial solution.

### 3.3. Variable neighborhood search

The variable neighborhood search, first suggested by Mladenovic and Hansen [46], is a recent meta-heuristic algorithm based on the principle of systematic change of neighborhood during the search process. In other words, it employs two or more neighborhoods in its structure, instead of one. VNS is a robust, effective and simply structured method for solving combinatorial optimization problems, such as the traveling salesman problem [46], the $p$-median problem [47], the minimum spanning tree problem [48], and a large number of other successful applications, which have been reported by Hansen and Mladenovic [49].

In our algorithm, to prevent costing too much computational time, three Neighborhood Structures (NS) are considered to produce new various solutions. Index $l$ is defined to show an NS type. Whenever a neighborhood is chosen, a random procedure is employed to generate a solution (which may be infeasible) from the selected neighborhood structure. Thus, the neighborhoods, $N_1(S)$, $N_2(S)$ and $N_3(S)$, for each $l$ are created as follows:

1. $N_1(S)$ or swap operator: Swap the positions of two randomly selected different tasks. Figure 3 shows the performance of this operator graphically.
2. $N_2(S)$ or Multi Swap Operator (MSO): Repeat the swap operator more than two times, that is, after exchanging the positions of two different tasks, $i_1$ and $i_2$, choose randomly two other tasks, $ii_1$ and $ii_2$, and swap them. In this paper, we perform the swap operator twice in $N_2(S)$.
3. $N_3(S)$ or Multi Single Point Operator (MSPO): Regenerate the position of more than two randomly picked tasks. In this paper, we select two tasks, $i_1$ and $i_2$ ($i_1 \neq i_2$), and two sequence positions, $r_1$ and $r_2$, randomly. Then, task $i_1$ is transferred at sequence position $r_1$ and task $i_2$ at sequence position $r_2$. Figure 4 demonstrates the performance of this operator graphically.

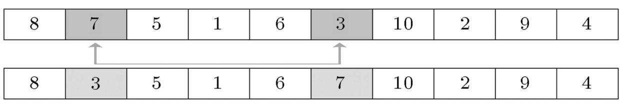


**Figure 3.** Swap operator in the proposed solution representation.

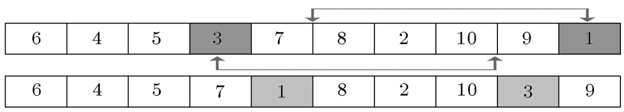


**Figure 4.** Multi single point operator in the proposed solution representation.

```
Input: The number of tasks (n), the precedence relations
       between tasks.

Output: Precedence matrix M.
Begin
   for i ← 1 to n do
     for j ← 1 to n do
       if task i must be completed before task j, then
         M_ij = 1, else, M_ij = 0
       end if
     end for
   end for
   for i ← 1 to n do
     for j ← 1 to n do
       if M_ij = 1, then
         for k ← 1 to n do
           M_kj ← M_kj + M_ki;
           if M_kj = 2, then
             M_kj ← 1;
           end if
         end for
       end if
     end for
   end for
   for j ← 1 to n do
     count ← 0;
     for i ← 1 to n do
       if M_ij = 1, then
       count ← count + 1;
       end if
     end for
     M_{n+1,j} ← count;
   end for
   Return M
End
```

**Algorithm 1.** Building precedence matrix $M$.

### *3.4. Repairing infeasible solutions*

Applying each of the NS types may provide an infeasible solution. Therefore, we need a procedure to repair generated infeasible solutions. For this purpose, a procedure similar to Nearchou [37] with a few modifications is employed. First, precedence relationships are presented in matrix $M$ in which $M_{ij} = 1$, if task $i$ must be finished immediately before task $j$, otherwise, $M_{ij} = 0$. The complete procedure for constructing the precedence matrix, $M$, is shown in Algorithm 1.

Note that indirect predecessors of each task are also represented in Algorithm 1. In other words, when task $k$ is an indirect predecessor of task $j$, the procedure sets $M_{kj} = 1$. Furthermore, the total number of the predecessors of every task is counted, and is saved in the corresponding column of the $(n+1)$th row of matrix $M$.

The most important difference between our algorithm and Nearchou's method is that in precedence matrix $M$, obtained by Algorithm 1, both direct and indirect predecessors are considered. In order to clarify the main idea behind Algorithm 1, an illustrative example is presented. In this regard, a precedence network with five tasks is given, as follows, which represents the relations between tasks. In this regard, Figure 5 shows a precedence network with 5 tasks.

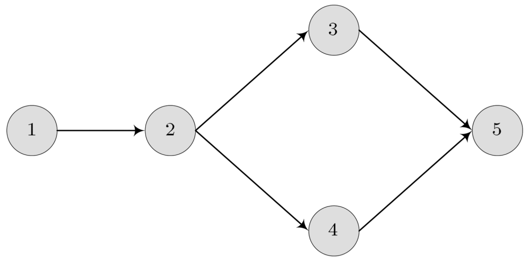


**Figure 5.** An example of a precedence network with 5 tasks.

According to Figure 5, the corresponding 0-1 connection matrix for the directed predecessor is as follows:

$$m = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Nevertheless, the precedence matrix $M$ obtained by Algorithm 1 is as follows:

$$M = \left(\begin{array}{ccccc} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 2 & 2 & 4 \end{array}\right).$$

where the last row, i.e. 6th row, presents the total number of the predecessors of each task. This information is then used in the repairing procedure illustrated by Algorithm 2.

Algorithm 2 corrects an infeasible solution, $S$, and returns the feasible version of it by string PS. Initially, PS is empty and, iteratively, each feasible task, $a$, is inserted in the next existing position to create a string of tasks, so that precedence relations are not violated.

### *3.5. Solution evaluation mechanism*

This mechanism corresponds to the calculation of the cycle time for each solution. Therefore, tasks have to be appropriately assigned to the workstations first. After experimenting with some well-known decoding schemes from the literature, we finally decided to adopt the scheme proposed by Kim et al. [51], since it was found to be superior and more adaptive, in our case. In this scheme, a feasible solution is put into an iterative process that solves the corresponding SALBP-I, with a theoretical cycle time being lessened until a near-optimum value is achieved. Due to the

**Input**: An infeasible solution, $S$, the number of tasks ($n$), precedence matrix, $M$.

**Output**: A feasible solution corresponding to $S$.

**Begin**
- $Mp \leftarrow M$; // make a copy of $M$ and work with it //
- $J \leftarrow 0$; // index of array PS (partial schedule) //
- $I \leftarrow 1$;
- **while** $J < n$ **do**
  - $a \leftarrow S(i)$;
  - **if** ($Mp_{n+1,a} = 0$) **and** ($a$ is not in array $PS$) **then**
    - $J \leftarrow J + 1$;
    - $PS\ [J] \leftarrow a$;
  - **Update** $Mp$: **for** $k \leftarrow 1$ to $n$ **do**
    - **if** $Mp_{a,k} = 1$, **then**
      - $Mp_{a,k} \leftarrow 0$;
      - $Mp_{n+1,k} \leftarrow Mp_{n+1,k} - 1$;
    - **end if**
  - **end for**
  - **end if**
  - $I \leftarrow I + 1$;
  - **if** $I > n$, **then**
    - $I \leftarrow 1$;
  - **end if**
- **end while**
- **Return** $PS$ // the repaired and feasible solution corresponding to $S$ //

**End**

**Algorithm 2.** Repairing infeasible solution ($S$).

existence of setup times, deteriorating tasks and the learning effect in our problem, we need to apply some modifications to this procedure while assigning tasks to workstations. This decoding scheme is expressed as follows:

**Step 1.** Set CT firstly equal to the theoretical minimum cycle time, i.e., CT= $\lceil \sum_{i=1}^{n} (p_i + \text{sut}_i/m) \rceil$, where $\lceil A \rceil$ denotes the smallest integer value greater than, or equal to, $A$ [34].

**Step 2.** Assign as many tasks as possible to the first $m-1$ workstations, when the sum of actual processing times is calculated using Relation (3), and their respective setup times are not more than CT for each workstation. Then, assign all the remaining tasks to the last workstation ($m$).

**Step 3.** Calculate workstation time, $\text{WT}_j$, for each workstation, $j(j = 1, 2, ..., m)$, and the potential workstation time, $\text{PWT}_j$ ($j = 1, 2, ..., m-1$), where $\text{PWT}_j = \text{WT}_j +$ sum of the actual processing time of the first task in workstation ($j+1$) and its corresponding setup time.

**Step 4.** Set $\text{CT}_w = \text{Max}\{\text{WT}_1, \text{WT}_2, ..., \text{WT}_m\}$ and $\text{CT} = \text{Min}\ \{\text{PWT}_1, \text{PWT}_2, ...., \text{PWT}_{m-1}\}$.

**Step 5.** If ($\text{CT}_w > \text{CT}$), then go to Step 2.

**Step 6.** Return $\text{CT}_w$ as the minimum value of the cycle time and stop.

It is noteworthy that the initial CT is obtained in Step 1. Then, the value of CT is updated in Step 4. Therefore, after Step 5, if we return to Step 2, the assignment of tasks to the workstations is undertaken based on the new CT.

### 3.6. Tabu search

Tabu Search (TS) is a decent meta-heuristic algorithm originally introduced by Glover [52], which avoids the trap of local optimum by allowing a non-improving move. TS has frequently been used to find good solutions to large-sized combinatorial problems in many industrial applications. TS starts from an initial solution and moves to a better solution in its neighborhood through the search space, until a specific number of iterations has been completed with no improvement in the best solution found so far. To avoid cycling back to previously visited solutions and trapping them a local optimum, a Tabu List (TL) is used to record the recent moves. The length of the tabu list is one of the most significant parameters in a TS algorithm, and is required to be determined accurately. In this paper, we apply TS in combination with VNS for solving our problem. In other words, TS is employed in each type of NS of VNS algorithm to improve the search process.

The proposed hybrid algorithm structure, with all its aforementioned features, is designed as illustrated in Algorithm 3.

## 4. Experimental design and discussion

In this section, we are going to examine the performance of the VNS/TS hybrid algorithm over standard sets of benchmark instances taken from the literature. The efficiency of the proposed algorithm is compared with the Differential Evolution Algorithm (DEA) of

```
Begin
Step 1. Initialization
    Input required data set;
    Initialize parameters: Basic processing times, sequence independent setup (for each task),
    number of workstations, deterioration effect (α), learning effect (β), length of tabu list; max number of
    iterations (Max_Iter), max number of neighborhood searches for each solution (Max_NS), l ← 1;
    Build matrix M to present precedence relations; // Using Algorithm 1 //
    CTN ← M; // CTN corresponding to the first neighborhood; M is a long integer value//
    S ← Generate initial solution; // Using the procedure presented in Subsection 3.2 //
    CT_best ← Calculate CT for solution S; // Using the decoding scheme introduced in Subsection 3.5 //
    Add S to tabu list (TL);
Step 2. Balancing and scheduling
    while Max number of iterations < Max_Iter do
        for Number of neighborhood searches ← 1 to < Max_NS do
            SN ← Generate a solution from the lth neighborhood of S (SN ∈ N_l(S));
            SN ← Repair infeasible solution (SN); // Using Algorithm 2 //
            CT_SN ← Calculate cycle time for SN using decoding scheme;
            if SN is not in tabu list, then
                Update TL; // add SN to TL, push all other entries in TL one position down and delete the entry at
                            the bottom of TL //,
            else
            CT_SN ← M; // M is a too large value //
            end if
            if CT_SN < CTN, then
               CTN ← CT_SN;
               SN_best ← SN;
            end if
        end for
        if CTN < CT_best , then
            CT_best ← CTN;
            S ← SN_best;
            l ← 1;
        else,
            l ← l + 1;
    end while
Step 3. Report optimization results.
End
```

**Algorithm 3.** Main body of hybrid algorithm.

Nearchou [37], which was originally used for solving SALBP-II.

All tests are implemented in MATLAB 7.9 and run on a personal computer with 2.10 GHz Intel Core 2 Due CPU and 3 GB of RAM memory, under a Microsoft Windows XP environment.

### *4.1. Data settings*

The required data for our problem includes: the number of tasks, the number of workstations, precedence network, processing times, deterioration effect, learning effect and setup times. The basic processing times and the precedence constraints for all instances are available in the assembly line optimization research homepage (http://www.assembly-line-balancing.de). We solve the problems for two given numbers of workstations from a set of $\{3, 4, 6\}$. The deterioration effect is set at 0.10 and 0.15 and learning rates are chosen from the set of $\{60\%, 70\%, 80\%\}$. The setup times are also uniformly distributed from 1 to the mean of the basic processing times.

### *4.2. Evaluation metrics*

After computing the objective value (cycle time) for each instance using the algorithms, the Relative Percentage Dviation (RPD), in percentages, is calculated by the following relation [53]:

$$\mathrm{RPD}(\%) = \frac{\mathrm{Algorithm}_{\mathrm{sol}} - \mathrm{Min}_{\mathrm{sol}}}{\mathrm{Min}_{\mathrm{sol}}} \times 100, \tag{15}$$

where $\mathrm{Algorithm}_{\mathrm{Sol}}$ is the objective value of each algorithm for a given instance, and $\mathrm{Min}_{\mathrm{Sol}}$ is the best solution obtained for each instance by any of two algorithms. An average RPD equal to 3%, generated by a specific algorithm, means that this algorithm is 3% over the best obtained solution, on average. As is obvious, lower RPD values are preferred.

### *4.3. Parameters tuning*

It is clear that the various levels of the parameters affect the quality of the solutions obtained by a hybrid algorithm. Selecting the best combination of parameters can intensify the search process and prevent the neighborhood search from being trapped in local optimum. Thus, we have applied parameter tuning for the maximum number of neighborhood searches for each solution (Max_NS) and the Length of the Tabu List (LTL). In this study, trials are performed considering three different sizes of instance: small, medium, and large. Table 5 shows the factor levels for each type of instance. In order to avoid over-fitting in the computational results, the various instances are

**Table 5.** Factor levels for different types of instances.

| Instances | Number of tasks | Max_ NS levels | LTL levels |
|---|---|---|---|
| Small | Lower than 20 | 20, 30, 40 and 50 | 5, 7 and 10 |
| Medium | Between 20 and 50 | 30, 50, 70 and 100 | 7, 10 and 12 |
| Large | Greater than 50 | 50, 100, 200 and 300 | 10, 15 and 20 |

**Table 6.** Two-way ANOVA results for RPD.

| **ANOVA table for medium size instances** | | | | | |
|---|---|---|---|---|---|
| **Source** | **DF** | **Sum of squares** | **Mean squares** | **F** | **P-value** |
| Max_NS | 3 | 0. 9919 | 0.3306 | 0.99 | 0.459 |
| LTL | 2 | 0.4879 | 0.2439 | 0.73 | 0.520 |
| Max_NS * LTL | 6 | 2.0053 | 0.3342 | 1.49 | 0.200 |
| Error | 48 | 10.7372 | 0.2237 | | |
| Total | 59 | 14.2222 | | | |
| **ANOVA table for large size instances** | | | | | |
| **Source** | **DF** | **Sum of squares** | **Mean squares** | **F** | **P-value** |
| Max_NS | 3 | 0.1246 | 0.0415 | 0.72 | 0.575 |
| LTL | 2 | 0.2199 | 0.1100 | 1.91 | 0.228 |
| Max_NS * LTL | 6 | 0.3457 | 0.0576 | 0.52 | 0.789 |
| Error | 48 | 5.2954 | 0.1103 | | |
| Total | 59 | 5.9856 | | | |

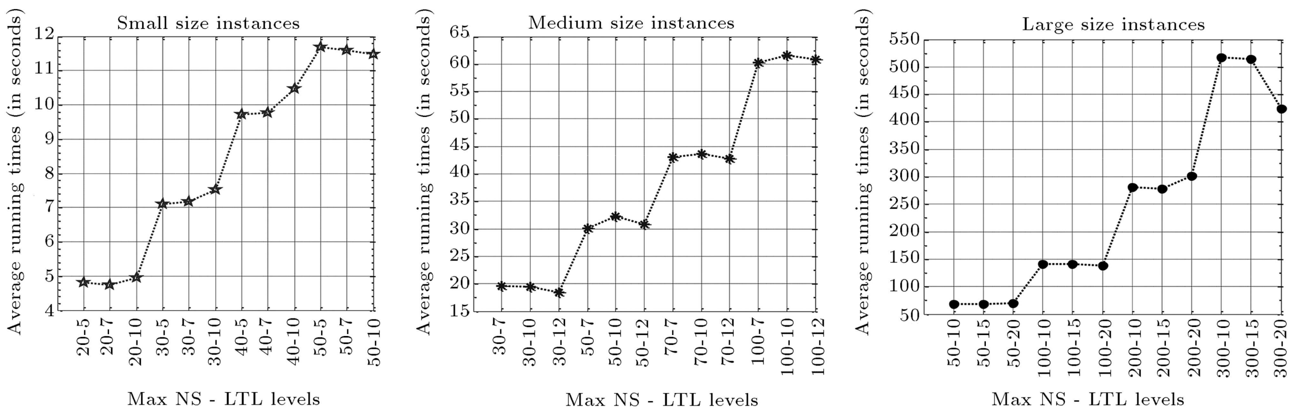


**Figure 6.** Average running times (in CPU seconds) for two factors and three sizes of instances.

employed for tuning the parameters and evaluating the proposed algorithm.

Since there are two factors to be tuned, we use the two-way analysis of variance (ANOVA) technique to analyze the obtained results. It is noteworthy that to employ ANOVA, three main hypotheses, namely, normality, homogeneity of variance and independence of residuals, must be checked. We performed that, and found no bias for questioning the validity of the experiment. All instances are solved by the VNS/TS algorithm 5 times by $3 \times 4$ different combinations. For small size instances, the algorithm leads to near-optimal solutions, with an RPD of 0.0%. So, it is not necessary to perform an ANOVA test for this size of instance, and the best levels of factors are selected on the basis of the least average running time. Table 6 summarizes the ANOVA results for medium- and large-sized instances.

As reported in Table 6, there is no significant difference between the levels of factors, because the p-values are greater than $\alpha$-level when the alpha is set at 0.05 (or even 0.1). Therefore, it is just only possible to select better levels of each factor based on lower average running times. Figure 6 illustrates the

**Table 7.** Average relative percentage deviation ($\overline{\text{RPD}}$) for two algorithms with deterioration effect = 0.10.

| Problem name | Problem size ($n$) | Number of workstations ($m$) | Algorithm | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | Learning rate = 60% | | Learning rate = 70% | | Learning rate = 80% | |
| | | | VNS/TS | DEA | VNS/TS | DEA | VNS/TS | DEA |
| MITCHELL | 21 | 3 | 0.000 | 0.000 | 0.000 | 0.000 | **0.000** | 2.998 |
| | | 4 | **0.328** | 0.437 | 0.000 | 0.000 | 0.000 | 0.000 |
| | | 5 | **0.257** | 0.318 | **0.026** | 0.069 | **0.040** | 0.112 |
| ROSZIEG | 25 | 4 | **1.997** | 4.980 | **0.173** | 0.777 | **0.601** | 1.234 |
| | | 5 | **0.577** | 4.362 | **0.069** | 4.158 | **0.070** | 4.006 |
| | | 6 | **0.695** | 1.089 | **0.518** | 1.619 | **0.118** | 2.784 |
| HESKIA | 28 | 4 | **0.034** | 5.699 | **0.213** | 1.335 | **0.553** | 1.940 |
| | | 5 | **0.041** | 2.231 | **1.160** | 3.008 | **1.587** | 2.894 |
| | | 6 | **0.125** | 3.279 | **1.492** | 3.442 | **1.403** | 3.882 |
| LUTZ1 | 32 | 4 | **0.781** | 1.017 | **0.000** | 0.281 | **0.830** | 2.208 |
| | | 5 | **0.000** | 1.703 | **1.278** | 3.467 | **0.924** | 1.917 |
| | | 6 | **0.020** | 1.221 | **1.732** | 2.762 | **0.871** | 1.721 |
| GUNTHER | 35 | 4 | **0.817** | 2.570 | **0.467** | 1.281 | 0.790 | 2.381 |
| | | 5 | **0.450** | 1.854 | **0.021** | 1.059 | 0.851 | 1.262 |
| | | 6 | **0.372** | 1.592 | **0.201** | 2.206 | 0.692 | 2.452 |
| KILBRIDGE | 45 | 4 | **0.899** | 3.292 | **0.674** | 1.452 | **1.278** | 3.058 |
| | | 5 | **0.473** | 4.561 | **0.577** | 1.593 | **1.120** | 4.042 |
| | | 6 | **0.312** | 1.883 | **0.662** | 3.789 | **0.717** | 3.852 |
| WARNECKE | 58 | 4 | **1.269** | 6.926 | **0.222** | 1.023 | **0.000** | 0.043 |
| | | 5 | **1.293** | 3.458 | **0.469** | 1.587 | **0.019** | 0.693 |
| | | 6 | **1.002** | 3.674 | **0.451** | 3.313 | **0.022** | 0.283 |
| TONG | 70 | 4 | **1.114** | 2.903 | **1.002** | 3.902 | **1.583** | 3.639 |
| | | 5 | **1.231** | 3.308 | **1.023** | 3.409 | **0.941** | 2.695 |
| | | 6 | **1.209** | 3.225 | **0.937** | 3.741 | **0.924** | 3.343 |
| WEE-MAG | 75 | 4 | **0.533** | 2.262 | **0.339** | 1.002 | **0.060** | 0.926 |
| | | 5 | **0.398** | 2.192 | **0.609** | 1.424 | **0.248** | 1.409 |
| | | 6 | **0.314** | 1.809 | **1.394** | 7.673 | **0.809** | 3.595 |
| LUTZ2 | 89 | 4 | **0.611** | 2.043 | **0.328** | 1.329 | **0.692** | 1.655 |
| | | 5 | **0.548** | 2.094 | **0.481** | 1.835 | **0.237** | 1.995 |
| | | 6 | **0.211** | 1.516 | **0.574** | 1.793 | **0.060** | 1.444 |
| ARCUS | 111 | 4 | **0.783** | 2.011 | **0.323** | 2.437 | **0.077** | 1.629 |
| | | 5 | **0.678** | 2.092 | **0.442** | 1.935 | **0.651** | 2.516 |
| | | 6 | **1.001** | 2.990 | **0.554** | 2.965 | **0.412** | 2.120 |
| Average | | | **0.617** | 2.563 | **0.558** | 2.172 | **0.581** | 2.143 |

mean running times (in CPU seconds) spent for three predefined sizes of instance. The results display that there is a significant difference in computational time between various levels of the Max_NS factor for each size of instance and this factor has a great effect on the running time. Thus, in order to decrease the running time, it is helpful to choose the lowest value for Max_NS in each group of instances. On the other hand, selecting the better level of the LTL factor in each category of instances pertains to the decision maker, because results indicate that this factor has no major influence on the RPD values and running times.

### *4.4. Experimental results*

In this subsection, the efficiency and effectiveness of our proposed algorithm are evaluated by setting various rates of deterioration and learning. The obtained results are in terms of cycle time as the objective function. As stated before, we compare the proposed VNS/TS with the differential evolution algorithm. For evaluating the performance of these two methods, the RPD measure is used. Tables 7 and 8 show the results of experiments for two deterioration rates (0.10 and 0.15), each one grouped by $n$, $m$ and learning rate. Note that each instance is solved using five different

**Table 8.** Average relative percentage deviation ($\overline{\text{RPD}}$) for two algorithms with deterioration effect = 0.15.

| Problem name | Problem size ($n$) | Number of workstations ($m$) | Algorithm: Learning rate = 60% | | Learning rate = 70% | | Learning rate = 80% | |
|---|---|---|---|---|---|---|---|---|
| | | | VNS/TS | DEA | VNS/TS | DEA | VNS/TS | DEA |
| MITCHELL | 21 | 3 | 0.000 | 0.000 | **0.450** | 2.439 | **0.209** | 0.651 |
| | | 4 | 0.532 | 0.532 | 0.000 | 0.000 | **0.855** | 1.139 |
| | | 5 | **0.239** | 0.381 | **0.001** | 0.210 | **0.682** | 0.921 |
| ROSZIEG | 25 | 4 | **1.257** | 6.267 | **1.187** | 2.895 | **0.932** | 1.985 |
| | | 5 | **1.230** | 3.118 | **0.038** | 8.554 | **0.050** | 5.235 |
| | | 6 | **1.012** | 2.691 | **0.192** | 1.833 | **0.219** | 1.128 |
| HESKIA | 28 | 4 | **0.154** | 5.808 | **0.636** | 3.103 | **1.134** | 1.954 |
| | | 5 | **0.807** | 3.675 | **0.239** | 1.570 | **1.525** | 4.321 |
| | | 6 | **0.932** | 2.741 | **0.478** | 1.846 | **1.461** | 4.232 |
| LUTZ1 | 32 | 4 | **1.867** | 3.197 | **0.928** | 1.436 | **0.126** | 1.889 |
| | | 5 | **0.737** | 3.131 | **0.691** | 1.345 | **0.000** | 0.602 |
| | | 6 | **0.561** | 2.451 | **0.403** | 2.019 | **0.390** | 0.815 |
| GUNTHER | 35 | 4 | **1.914** | 6.128 | **1.353** | 2.775 | **0.679** | 2.636 |
| | | 5 | **0.435** | 2.755 | **0.000** | 1.609 | **0.365** | 1.374 |
| | | 6 | **0.598** | 1.862 | **0.327** | 1.968 | **0.421** | 1.681 |
| KILBRIDGE | 45 | 4 | **1.790** | 4.335 | **0.244** | 1.443 | **1.605** | 3.980 |
| | | 5 | **0.519** | 3.761 | **0.291** | 1.763 | **0.451** | 2.091 |
| | | 6 | **0.478** | 3.469 | **1.163** | 3.547 | **0.020** | 1.839 |
| WARNECKE | 58 | 4 | **1.122** | 3.097 | **0.000** | 0.070 | **0.000** | 0.114 |
| | | 5 | **0.581** | 2.341 | **0.345** | 1.391 | **0.230** | 1.094 |
| | | 6 | **0.386** | 1.629 | **0.311** | 2.166 | **0.547** | 4.418 |
| multirow3*TONG | 70 | 4 | **0.109** | 1.691 | **0.491** | 1.091 | **0.901** | 2.813 |
| | | 5 | **0.611** | 2.918 | **0.386** | 1.901 | **0.178** | 1.912 |
| | | 6 | **0.617** | 1.561 | **0.471** | 1.871 | **0.382** | 1.832 |
| WEE-MAG | 75 | 4 | **1.539** | 6.397 | **0.650** | 1.387 | **0.697** | 2.028 |
| | | 5 | **1.297** | 3.910 | **0.494** | 2.103 | **0.450** | 2.901 |
| | | 6 | **1.749** | 4.609 | **0.417** | 2.635 | **0.635** | 6.040 |
| LUTZ2 | 89 | 4 | **0.349** | 1.173 | **0.366** | 2.682 | **0.340** | 4.097 |
| | | 5 | **0.591** | 1.091 | **0.671** | 2.190 | **0.492** | 1.671 |
| | | 6 | **0.467** | 1.962 | **0. 841** | 4.364 | **1.752** | 6.574 |
| ARCUS | 111 | 4 | **0.316** | 3.261 | **0.617** | 2.125 | **0.603** | 3.729 |
| | | 5 | **0.410** | 2.915 | **0.312** | 2.461 | **0.495** | 2.853 |
| | | 6 | **0.084** | 2.294 | **0.301** | 3.409 | **0.193** | 2.055 |
| Average | | | **0.766** | 2.944 | **0.452** | 2.188 | **0.576** | 2.503 |

seeds and the average solution is considered. As can be seen, our hybrid VNS/TS algorithm provides better results than DEA. In order to analyze the results more precisely and to verify which algorithm is better, statistically, we carried out an ANOVA test, where two algorithms and RPD values were considered as the factor and response variables, respectively. The means plot and LSD intervals (at 95% confidence level) are shown in Figures 7 and 8 for two algorithms. As can be seen, there is a clear significant difference between the results of the two algorithms, and our proposed VNS/TS shows statistically better performance than DEA for learning rate $\in \{60\%, 70\%, 80\%\}$ and deterioration effect $\in \{0.10, 0.15\}$. We also evaluated the

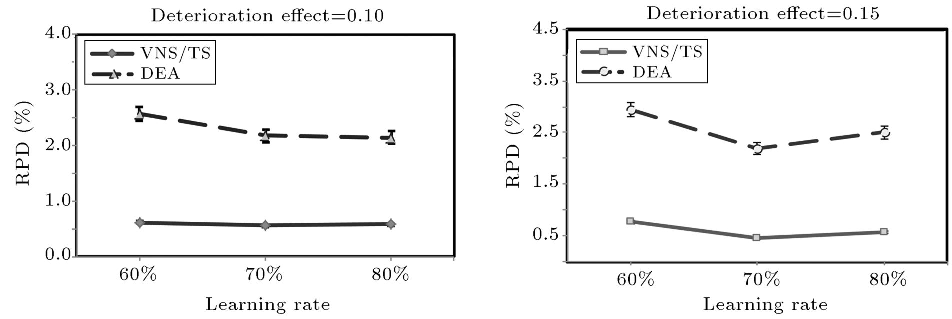


**Figure 7.** Plots of $\overline{\text{RPD}}$ versus learning rates for the type of algorithm factor grouped by deterioration effect.

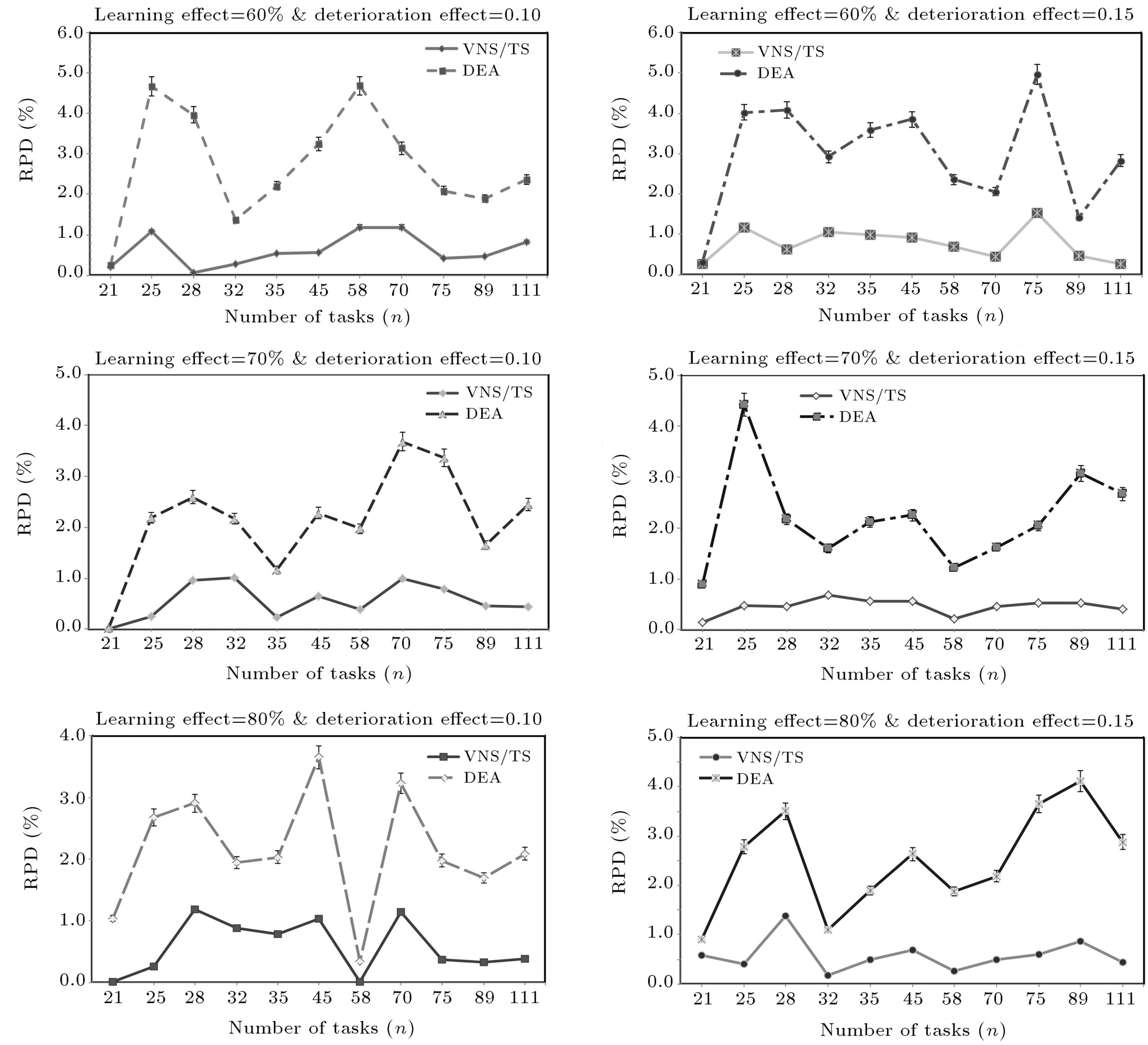


**Figure 8.** Plots of $\overline{\text{RPD}}$ versus number of tasks for the type of algorithm grouped by learning and deterioration effects.

effect of a different number of tasks on the performance of the algorithms (Figure 8). As shown, VNS/TS works better than DEA in all cases.

## 5. Conclusions and future study

In the literature of the assembly line balancing problem, most studies have assumed that the task times are independent of realistic effects, such as the learning of worker(s) for repetition tasks, and deterioration. However in many real-world environments, the task processing times are a function of their starting times due to the deterioration effect. On the other hand, workstation efficiency improves continuously because of repeating the same activities by worker(s) or machine(s). Hence, in order to reflect the industrial situation adequately, this paper investigates the simultaneous effects of learning and linear deterioration on the simple assembly line balancing and scheduling problem. Furthermore, we suppose there is a sequence-independent setup time relating to each task, which corresponds only to the task to be performed. A Mixed Integer Non-Linear Programming (MINLP) model was developed to minimize the cycle time for a pre-defined number of workstations. We also proposed a hybrid meta-heuristic algorithm, called VNS/TS, comprising two components: the Variable Neighborhood Search (VNS) and the Tabu Search (TS) in which TS was employed within defined neighborhood structures. Finally, the performance of the VNS/TS algorithm was evaluated over a standard set of benchmark instances taken from the literature. The obtained results were compared with a Differential Evolution Algorithm (DEA). The experimental results demonstrate that our algorithm performs superior to DEA and obtains better results in terms of solution quality.

In order to enrich the current work, other assumptions, such as $U$-shaped, two-sided lines, parallel stations and equipment selection, can be considered in the assembly line balancing and scheduling problem. Addressing the sequence-dependent setup time between tasks, and developing a Mixed Integer Linear Programming (MILP) model would be interesting future research lines. On the other hand, application of an exact solution technique (e.g., branch and bound method) or other efficient meta-heuristic algorithms to solve relatively large-scale instances is a challenging area for future study.

## Acknowledgements

The authors would like to thank anonymous referees for their valuable and helpful comments. The suggestions have strongly improved the presentation and clarification of earlier version of the paper.

## References

1. Bryton, B. "Balancing of a continuous production line", MS Thesis, Northwestern University, Evanston, Illinois (1954).
2. Salveson, M.E. "The assembly line balancing problem", *Journal of Industrial Engineering*, **6**, pp. 18-25 (1955).
3. Becker, C. and Scholl, A. "A survey on problems and methods in generalized assembly line balancing", *European Journal of Operational Research*, **168**(31), pp. 694-715 (2006).
4. Pastor, R. and Ferrer, L. "An improved mathematical program to solve the simple assembly line balancing problem", *International Journal of Production Research*, **47**(11), pp. 2943-2959 (2009).
5. Levitin, G., Rubinovitz, J. and Levitin, B.S. "A genetic algorithm for robotic assembly line balancing", *European Journal of Operational Research*, **168**, pp. 811-825 (2006).
6. Kim, Y.K., Song, W.S. and Kim, J.H. "A mathematical model and a genetic algorithm for two-sided assembly line balancing", *Computers & Operations Research*, **36**, pp. 853-865 (2009).
7. Kara, Y., Paksoy, T. and Chang, C.T. "Binary fuzzy goal programming approach to single model straight and U-shaped assembly line balancing", *European Journal of Operational Research*, **195**, pp. 335-347 (2009).
8. Ghosh, S. and Gagnon, R.J. "A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems", *International Journal of Production Research*, **27**, pp. 637-670 (1989).
9. Erel, E. and Sarin, S.C. "A survey of the assembly line balancing procedures", *Production Planning and Control*, **9**, pp. 414-434 (1998).
10. Rekiek, B., Dolgui, A., Delchambre, A. and Bratcu, A. "State of art of optimization methods for assembly line design", *Annual Reviews in Control*, **26**, pp. 163-174 (2002).
11. Scholl, A. and Becker, C. "State-of-the-art exact and heuristic solution procedures for simple assembly line balancing", *European Journal of Operational Research*, **168**(3), pp. 666-693 (2006).
12. Battaïa, O. and Dolgui, A. "A taxonomy of line balancing problems and their solution approaches", *International Journal of Production Economics*, **142**(2), pp. 259-277 (2013).
13. Ozcan, U. and Toklu, B. "Multiple-criteria decision-making in two-sided assembly line balancing: A goal programming and a fuzzy goal programming models", *Computers & Operations Research*, **36**(6), pp. 1955-1965 (2009).
14. Ozcan, U. "Balancing stochastic two-sided assembly lines: A chance-constrained, piecewise-linear, mixed integer program and a simulated annealing algorithm", *European Journal of Operational Research*, **205**(1), pp. 81-97 (2010).
15. Toksarı, M.D., Isleyen, S.K., Güner, E. and Baykoç, O.F. "Assembly line balancing problem with deterioration tasks and learning effect", *Expert Systems with Applications*, **37**, pp. 1223-1228 (2010).
16. Shahanaghi, K., Yolmeh, A.M. and Bahalke, U. "Scheduling and balancing assembly lines with the task deterioration effect", *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, **224**, pp. 1145-1153 (2010).
17. Chen, R.S., Lu, K.Y. and Yu, S.C. "A hybrid genetic algorithm approach on multi-objective of assembly planning problem", *Engineering Applications of Artificial Intelligence*, **15**, pp. 447-457 (2002).
18. Agpak, K. and Gokcen, H. "Assembly line balancing: Two resource constrained cases", *International Journal of Production Economics*, **96**, pp. 129-140 (2005).

19. Scholl, A., Fliedner, M. and Boysen, N. "ABSALOM: Balancing assembly lines with assignment restrictions", *European Journal of Operational Research*, 200, pp. 688-701 (2010).

20. Watkins, R.E. and Cochran, J.K. "A line balancing heuristic case study for existing automated surface mount assembly line setups", *Computers & Industrial Engineering*, **29**, pp. 681-685 (1995).

21. Andres, C., Miralles, C. and Pastor, R. "Balancing and scheduling tasks in assembly lines with sequence-dependent setup times", *European Journal of Operational Research*, **187**, pp. 1212-1223 (2008).

22. Martino, L. and Pastor, R. "Heuristic procedures for solving the general assembly line balancing problem with setups", *International Journal of Production Research*, **48**(6), pp. 1787-1804 (2010).

23. Tasan, S.O. and Tunali, S. "A review of the current applications of genetic algorithms in assembly line balancing", *Journal of Intelligent Manufacturing*, **19**, pp. 49-69 (2008).

24. Corominas, A., Ferrer, L. and Pastor, R. "Assembly line balancing: General resource-constrained case", *International Journal of Production Research*, **49**(12), pp. 3527-3542 (2011).

25. Hazır, Ö. and Dolgui, A. "Assembly line balancing under uncertainty: Robust optimization models and exact solution method", *Computers & Industrial Engineering*, **65**(2), pp. 261-267 (2013).

26. Hamta, N., Fatemi Ghomi, S.M.T., Jolai, F. and Bahalke, U. "Bi-criteria assembly line balancing by considering flexible operation times", *Applied Mathematical Modelling*, **35**(12), pp. 5592-5608 (2011).

27. Vilarinho, P.M. and Simaria, A.S. "ANTBAL: An ant colony optimization algorithm for balancing mixed-model assembly lines with parallel workstations", *International Journal of Production Research*, **44**(2), pp. 291-303 (2006).

28. Simaria, A.S. and Vilarinho, P.M. "2-ANTBAL: An ant colony optimization algorithm for balancing two-sided assembly lines", *Computers & Industrial Engineering*, **56**, pp. 489-506 (2011).

29. Ozbakir, L., Baykasoglu, A., Gorkemli, B. and Gorkemli, L. "Multiple-colony ant algorithm for parallel assembly line balancing problem", *Applied Soft Computing*, **11**(3), pp. 3186-3198 (2011).

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

31. Erel, E., Sabuncuoglu, I. and Aksu, B.A. "Balancing of U-type assembly systems using simulated annealing", *International Journal of Production Research*, **39**(13), pp. 3003-3015 (2001).

32. Mohammadi, G. and Ozbayrak, M. "Scheduling mixed-model final assembly lines in JIT manufacturing", *International Journal of Computer Integrated Manufacturing*, **19**(4), pp. 377-382 (2006).

33. Seyed-Alagheband, S.A., Fatemi Ghomi, S.M.T. and Zandieh, M. "A simulated annealing algorithm for balancing the assembly line type II problem with sequence-dependent setup times between tasks", *International Journal of Production Research*, **49**(3), pp. 805-825 (2011).

34. Roshani, A., Fattahi, P., Roshani, A., Salehi, M. and Roshani, A. "Cost-oriented two-sided assembly line balancing problem: A simulated annealing approach", *International Journal of Computer Integrated Manufacturing*, **25**(8), pp. 689-715 (2012).

35. Hamta, N., Fatemi Ghomi, S.M.T., Jolai, F. and Akbarpour Shirazi, M. "A hybrid PSO algorithm for a multi-objective assembly line balancing problem with flexible operation times, sequence-dependent setup times and learning effect", *International Journal of Production Economics*, **141**(1), pp. 99-111 (2013).

36. Mozdgir, A., Mahdavi, I., Seyedi Badeleh, I. and Solimanpur, M. "Using the Taguchi method to optimize the differential evolution algorithm parameters for minimizing the workload smoothness index in simple assembly line balancing", *Mathematical and Computer Modelling*, **57**(1-2), pp. 137-151 (2013).

37. Nearchou, A.C. "Balancing large assembly lines by a new heuristic based on differential evolution method", *International Journal of Advanced Manufacturing Technology*, **34**, pp. 1016-1029 (2007).

38. Michalos, G., Makris, S. and Mourtzis, D. "An intelligent search algorithm-based method to derive assembly line design alternatives", *International Journal of Computer Integrated Manufacturing*, **25**(3), pp. 211-229 (2012).

39. Allahverdi, A. "Two-machine flowshop scheduling problem to minimize total completion time with bounded setup and processing times", *International Journal of Production Economics*, **103**, pp. 386-400 (2006).

40. Mladenovic′, N., Drazic′, M., Kovacevic-Vujcic′, V. and Cangalovic′, M. "General variable neighborhood search for the continuous optimization", *European Journal of Operational Research*, **191**(3), pp. 753-770 (2008).

41. Liao, C.J. and Cheng, C.C. "A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date", *Computers & Industrial Engineering*, **52**, pp. 404-413 (2007).

42. Browne, S. and Yechiali, U. "Scheduling deteriorating jobs on a single processor", *Operations Research*, **38**, pp. 495-498 (1990).

43. Badiru, A.B. "Computational survey of univariate and multivariate learning curve models", *IEEE Transactions on Engineering Management*, **39**, pp. 176-188 (1992).

44. Gutjahr, A.L. and Nemhauser, G.L. "An algorithm for the line balancing problem", *Management Science*, **11**, pp. 308-315 (1964).

45. Easton, F., Faaland, B., Klastorin, T.D. and Schmitt, T. "Improved network based algorithms for the assembly line balancing problem", *International Journal of Production Research*, **27**, pp. 1901-1915 (1989).

46. Mladenovic, N. and Hansen, P. "Variable neighborhood search", *Computers & Operations Research*, **24**, pp. 1097-1100 (1997).

47. Hansen, P. and Mladenovic, N. "Variable neighborhood search for the p-median", *Location Science*, **5**, pp. 207-226 (1997).

48. Ribeiro, C.C. and Souza, M.C. "Variable neighborhood search for the degree-constrained minimum spanning tree problem", *Discrete Applied Mathematics*, **118**, pp. 43-54 (2002).

49. Hansen, P. and Mladenovic, N. "Variable neighborhood search: Principles and applications", *European Journal of Operational Research*, **130**, pp. 449-467 (2001).

50. Behnamian, J. and Fatemi Ghomi, S.M.T. "Hybrid flowshop scheduling with machine and resource-dependent processing times", *Applied Mathematical Modelling*, **35**, pp. 1107-1123 (2011).

51. Kim, Y.K., Kim, Y.J. and Kim, Y. "Genetic algorithms for assembly line balancing with various objectives", *Computers & Industrial Engineering*, **30**(3), pp. 397-409 (1996).

52. Glover, F. "Future paths for integer programming and links to artificial intelligence", *Computers & Operations Research*, **13**, pp. 533-549 (1986).

53. Roshanaei, V., Naderi, B., Jolai, F. and Khalili, M. "A variable neighborhood search for job shop scheduling with set-up times to minimize makespan", *Future Generation Computer Systems*, **25**, pp. 654-661 (2009).

## Biographies

**Nima Hamta** completed his B.Sc. in Industrial Engineering at Iran University of Science and Technology, Tehran, in 2009 and M.Sc. in Industrial Engineering at Amirkabir University of Technology, Tehran, Iran in 2011. Currently, he is a Ph.D. student in Industrial Engineering at Amirkabir University of Technology, Tehran, Iran. His research interests are operations research, mathematical modeling, production scheduling and supply chain planning.

**Seyyed Mohammad Taghi Fatemi Ghomi** was born in Ghom, Iran on 11 March 1952. He received his B.Sc. degree in industrial engineering from Sharif University, Tehran in 1973, and the Ph.D. degree in industrial engineering from University of Bradford, England, in 1980. He worked as planning and control expert in the group of construction and cement industries, a group of Organization of National Industries of Iran during the years 1980-1983. Also, he founded the department of industrial training in the aforementioned organization in 1981. He joined to the Amirkabir University of Technology in Tehran, Iran as a faculty member in 1983. He is the author and co-author of more than 200 technical papers and the author of six books in the area of Industrial Engineering topics. He has supervised 125 M.Sc. and 15 PhD theses. His research and teaching interests are in stochastic activity networks, production planning, scheduling, queueing theory, statistical quality control, and time series analysis and forecasting. He is currently Professor in the Department of Industrial Engineering at the Amirkabir University of Technology, Tehran, Iran. Amirkabir University of Technology recognized him as one of the best researchers of the years 2004 and 2006. Also, Ministry of Science and Technology recognized him as one of the best Professor of Iran for the year 2010.

**Reza Tavakkoli-Moghaddam** is a professor of Industrial Engineering at College of Engineering, University of Tehran in Iran. He obtained his Ph.D. in Industrial Engineering from the Swinburne University of Technology in Melbourne (1998), his M.Sc. in Industrial Engineering from the University of Melbourne in Melbourne (1994) and his B.Sc. in Industrial Engineering from the Iran University of Science & Technology in Tehran (1989). He serves as Editorial Boards of the International Journal of Engineering, Iranian Journal of Operations Research, Production & Operations Management (in Farsi), and Journal of Industrial Engineering (in Farsi). Also, he is the member of the Academy of Sciences in Iran. He is the recipient of the 2009 and 2011 Distinguished Researcher Award and the 2010 Distinguished Applied Research Award at University of Tehran, Iran. He has been selected as National Iranian Distinguished Researcher for 2008 and 2010. Professor Tavakkoli-Moghaddam has published 3 books, 12 book chapters, more than 570 papers in reputable academic journals and conferences.

**Fariborz Jolai** is currently a professor of Industrial Engineering at College of Engineering, University of Tehran, Tehran, Iran. He obtained his Ph.D. degree in Industrial engineering from INPG, Grenoble, France in 1998. He completed his B.Sc. and M.Sc. in Industrial Engineering at Amirkabir University of Technology, Tehran, Iran. His current research interests are scheduling and production planning, supply chain modeling, optimization problems under uncertainty conditions.