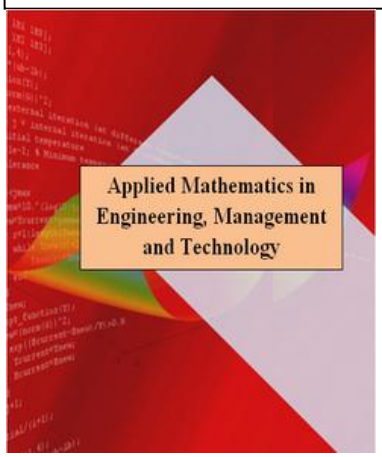
Applied mathematics in Engineering, Management and Technology 1 (3) 2013: 190-209
www.amiemt-journal.com

Balancing and scheduling U shaped assembly lines with the task deterioration effect

Maryam Shabani[a]
Department of Industrial Engineering, Faculty of Industrial and Mechanical Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran.



Abstract

Traditionally in assembly lines, the stations are arranged consecutively along a straight line and tasks are assigned to the stations while moving forward through a precedence network. However recently as a consequence of using just in time principles, many lines are arranged in a U shaped manner. U shaped line configuration provides more flexibility in assigning tasks to the stations. Therefore, this configuration results in more balanced and shorter lines.
On the other hand in many realistic situations in assembly lines delaying the starting time of a task may increase its processing time. This phenomenon is called task deterioration effect. Usually in assembly lines it is assumed that the sequence in which the tasks are executed has no effect on the processing time of tasks. However, not considering this effect may result in increasing the cycle time.
In this paper the effect of task deterioration on a U shaped line is considered and an integer linear programming (ILP) formulation is proposed to solve the problem. Furthermore a genetic algorithm (GA) and a particle swarm optimization (PSO) are proposed to solve the problem in a reasonable amount of time. In order to illustrate the proposed model and algorithms, several instances are solved using the proposed model and meta-heuristic algorithms.

Keywords: Assembly line; U shaped line; Task deterioration; Task scheduling, Genetic algorithm, particle swarm optimization;

1 Introduction

When the production levels of a specific commodity is sufficiently high, it is advantageous to use a specific production line to produce that commodity. This production line consists of some workstations through which the work-piece progresses. The work-piece remains at each station for C time units (cycle time); during this time a group of tasks are performed on the work-piece, then the work-piece proceeds to the next station. This procedure continues until the complete product is produced. This production system is called assembly line.
The most studied problem in the field of assembly line research is called simple assembly line balancing problem (SALBP). SALBP can be described as follows: given a list of tasks, the time required to perform each task and the precedence constraints between tasks, partition these tasks into stations such that either:
(1) For a given cycle time the number of stations is minimized (SALBP-1).
(2) For a given number of stations, the cycle time is minimized (SALBP-2).
Assembly line balancing problem is first introduced by (Salveson, 1955). SALBP has many simplifying assumptions that make it impractical for real world assembly line systems. Therefore many researchers in recent years have focused on identifying and modeling practical situations in assembly lines. The resulting problems are called generalized assembly line balancing problems (GALBP). GALBPs are mainly obtained by releasing some of the unrealistic assumptions of SALBP. Some examples of these GALBPs are considering resource constraints (Agpak & Gokcen, 2005; Corominas et al., 2010), considering setup times between tasks (Andres et al., 2008; Seyed-Alagheband et al., 2011; Martino & Pastor, 2009), considering other objective functions that are more interesting in real world assembly lines (Pastor, 2011), rebalancing assembly lines (Grangeon et al., 2011), parallel workstations (Buxey, 1974), considering U-shaped assembly lines balancing (Miltenburg & Wijngaard, 1994), considering process alternatives (Pinto et al., 1983), and two sided assembly lines (Bartholdi, 1993). Some of the latest surveys of assembly line balancing problems are (Erel & Sarin, 1998; Scholl, 1999; Rekiek et al., 2002; Scholl & Becker, 2006).

Traditionally the assembly line is arranged in a straight line layout. In this configuration beginning from the first tasks in the precedence graph, tasks are assigned to stations while moving forward through the precedence graph. In this type of line the worker handles one work-piece in each station.
However many real-world assembly line systems are arranged in U shaped lines rather than straight lines. In a U shaped line tasks from different parts of the precedence network can be assigned to one station. In this type of line there are two work-pieces handled by the worker in each station. Figure 1 shows an example of a simple U shaped line. As seen in this figure, the first task of one work-piece, which is starting to be produced, and the last tasks on another work-piece, which is on the verge of completion, have been assigned to the first station (WS1).
In a straight line configuration a task can be assigned to a station only when all of its predecessors have been assigned to an earlier station. But in a U shaped line configuration a task can be assigned to a station when all of its predecessors or all of its successors have been assigned to an earlier station. Since U shaped line provides more flexibility in assigning tasks to stations, it may result in better line balances than the traditional straight line.

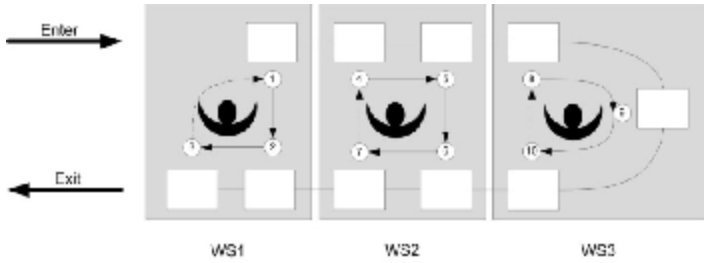


Figure 1 A simple U line

Some advantages of the U shaped line configuration over the straight lines are better proximity of operators; easier rebalancing of the line and better efficiency in terms of number of stations and cycle time (Miltenburg, 1998).
In most studies in the field of assembly line research the duration of task is assumed to be fixed and doesn't change through time. However in some real-world situations the processing time of a task increases if the task is performed later. This effect is called deterioration and firstly introduced by (Gupta & Gupta, 1988) and (Browne & Yechiali, 1990), in the field of scheduling. They assumed that processing time of each job is a linear function of its starting time. One of the popular industrial situations for deterioration is temperature of an ingot, which maybe dropped below the desired level while waiting for the rolling machine, the more the ingot waits the more time is required to reheat it.
Typically in the literature of assembly line research, it is assumed that, inside each station, Tasks can be performed in any feasible sequence without affecting processing time of each other. However in practice, the sequence in which tasks are performed may have a significant effect on the processing time of each other due to existence of task deterioration effect. Therefore lack of an efficient schedule among tasks in each station may result in significant increase in cycle time.
Even though the task deterioration effect is a very important concept in assembly lines, there have been few research papers that take this effect into account. (Toksarı et al., 2010) considered machine and worker deterioration and learning effect simultaneously in assembly lines. They proposed a mixed non-linear integer programming formulation for this model and also discussed adaption of the COMSOAL for large scale instances of the proposed model. (Shahanaghi et al., 2009), introduced the task deterioration effect in assembly lines. They proposed a mathematical formulation to solve the proposed model. They also proposed a Genetic algorithm (GA) to solve the instances of large size.
In this paper the task deterioration effect in U shaped assembly line is taken into consideration, which to the best of the author's knowledge hasn't been considered so far. A mathematical formulation is developed to solve the problem under consideration. Furthermore a GA and a particle swarm optimization (PSO) are proposed to solve the problem under consideration.
The rest of this paper is organized as follows: in section 1 the proposed model is illustrated and a mathematical formulation is proposed to solve the model. In section 2 and 3 the proposed GA and PSO are explained respectively. Section 4 presents the computational experiments for the mathematical model and the proposed

algorithms. Finally the main conclusions of the paper and further research suggestions in this area are presented in section 5.

2 Problem description and mathematical formulation

In this paper the U shaped assembly line balancing problem with the effect of task deterioration is considered, which hasn't been considered in the literature so far. This model adds up the effect of task deterioration to the U shaped assembly line balancing problem (*UALBP*). Specifically the second type of this problem is taken into account which is minimizing the cycle time *C* for a given number of stations *m*. As said before, each worker handles two work-pieces in each station. One work-piece moves forward through the line and the other one moves backward through the line. The tasks scheduled to be performed on the first work-piece build the forward schedule and the tasks that are performed on the second work-piece build the backward schedule. It is important to note that in each station at first the tasks in the forward schedule are performed and then the tasks of the backward schedule are executed.
The task deterioration effect is modeled using the formula presented in (1). In this formula P_i is the processing time of task *i*, a_i and b_i are constant part of the processing time and the deterioration growth rate respectively, finally St_i is the starting time of task *i* in the schedule.

$$p_i = a_i + b_i \times St_i \quad (1)$$

2.1Mathematical formulation

The notations used to formulate this problem are presented in table (1).

Table 1 Notations used in the model

i, k	Task
j	Station
I	Set of tasks
J	Set of workstations
$P_i(P_i^*)$	Set of direct (all) predecessors of task i
$F_i(F_i^*)$	Set of direct (all) successors of task i
C	Cycle time
m	Number of stations
M	A big positive number
N	Number of tasks
NTM	Maximum number of tasks that can be assigned to a station
a_i	Constant part of the processing time of task *i*
b_i	Deterioration rate of task *i*
st_i (ft_i)	Start (finish) time of task *i*
$E_i(L_i)$	The earliest (latest) station that task *i* can be assigned
T_j	The set of tasks that can be assigned to station *j*
x_{ijs} $(\acute{x}_{ijs})$	Equals to 1 if task *i* is assigned to workstation *j* in position s of its forward (backward) schedule, otherwise equals to 0.
z_{ik}	Is equal to 1 if task *k* is performed immediately after tasks *i*; otherwise it assumes the value 0.
w_{ij}	Is equal to 1 if task *i* is the last task assigned to the forward schedule in station *j*.
$u_j(\acute{u}_j)$	Is equal to 1 if at least one task is assigned to the forward (backward) schedule in station *j*; otherwise it is equal to 0.

Using the notations presented in Table (1), the model can be formulated as follows:

$$Min\,(C) \quad (2)$$

$$\sum_{j=E_i}^{L_i}\sum_{s=1}^{Nm_j}(x_{ijs}+\acute{x}_{ijs})=1 \quad (\forall i) \quad (3)$$

$$\sum_{\forall i\in T_j} x_{ijs} \leq 1 \quad (\forall j; s=1,2\ldots Nm_j) \quad (4)$$

$$\sum_{\forall i\in T_j} \acute{x}_{ijs} \leq 1 \quad (\forall j; s=1,2\ldots Nm_j) \quad (5)$$

$$\sum_{\forall i\in T_j} x_{ij,s+1} \leq \sum_{\forall i\in T_j} x_{ijs} \quad (\forall j; s=1,2\ldots Nm_j-1) \quad (6)$$

$$\sum_{\forall i\in T_j} \acute{x}_{ij,s+1} \leq \sum_{\forall i\in T_j} \acute{x}_{ijs} \quad (\forall j; s=1,2\ldots Nm_j-1) \quad (7)$$

$$\sum_{j=E_i}^{L_i}\sum_{s=1}^{Nm_j}(N.j+s).x_{ijs}+\sum_{j=E_i}^{L_i}\sum_{s=1}^{Nm_j}(N.(2m+1-j)+s).\acute{x}_{ijs} \leq \sum_{j=E_i}^{L_i}\sum_{s=1}^{Nm_j}(N.j+s).x_{kjs}+\sum_{j=E_i}^{L_i}\sum_{s=1}^{Nm_j}(N.(2m+1-j)+s).\acute{x}_{kjs} \quad \forall(i,k)\in P \quad (8)$$

$$\sum_{j=1}^{m}\left[(x_{ijs})+(x_{kjs+1})\right] \leq 1+z_{ik} \quad (9)$$

$$\sum_{j=1}^{m}\left[(\acute{x}_{ijs})+(\acute{x}_{kjs+1})\right] \leq 1+z_{ik} \quad (10)$$

$$x_{ijs}-\sum_{\forall k\in T_j|(i\neq k)\wedge(k\in PT_i)} x_{kj,s+1} \leq w_{ij} \quad (\forall j; s=1,\ldots,Nm_j-1; \forall i\in T_j) \quad (11)$$

$$u_i=\sum_{\forall i\in T_j} x_{ij1} \quad (\forall j) \quad (12)$$

$$\acute{u}_i=\sum_{\forall i\in T_j} \acute{x}_{ij1} \quad (\forall j) \quad (13)$$

$$w_{ij}+\acute{x}_{kj1} \leq 1+z_{ik}+M.(2-u_j-\acute{u}_j) \quad (\forall i,j,k) \quad (14)$$

$$ft_i \geq st_i+a_i+b_i.st_i \quad (\forall i) \quad (15)$$

$$st_k \geq ft_i-M.(1-z_{ik}) \quad \left(\forall(i,k)\middle|(i\neq k)\wedge(i,k\in T_j)\wedge(k\notin PT_i)\right) \quad (16)$$

$$ft_i \leq C \quad (\forall i) \quad (17)$$

$$st_i \geq 0 \quad (18)$$

In this formulation (2) determines the objective function to be minimized, which is the cycle time. Constraints (3) imply that each task must be assigned to exactly one position in the forward or backward schedule of one station. Equations (4) and (5) ensure that at most one task can be assigned to each position in the forward and backward schedules of each station respectively. Constraints (6) and (7) imply that the positions must be filled in an increasing order in the forward and backward schedules respectively. Constraints (8) indicate the precedence relations, these constraints must be considered both among stations and inside each station. Constraints (9) and (10) ensure that if task *k* is performed immediately after task *i*, z_{ik} is equal to 1. These constraints are applied respectively on the forward and backward schedules in each station. Constraints (11)

imply that if task *i* is the last task in the forward schedule in station *j*; w_{ij} is equal to 1. Equations (12) ensure that if the forward schedule in station *j* is not empty, u_j is equal to 1. Equations (13) ensure that if the backward schedule in station *j* is not empty, v_j is equal to 1. Equations (14) indicate that if task *i* is the last task in the forward schedule in station *j* and task *k* is the first task in the backward schedule of the same station, z_{ik} gets equal to 1. This means that (as assumed before in this section) after performing the tasks in the forward schedule, the tasks in the backward schedule are performed. If any of the forward or the backward schedules is empty the equations (14) become redundant. Constraints (15) indicate that finish time of each task *i* must be greater than or equal to the starting time of it plus the processing time. The processing time in these constraints are calculated using equation (1). Constraint (16) imply that if task *k* is performed immediately after task *i* then starting time of task *k* must be greater or equal to the finish time of task *i.* Equations (17) indicate that every task *i* must be finished before the cycle time limit. Constraints (18) ensure that the starting time of all tasks are greater than or equal to 0.

2.2Numerical example

In this section a numerical example is presented to illustrate the benefits of considering U shaped lines and task deterioration effect simultaneously. The precedence graph for this example is presented in Figure 2, in this figure each circle represents a task.

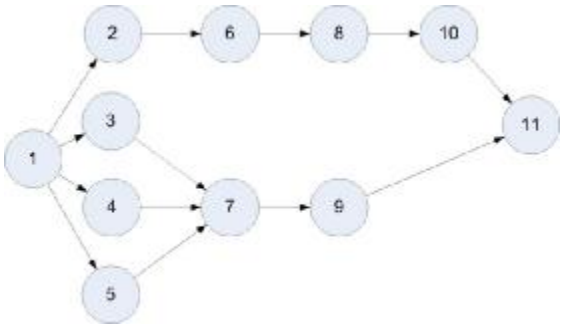


Figure 2 the precedence graph for the instance

Processing time information for each task is presented in Table 2. The number of stations is assumed to be equal to 5.

Table 2 processing time information for the instance

task	a_i	b_i
1	6	1.2
2	2	0.6
3	5	1
4	7	1.4
5	1	0.6
6	2	0.8
7	3	0.9
8	8	1
9	9	1.8
10	15	2
11	4	2

This example is solved in both serial and U shaped lines and the resulting solutions are presented in tables 3 and 4. As seen in these tables the obtained cycle time for the straight line is 30. This number for the U shaped line is equal to 21.4 which is improved with respect to the straight line.

Table 3 optimal scheduling and assignment of tasks for a straight line

Station 1		Station 2		Station 3		Station 4		Station 5	
schedule	Station time	schedule	Station time	schedule	Station time	schedule	Station time	schedule	Station time
{1,6,2}	22.88	{5,4,8}	26.8	{3,10}	30	{7,9}	17.4	{11)	4

Table 4 optimal scheduling and assignment of tasks for a U shaped line

	Station 1		Station 2		Station 3		Station 4		Station 5	
	schedule	time	schedule	time	schedule	time	schedule	time	schedule	time
Forward schedule	{1,4}	21.4	{5,2}	14.8	{6}	21	-	17.4	-	18
backward schedule	-		{11}		{10}		{7,9}		{3,8}	

3 The proposed GA

GA is a strong meta-heuristic algorithm which has been used to solve many combinatorial optimization problems. In a GA every solution, also called individual or chromosome, is represented by a structure which shows the properties of the solution. At first a population of these solutions are generated and then each chromosome in the population is evaluated and a fitness value is assigned to it, this value represents the quality of the individual, i.e. the higher the fitness value the better the solution. Then through a selection procedure, a set of chromosomes are selected as parents to perform the crossover and mutation operations to generate a new population. This process is iterated until the termination criteria are met.
In order to solve the problem under consideration, two decisions should be made: the sequencing of tasks and assigning tasks to the stations. In this study a GA is applied to make each of these decisions, in other words the sequence of tasks is determined using a GA then for each sequence, another GA is used to determine assignment of tasks to the stations.
On the other hand since the correct selection of the operators and parameters of the GA has a major effect on its performance, these operator and parameters are determined using the design of experiment (DOE) method, DOE is a structured method that shows the relationship between factors that affect output of a process. The use of DOE approach in GA was first proposed by (Bagchi & Deb, 1996), this approach has been used in the field of flow-shop and single machine scheduling problems (Nagano et al., 2008; Iyer & Saxena, 2004; Vallada & Ruiz, 2010; Ruiz et al., 2005; Bahalke et al., 2010). However it is less considered in assembly line balancing problems. The elements of the proposed GA are presented on the following sections.

3.1Representation method and initial population

In this study a simple permutation is used to represent the sequence of tasks which determines the sequence of performing tasks on the work-piece. Figure 3 shows an instance of chromosome representation, this instance has seven tasks and it is obvious that this sequence must observe the precedence constraints. In this study the initial population is generated randomly.

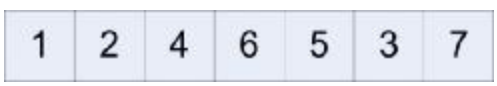


Figure 3 an instance of chromosome representation

3.2Fitness function

In order to evaluate an individual at first the assignment of tasks to stations must be determined, in order to determine this assignment another GA is used. The elements of this GA are explained in the following sections.

3.2.1 Representation method and initial population

In order to represent the assignment of tasks to the station a string with length of 2*m is used, where m is the number of stations. In this representation the first m numbers determine the number of tasks assigned to stations in a forward manner and the last m numbers determines the number if tasks assigned to the stations in a backward manner. More specifically for $i \leq m$ the i th number in the string determines the number of tasks assigned to the i th station and for $i > m$ this number determines the number of tasks assigned to the (2*m+1-i)th station.

Figure 4 shows an example of this representation method. In this example there are seven tasks and three stations. If this this figure is considered with Figure 3, since the first number in this example is 1, the first task, task 1, is assigned to the first station in a forward manner. And since the second number in this string is 2, two tasks, tasks 2 and 4, are assigned to the second station in a forward manner. Following this order task 6 is assigned to the third station. After considering the first three numbers in the string, the remaining numbers determine backward assignment of tasks to the stations. Since the fourth number in the string is equal to zero, no backward task is assigned the third (2*3+1-4=3) station. Accordingly task 5 is assigned to the second station in a backward manner and tasks 3 and 7 are assigned to the first station in a backward manner.

1	2	1	0	1	2

Figure 4 an instance of representation for assignment of tasks to stations

3.2.2 Solution evaluation, selection method and elitism

After determining the sequence of tasks and assignment of tasks to the stations, the station times and cycle time can be calculated. Since the objective is minimizing the cycle time and GA maximizes the fitness, fitness function is set to be equal to the reciprocal of cycle time.

In this study the well-known roulette-wheel method (Sivanandam & Deepa, 2008) is used to select the parents. On the other hand applying crossover and mutation operators may result in producing chromosomes that are not better than the individuals in the previous population, in other words the best individuals in the previous population may be lost. In order to avoid this, a percentage of the best solutions in the previous population are transferred directly to the next population. This procedure is called elitism and it ensures that the best solution in the new population will not be worth than that of the previous population. In this study 10% of best solutions in the population are transferred to the next population.

3.2.3 Crossover

In order to perform the crossover procedure, two parents are selected using the selection method mentioned in section 0, then for each station, the number of assigned tasks for the child is inherited from one of the parents which is selected randomly. Figure 5 shows an example of this crossover operator. In this example there are seven tasks and two stations. In this example P1 and P2 are the parents and Random string determines the parent selected to inherit the number of tasks. For example in order to determine the first number in the child, the first number in the random string is considered, since this number is equal to 1, therefore the child inherits this feature from the first parent, and since the first place in this parent's string is equal to one, the first place in the child's string gets equal to 1. Following this order the child presented in the figure is obtained. However, as seen in this figure the resulting child is not feasible the sum of numbers in the child is 6 which is less than 7, the number of tasks. In order to avoid this issue, after performing the crossover operator, if sum of the numbers is

less than the total number of tasks, one of the stations is selected randomly and the number of tasks for this station is incremented by one unit. This procedure is repeated until the sum of the numbers is equal to the total number of tasks. On the other hand if the sum of the numbers is more than the total number of tasks, one of the stations that have one or more than one tasks is selected randomly and the number of tasks assigned to this station is decremented by one. This procedure is repeated until the sum of the number of tasks in each station is equal to the total number of tasks.

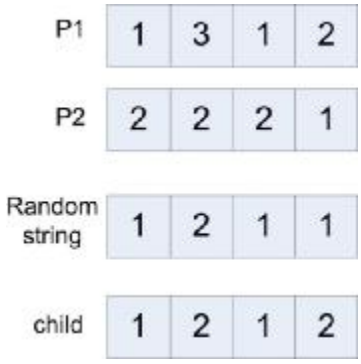


Figure 5 an example of the crossover operator

3.2.4 Mutation

The mutation procedure is performed on each chromosome with a certain probability. This procedure is performed as follows: two numbers are selected from the string of the child and their places are exchanged. Figure 6 illustrates this mutation with an example; this example has seven tasks and 2 stations. The first string is the solution before undergoing the mutation and the second string shows the solution after performing it. In this example two tasks are selected and swapped with each other, these two tasks are highlighted in the figure.

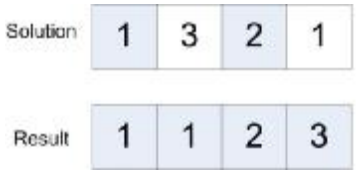


Figure 6 an example of mutation procedure

3.2.5 Termination criteria

In this study performing a certain number of iterations is assumed to be the termination criterion. In the next section the number of iteration is determined using the DOE method.
After solving the problem of assigning tasks to the stations, considering the sequence of tasks, the best cycle time obtained by the GA is considered as the cycle time for the sequence of tasks. Since the objective is to minimize the cycle time, and GA tends to maximize the fitness function, the fitness function is assumed to be equal to the reciprocal of cycle time.

3.3Selection method and elitism

In this study the well-known roulette wheel method is used to select parents in order to perform crossover and mutation operators. Furthermore in order to preserve the best solutions found so far, 10% of the best solutions in the population are transferred directly to the next population.

3.4Crossover

The crossover operator takes two parents and combines their features two produce a new child. Therefore the generated child inherits it's characteristics from the parents. The crossover operator is performed with the hope of producing better children by combining the key features of the parents. In the proposed GA, two crossover operators are used. The first crossover operator considered in this paper is called ordered crossover or OX. In the OX crossover given two parents, parent 1 and parent 2, one random point is selected which divides the parents into right and left portions; then child 1 inherits its right portion from parent 1, and the left portion of it is determined by the tasks in the left section of parent 1 but in the order in which they appear in parent 2. A similar procedure is used to determine child 2.
The second crossover to be applied is called two-point ordered crossover or 2OX. 2OX involves selecting two random points dividing the parents into left, middle and right portions. The left and right parts of child 1 is directly inherited from parent 1, and its middle section is determined by the middle section of parent 1, in the order in which the tasks appear in parent 2. A similar procedure is used to determine child 2. Both OX and 2OX can be found in (Sivanandam & Deepa, 2008).

3.5Mutation

After performing the crossover operator on the chromosomes, the mutation operator is performed. Mutation helps the algorithm to avoid from getting trapped in a local optimal point. This operator is of major importance in searching the whole solution space. The mutation operator is considered as a basic operator for maintaining the population diversity. In this paper two mutation operators are considered:

- Swap mutation: a task is selected randomly and if it doesn't have any precedence relation with its adjacent task, these two tasks are swapped.
- Scramble mutation: two random points are selected and the tasks between them are rearranged, considering the precedence relations.

3.6Restart scheme

In GA after some iterations the population diversity may get low enough to trap in a local optimum solution. To overcome this issue a method called restart scheme is used. This method has been used in (Vallada & Ruiz, 2010; Ruiz et al., 2005; Bahalke et al., 2010) to solve scheduling problems. In this method if the best solution in the population is not improved for G_r consecutive iterations, the population is regenerated. More specifically the restart scheme used in the proposed GA works as follows:

1) Set $count = 0$.
2) In each iteration i store the minimum cycle time: $cycle_i$
3) If $cycle_i = cycle_{i-1}$ then $count = count + 1$. else set count=0.
4) If $count > G_r$ do the followings:

- Sort the population in ascending order of cycle time
- Skip the first 20% of the population
- From the remaining 80% half of them are produced by doing swap mutation on the first 20% best individuals.
- The remaining 30% of the population is produced randomly

3.7Experimental calibration of GA

In this section the operators and parameters of the GA is selected using the Taguchi method. These operators and parameters are presented in

Table 5 the factors and their levels

Factor	levels	Number of levels
Crossover type (Cross_type)	{OX,2OX}	2
Mutation type (Mut_type)	{swap,scramble}	2
Population size for the first GA (pop_size1)	{20,40,60}	3
Population size for the second GA (pop_size1)	{10,20,30}	3
The number of unimproved iteration for restart scheme (GR)	{10,20,30}	3
The probability of mutation (mut_rate)	{0.05,0.1,0.15}	3
The number of iterations for the second GA (iter2)	{40,60,80}	3

Since the values of these operators and parameters can greatly affect the performance of the GA, it is interesting to find the proper combination of these operator and parameters in order to optimize the performance of the GA.

There are several ways of designing the experimental tests. The best but most exhaustive and approach is full-factorial. In most cases, this approach is inefficient due to the large number of factors and their respective levels. To overcome this issue, fractional factorial experiments (FFEs) are developed which limit the number of required tests to a fraction of the total possible combinations. (Taguchi, 1986), developed a family of FFE matrices that reduce the number of experiments, but still provides sufficient information. In the Taguchi method, orthogonal arrays are used to study a large number of decision variables with a small number of experiments. The Taguchi method is more efficient against the full factorial procedure especially in large-scale cases.

Taguchi separates the factors into two groups: controllable and noise factors. Noise factors are those which we have no direct control over them. Since it is often impossible to get rid of the noise factors, the Taguchi method seeks to minimize the effect of noise factors and to determine optimal level of important controllable factors based on the concept of robustness.

Taguchi classifies the objective functions into three categories: the smaller is better, the larger is better and the nominal is best. In this paper since the objective function is minimizing the cycle time, the smaller is better is selected.

The instance to perform the experiment is generated as follows: at first the precedence graph with the desired order strength is generated using the algorithm proposed by (Gehrlein, 1986). Then the fixed part of tasks time is generated between 5 and 30, and then the deterioration rate for each task is randomly selected from the values 0.1, 0.2, 0.3 and 0.4. Using this method an example with 30 tasks and order strength of 0.75 is generated and this example is solved for the number of stations 15, 20 and 25 and the mean value of cycle time is considered as the response variable.

The degrees of freedom for the factors considered for the GA is equal to 12, and there are 7 factors to be considered. Therefore the Taguchi design must have at least 12 rows and 7 columns. Among the standard Taguchi designs L27 (3^7) is selected as an appropriate design. But this design needs to be modified because there are two factors that have two levels and in order to match the standard L27 design these factors must have one more level. In order to overcome this issue one of the existing levels for these factors is randomly selected and repeated. For example for the crossover type the second level is repeated. Table 6 show the modified design for L27.

Table 6 the modified design for L27

test	cross_type	mut_type	pop_size1	pop_size2	GR	mut_rate	iter2
1	1	1	1	1	1	1	1
2	1	1	1	1	2	2	2
3	1	1	1	1	3	3	3
4	1	2	2	2	1	1	1
5	1	2	2	2	2	2	2
6	1	2	2	2	3	3	3
7	1	1	3	3	1	1	1
8	1	1	3	3	2	2	2
9	1	1	3	3	3	3	3

10	2	1	2	3	1	2	3
11	2	1	2	3	2	3	1
12	2	1	2	3	3	1	2
13	2	2	3	1	1	2	3
14	2	2	3	1	2	3	1
15	2	2	3	1	3	1	2
16	2	1	1	2	1	2	3
17	2	1	1	2	2	3	1
18	2	1	1	2	3	1	2
19	2	1	3	2	1	3	2
20	2	1	3	2	2	1	3
21	2	1	3	2	3	2	1
22	2	2	1	3	1	3	2
23	2	2	1	3	2	1	3
24	2	2	1	3	3	2	1
25	2	1	2	1	1	3	2
26	2	1	2	1	2	1	3
27	2	1	2	1	3	2	1

After obtaining the results of this design the mean S/N ratio is calculated for each factor using Minitab 16 software. The results are presented in Figure 7.

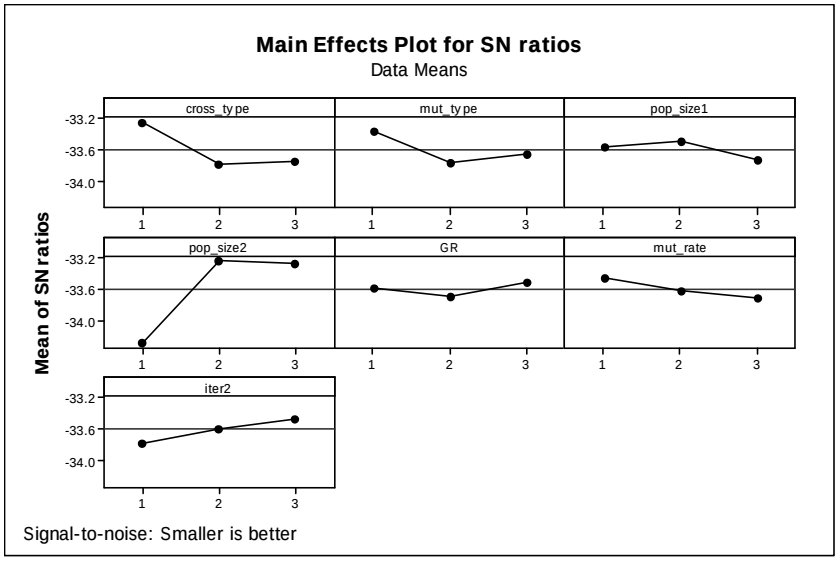


Figure 7 the S/N ration plot for each factor

Using this figure, the optimum levels for each factor is determined, the results are presented in

Table 7 the optimum level for each factor

Factor	Optima level
Crossover type (Cross_type)	OX
Mutation type (Mut_type)	swap
Population size for the first GA (pop_size1)	40
Population size for the second GA (pop_size1)	20
The number of unimproved iteration for restart scheme (GR)	30
The probability of mutation (mut_rate)	0.05
The number of iterations for the second GA (iter2)	80

In order to determine the relative effect of different factors on the response variable, an analysis of variance is performed and the results are presented in

Table 8. As seen in this table the population size for the second GA with the F-ratio of 15.01 has the most effect on the objective function, the second important factor is the crossover type with f-ratio of 3.82.

Table 8 results of ANOVA for the objective function

Source	DF	Seq SS	Adj SS	Adj MS	F	P
cross_type	2	50.479	50.479	25.239	3.82	0.052
mut_type	2	25.483	25.483	12.741	1.93	0.188
pop_size1	2	8.811	8.811	4.406	0.67	0.531
pop_size2	2	198.228	198.228	99.114	15.01	0.001
GR	2	4.224	4.224	2.112	0.32	0.732
mut_rate	2	7.548	7.548	3.774	0.57	0.579
iter2	2	12.794	12.794	6.397	0.97	0.407

4 The proposed PSO

The particle swarm optimization algorithm is introduced by (Kennedy & Eberhart, 1995). This algorithm imitates the social behavior of a flock of birds that are flying to an unknown destination. In PSO every solution is a bird in the flock and is called a particle. A particle in PSO is equivalent to chromosome in GA. In contrast with GA the evolutionary process does not produce new children from the parents. Instead the evolution of social behavior of the birds in the population is considered.
The evolutionary process begins with a random initial population of particles (solutions). Each of these particles is represented by a position in and S dimensional space, where S is the number of variables. Throughout the process, each particle i, preserves three values: current position x_i, best position reached in previous iterations P_i and its flying velocity V_i. In each cycle, the position $P_g$is calculated as the best reached by all of the particles. Accordingly the velocity and position of each particle is updated using the following equations:

$$\begin{cases} V_{i+1} = \omega V_i + C_1 Rand1()(P_i - X_i) + C_2 Rand2()(P_g - X_i) & (19) \\ X_{i+1} = X_i + V_{i+1}, \quad -V_{max} <= V_{i+1} <= V_{max} & (20) \end{cases}$$

In these equations C_1 and C_2 are two constant positive numbers called acceleration coefficients. $Rand1()$ and $Rand2()$ are two random function in the range [0,1]. V_{max} is an upper limit on the maximum change in the velocity, and ω is an inertia weight employed as an improvement proposed by (Shi & Eberhart, 1998) to improve the algorithm. This weight controls the effect of previous velocities on the current velocity and creates a balance between global and local search. The pseudo code of the proposed PSO is as follows:

Algorithm *PSO*
Begin
Generate random population of N solutions (particles);
For each individual $i \in N$ calculate fitness (i);
Initialize the value of the weight factor ω;
For each particle;
Set *pBest* as the best position of particle i;
If fitness (i) is better than *pBest*;
pBest(i)=fitness (i);
End;
Set *gBest* as the best fitness of all particles;
For each particle;
Calculate particle velocity according to Eq. (6a);
Update particle position according to Eq. (6b);
End;
Update the value of the weight factor ω (option);
Check if termination=true;
End

The PSO algorithm is originally proposed to solve continuous and cannot be directly used to solve discrete optimization problems, in order to use this algorithm to solve the problem under consideration a proper representation method must be used.

4.1Solution representation

In order to solve the problem under consideration two types of decisions should be considered: the sequence of tasks and assignment of tasks to stations. In order to represent the sequence of tasks a string of N float numbers is used. In order to determine the sequence of performing tasks, at first the tasks are sorted in an increasing order of this float value. This resulting sequence may not observe the precedence constraints. Therefore using a topological sort method (Baybars, 1986) a feasible sequence is obtained. In order to determine the assignment of tasks to the stations the GA proposed in section 3.2 is used.

4.2Solution evaluation and termination criteria

In order to evaluate each solution, which is a string of float numbers, the sequence of tasks and assignment of them to the stations is determined using the methods explained in the previous section. Then the station times and the cycle time is calculated. The algorithm terminates when a certain computation time has passed.

4.3Experimental calibration of PSO

In this section the parameters of the PSO is determined using the Taguchi method, these parameters are presented in

Table 9 as a number of factors. The instance to perform the experiment is generated using the method explained in section 3.7 and the generated instance is solved for the number of stations equal to 15, 20 and 25 and the mean cycle time is considered as the fitness function.

Table 9 the parameters and their levels

Factor	levels	Number of levels
Population size for PSO (Pop_size1)	{20,40,60}	3
Inertia weight (w)	{0.7,0.8}	2
The first acceleration coefficient (C1)	{1,2}	2
The second acceleration coefficient (C2)	{1,2}	2
Population size for the GA (Pop_size2)	{20,40,60}	3
The probability of mutation (mut_rate)	{0.05,0.1,0.15}	3
The number of iterations for the second GA (iter)	{40,60,80}	3

There are 7 factors considered and the degree of freedom for these factors is equal to 11, therefore the selected Taguchi design must have at least 11 rows and 7 columns. Among the available standard designs L27 is selected as an appropriate design. But this design should be modified to match the considered factors, because there are three factors that have only two levels, in order to fully match the standard design, these three factors must have three levels. In order to overcome this issue for each factor that have less number of levels than required, one of the existing levels is selected and this level is repeated for the factor.

Table 10 presents the modified L27 design for the factors considered here.

Table 10 the modified L27 design

test	pop_size1	w	C1	C2	pop_size2	mut_rate	iter

1	1	1	1	1	1	1	1
2	1	1	1	1	2	2	2
3	1	1	1	1	3	3	3
4	1	2	2	2	1	1	1
5	1	2	2	2	2	2	2
6	1	2	2	2	3	3	3
7	1	1	2	1	1	1	1
8	1	1	2	1	2	2	2
9	1	1	2	1	3	3	3
10	2	1	2	1	1	2	3
11	2	1	2	1	2	3	1
12	2	1	2	1	3	1	2
13	2	2	2	1	1	2	3
14	2	2	2	1	2	3	1
15	2	2	2	1	3	1	2
16	2	1	1	2	1	2	3
17	2	1	1	2	2	3	1
18	2	1	1	2	3	1	2
19	3	1	2	2	1	3	2
20	3	1	2	2	2	1	3
21	3	1	2	2	3	2	1
22	3	2	1	1	1	3	2
23	3	2	1	1	2	1	3
24	3	2	1	1	3	2	1
25	3	1	2	1	1	3	2
26	3	1	2	1	2	1	3
27	3	1	2	1	3	2	1

After performing this experiment the results are analyzed using Minitab 16 software and the S/N ratio for each factor is presented in Figure 8.

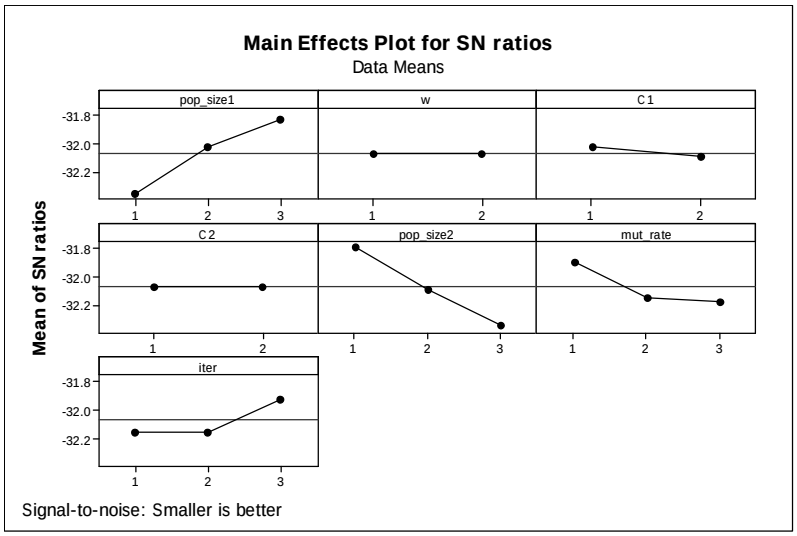


Figure 8 S/N plot for the factors

Using this figure the optimum levels for each factor is presented in

Table 11.

Table 11 optimal level for each factor

factor	Optimal level
Population size for PSO (Pop_size1)	{60}
Inertia weight (w)	{0.8}
The first acceleration coefficient (C1)	{1}
The second acceleration coefficient (C2)	{1}
Population size for the GA (Pop_size2)	{20}
The probability of mutation (mut_rate)	{0.05}
The number of iterations for the second GA (iter)	{80}

In order to determine the relative effect of different factors on the response variable, an analysis of variance is performed and the results are presented in

Table 8. As seen in this table the population size for the second GA with the F-ratio of 62.47 has the most effect on the objective function, the second important factor is the population size for the PSO with f-ratio of 55.93.

Table 12 results of ANOVA for the objective function

Source	DF	Seq SS	Adj SS	Adj MS	F	P
pop_size1	2	1.26	1.26	0.63	55.93	0.00
w	1	0.00	0.00	0.00	0.01	0.93
C1	1	0.02	0.02	0.02	2.10	0.17
C2	1	0.00	0.00	0.00	0.00	0.96
pop_size2	2	1.40	1.40	0.70	62.47	0.00
mut_rate	2	0.45	0.45	0.22	19.83	0.00
iter	2	0.33	0.33	0.17	14.73	0.00

5 Computational results

In this section the computational results are presented at first several small instances of the problem are solved using the mathematical model and proposed algorithms. The obtained solutions are compared with each other in terms of solution quality and computation time. Both meta-heuristic algorithms are calibrated using the DOE method explained in sections 3.7 and 4.3 and the termination criteria for both algorithms is set to be a maximum running time of 30 seconds or reaching an optimum solution. These algorithms are coded in C++ language and ran on a personal computer with Intel Pentium dual processor 2.2 GHz and 2 GB of RAM. Also in order to solve the mathematical model, CPLEX software is used,

5.1Computational results for small problems

In this section three small examples are generated and solved using the mathematical formulation and the proposed algorithms. In order to generate these instances three precedence graphs are chosen from the available problems on the assembly line balancing research homepage (www.assembly-line-balancing.de). Then the fixed part of the processing time for each task is randomly generated between 5 and 30. Afterwards the deterioration rate for each task is randomly selected from the values 0.1, 0.2 and 0.3. For each of the generated instances the meta-heuristic algorithms are run five times and the average and standard deviation of the results is reported in

Table 13. This table also presents the results of the mathematical model, for the first two examples the model can be solved to optimality in a reasonable amount of time. But for the third example, the obtained lower and

upper bounds after 30 minutes of running the CPLEX software is reported. In this table GA and PSO are the proposed genetic algorithm and particle swarm optimization respectively. As seen in this table for the first two instances both of the proposed algorithms obtain the optimum solution. This shows the efficiency of the proposed algorithms.

Table 13obtained cycle time for the mathematical model and proposed algorithms

instance	Number of stations	Solution method				
		Mathematical model	GA		PSO	
			mean	SD	mean	SD
MERTENS	3	36.28	36.28	0	36.28	0
JAESCHKE	3	60.57	60.57	0	60.57	0
	4	44.1	44.1	0	44.1	0
JACKSON	4	[30-64.38]	58.92	0	58.92	0
	5	[30-50.8]	43.6	0	43.6	0
	6	[30-53.4]	38.8	0	38.8	0

in order to examine the performance of these algorithms in more detail the running time of these algorithms and the mathematical model for each instance is presented in

Table 14, as seen in this table both meta-heuristic algorithms have obtained the optimum solution in a less amount of time.

Table 14 running time of the proposed algorithms and mathematical model (in seconds)

instance	Number of stations	Solution method				
		Mathematical model	GA		PSO	
			mean	SD	mean	SD
MERTENS	3	5.49	0.79	0.072	0.75	0.048
JAESCHKE	3	65.54	0.8	0.071	1.47	1.82
	4	145.33	0.88	0.074	0.851	0.012
JACKSON	4	1800	30.5	0.467	30.29	0.158
	5	1800	30.45	0.195	30.38	0.283
	6	1800	30.57	0.498	30.5	0.251

5.2Computational results for large problems

In this section several large-sized problems are generated and solved using the proposed meta-heuristic algorithms. The instance are generated using the same method as the one used in the previous section and for each instance the meta-heuristic algorithms are run five times and the average and standard deviation of the results is reported in

Table 15. As seen in this table for the first three instances the PSO obtains acceptable solutions with respect to the GA and in some case even reaches better solutions. An interesting observation is that for the first three instances with increasing the number of stations the relative performance of the PSO with respect to the GA improves. However for the next two instances the GA has a better overall performance. This means that with increasing the number of tasks i.e. the size of the problem, the GA performs better than PSO. Therefore for instances with number of tasks less than 90, if the number of stations is low GA is recommended and for the higher number of stations it is better to use PSO; and for the instances with more than 90 tasks the GA is recommended.

Table 15 the results of the proposed algorithms for large sized instances

instance	Number of tasks	Number of stations	Solution method	
			GA	PSO

			mean	SD	mean	SD
TONGE	70	10	218.8	0.69	218.43	2.46
		20	90.65	2.6	92.84	1.03
		30	66.09	1.47	64.71	1.46
		40	53.38	0.53	53.67	1
		50	47.84	0.78	47.47	1.59
WEE-MAG	75	10	259.17	1.62	260.06	1.79
		20	107.47	0.5	106.3	1.07
		30	77.06	0.92	76.5	1.25
		40	60.96	1.91	60.33	1.45
		50	52.38	0.57	52.24	0.96
LUTZ2	89	10	355.53	1.98	355.8	3.1
		20	129.26	2.31	130.94	2.17
		30	88.7	0.94	87.42	1.04
		40	70.82	0.9	70.62	2.15
		50	60.9	1.36	60.5	1.69
		60	53.36	0.59	53.68	0.84
MUKHERJE	94	10	410.58	1.96	420.75	4.51
		20	146.53	3	150.99	2.89
		30	98.21	2.49	101.56	1.32
		40	77.98	2.05	82.39	1.2
		50	67.33	1.46	70.1	3.08
		60	57.94	0.92	60.13	2.48
BARTHOLD	148	20	296.23	6.41	303.87	5.58
		40	128.12	2.87	134.14	2.15
		60	92.72	1.58	97.02	2.11
		80	76.28	2.47	79.19	0.53
		100	66.79	1.45	68.64	1.61

6 Conclusions and future research

In this paper the U shaped assembly line balancing problem with the task deterioration effect is considered. More specifically second type of this problem which is minimizing the cycle time for a fixed number of stations is taken into account. Task deterioration effect means that if the task is performed later its processing time increases. In order to optimally solve this problem, an integer programming formulation is proposed. Furthermore two meta-heuristic algorithms, namely a GA and a PSO are proposed to solve the proposed model in reasonable amount of time. The computational results illustrate the performance of the proposed model and meta-heuristic algorithms.

However further research is required to consider the task deterioration effect in other assembly line balancing environments such as two-sided lines or multi-manned assembly lines. Also considering the first type of the proposed model is suggested for future research in this area.

References

Agpak, K. & Gokcen, H., 2005. Assembly line balancing: Two resource constrained cases. International journal of production economics, 96, pp.129-40.

Andres, C., Miralles, C. & Pastor, R., 2008. Balancing and sequencing tasks in assembly lines with sequence–dependent setup times. European Journal of Operational Research.

Bagchi, T. & Deb, K., 1996. Calibration of GA parameters: the design of experiments approach. comput Sci Inform, 26(3), p.46–56.

Bahalke, U., Yolmeh, A.M. & Shahanaghi, K., 2010. Meta-heuristics to solve single-machine scheduling problem with sequence-dependent setup time and deteriorating jobs. Int J Adv Manuf Technol, 50, p.749–759.

Bartholdi, J., 1993. Balancing two–sided assembly lines: A case study. 31, p.2447−2461.
Baybars, I., 1986. A survey of exact algorithms for the simple assembly line balancing problem. Management Science, 32, p.909−932.
Browne, S. & Yechiali, U., 1990. Scheduling deteriorating jobs on a single processor. Ops Res, 38, p.495–498.
Buxey, G., 1974. Assembly line balancing with multiple Stations. Management Science, 20, p.1010−1021.
Corominas, A., Ferrer, L. & Pastor, R., 2010. Assembly line balancing: general resource-constrained case, , (), . International Journal of Production Research, 49 (12), p.3527 – 3542.
Erel, E. & Sarin, S., 1998. A survey of the assembly line balancing procedures. Production Planning & Control, 9, p.414−434.
Gehrlein, W., 1986. On methods for generating random partial orders. Operation s Research Letters, 5 (6), p.285–291.
Grangeon, N., Leclaire, P. & Norre, S., 2011. Heuristics for the re-balancing of a vehicle assembly line. International Journal of Production Research, p.to appear.
Gupta, J.N.D. & Gupta, S.K., 1988. Single facility scheduling with nonlinear processing times. Comput. Ind. Engng, 14, p.387–393.
Iyer, S. & Saxena, B., 2004. Improved genetic algorithm for the permutation flowshop scheduling problem. Comp Oper Res, 31, p. 593–606.
Kennedy, J. & Eberhart, R., 1995. Particle swarm optimization. In Proceedings of the IEEE international conference on neural networks. Perth,Australia, 1995.
Martino, L. & Pastor, R., 2009. Heuristic procedures for solving the general assembly line balancing problem with setups. International Journal of Production Research.
Miltenburg, J., 1998. Balancing U–lines in a multiple U−line facility. European Journal of Operational Research, 109, p.1−23.
Miltenburg, J. & Wijngaard, J., 1994. The U–line line balancing problem. Management Science, 40, p.1378−1388.
Nagano, M., Ruiz, R. & Lorena, L., 2008. A constructive genetic algorithm for permutation flowshop scheduling. Comput Ind Eng, 55, p.195–207.
Pastor, R., 2011. LB-ALBP: the lexicographic bottleneck assembly line balancing problem.. International Journal of Production Research, 49 (8), p.2425 — 2442.
Pinto, P., Dannenbring, D. & Khumawala, B., 1983. Assembly line balancing with processing alternatives: an application. Management Science, 29, p.817−830.
Rekiek, B., Dolgui, A., Delchambre, A. & Bratcu, A., 2002. State of art of optimization methods for assembly line design. Annual Reviews in Control, 26, p.163−174.
Ruiz, R., Maroto, C. & Alcaraz, J., 2005. Solving the flowshop scheduling problem with sequence-dependent setup times using advanced metaheuristics. Eur J Oper Res, 165, p.34–54.
Salveson, M.E., 1955. The assembly line balancing problem. The Journal of Industrial Engineering, 6 (3), p.18–25.
Scholl, A., 1999. Balancing and sequencing assembly lines. 2nd ed. Physica–Verlag, Heidelberg.
Scholl, A. & Becker, C., 2006. State–of–the–art exact and heuristic solution procedures for simple assembly line balancing. European Journal of Operational Research, 168, p.666−693.
Seyed-Alagheband, S.A., Fatemi Ghomi, S.M.T. & Zandieh, M., 2011. A simulated annealing algorithm for balancing the assembly line type 2 problem with sequence-dependent setup times between tasks. International Journal of Production Research, 49(3).
Shahanaghi, K., Yolmeh, A.M. & Bahalke, U., 2009. Scheduling and balancing assembly lines with the task deterioration effect. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 224(7), pp.1145-53.
Shi, Y. & Eberhart, R., 1998. A modified particle swarm optimizer. In Proceedings of the IEEE international conference on evolutionary computation., 1998.
Sivanandam, S. & Deepa, S., 2008. Introduction to genetic algorithms. Springer, New York.
Taguchi, G., 1986. Introduction to quality engineering. White Plains: Asian Productivity Organization/UNIPUB.
Toksarı, M.D. et al., 2010. Assembly line balancing problem with deterioration tasks and learning effect. Expert Sys. with App, 37(2), p.1223–1228.
Vallada, E. & Ruiz, R., 2010. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. Omega, 38, p.57–67.