Journal of Optimization in Industrial Engineering
Vol.13, Issue 2, Summer & Autumn 2020, 123-140
DOI: 10.22094/JOIE.2020.579974.1605

Research Article

A Hybrid Unconscious Search Algorithm for Mixed-model Assembly Line Balancing Problem with SDST, Parallel Workstation and Learning Effect

Moein Asadi-Zonouz [a], Majid Khalili*[,b], Hamed Tayebi [b]

[a] *Department of Industrial and Systems Engineering, Tarbiat Modares University, Tehran, Iran*
[b] *Department of Industrial Engineering, Islamic Azad University Karaj Branch, Karaj, Iran*
Received 27 November 2018; Revised 26 April 2020; Accepted 01 June 2020

Abstract

Due to the variety of products, simultaneous production of different models has an important role in production systems. Moreover, considering the realistic constraints in designing production lines attracted a lot of attentions in recent researches. Since the assembly line balancing problem is NP-hard, efficient methods are needed to solve this kind of problems. In this study, a new hybrid method based on unconscious search algorithm (USGA) is proposed to solve mixed-model assembly line balancing problem considering some realistic conditions such as parallel workstation, zoning constraints, sequence dependent setup times and learning effect. This method is a modified version of the unconscious search algorithm which applies the operators of genetic algorithm as the local search step. Performance of the proposed algorithm is tested on a set of test problems and compared with GA and ACOGA. The experimental results indicate that USGA outperforms GA and ACOGA.

***Keywords*:** Unconscious Search algorithm; Assembly line balancing problem; Learning Effect; Parallel workstation; Sequence-dependent setup times.

1. Introduction

The most important purpose of each organization can be defined as maximizing the benefits and revenues. Since time and resources have a strong effect on achieving this purpose, optimal use of time and resources is crucial. Moreover, due to the variety of products, simultaneous production of different models is significant and using of a mixed-model production system is essential.

Mixed-model assembly lines are commonly used for their flexibility with respect to model changes, for reducing the final product inventories and for a continuous flow of materials (Yagmahan, 2011). In order to avoid the high cost of building and maintaining an assembly line for each model, mixed-model assembly line handled for the first time by Thomopoulos (1967), which has shown that single-model line balancing techniques are adaptable to mixed-model schedules. In a follow up work, Thomopoulos (1970) shown how mixed-model line balancing problems can be modified to yield smoother model assignments in continuous assembly stations.

Mainly two types of balancing problems come up for mixed-model assembly lines: (1) design of a new assembly line for which the demand can be easily forecasted and (2) redesign of an existing assembly line when changes in the assembly process or in the product range occurs. The first type is discussed in this paper which is called as MMALBP-I and can be stated as follow: Given M models, the set of tasks associated with each model, the performance times of the tasks, and the set of precedence relations which specify the permissible orderings of the tasks for each model, the problem is to assign the tasks to an ordered sequence of stations such that the precedence relations of each model are satisfied (Gokcen & Erel, 1997).

Before the 70^{s} of the 20^{th} century, most of the techniques applied to solve the ALBP required assigning each task to a single workstation and, consequently, the production rate was limited by the longest task time. This assumption could be relaxed by utilizing parallel workstations in such a way that two or more replicas of a workstation can perform the same set of tasks on different assemblies. The introduction of parallel workstations not only allowed for cycle times shorter than the longest task time and thus an increase in the production rate, but also provided greater flexibility in designing the assembly line (Buxey, 1974). When parallel workstations are introduced, the number of tasks performed by each worker increases, but this contradicts one of the main advantages of using an assembly line which is using of low-skilled labor that can be easily trained. Therefore, in order to maintain that advantage, it is necessary to control the process to create parallel workstations in such a way that workstations are replicated only when required (Vilarinho & Simaria, 2002).

In many real production lines, however, the sequence in which tasks are developed inside the workstation matters,

*Corresponding author Email address: khalili.mj@kiau.ac.ir

since sequence-dependent setup times between tasks are present. Andres showed that assembly line not only requires balancing, but also the scheduling of tasks assigned to every workstation must be defined due to the existence of sequence-dependent setup times (Andres et al., 2008). For example, sequence-dependent setup times must be considered when they are very low compared to operation times, or in robotic lines, or when components placed in distanced containers. So, if these kinds of setup times are considered, better solutions will be achieved for line balancing problems.

In many realistic settings, the production facility (a machine, a worker) improves continuously with time. As a result, the production time of a given product is shorter if it is scheduled later, rather than earlier in the sequence. This phenomenon is called as "learning effect" by Mosheiov (2001) who has used it in some scheduling problem.

It is known that a sequencing problem in MMAL falls into NP-hard class of combinatorial optimization problems and thus a large-sized problem may be computationally intractable (Hyun et al., 1998; Moradi & Zandieh, 2013). So the problem cannot be solved by ordinary methods and more efficient algorithms are needed to solve this kind of problems. The unconscious search algorithm which is proposed by Ardjmand and Amin-Naseri (2012), is based on psychoanalysis theory of Freud and mimics the process of treating the patient by the psychoanalytic. Since the US has been never used to tackle the balancing problem, in this study, a new method based on unconscious search algorithm hybridized with the operators of genetic algorithm (USGA) is proposed to solve the mixed-model assembly line balancing problem with setup times (MMALBPS) by considering some real world constraints. In the past researches, Sequence-dependent setup time, parallel workstation, zoning constraints have been used in the mixed model assembly line balancing problem. In this study, in addition to the mentioned constraints the effect of learning phenomenon is considered in modeling the problem.

The rest of this study is organized as follows. The literature of the mentioned areas is reviewed in section 2. In section 3, the definition of the problem is presented and considered conditions are described by examples. In section 4, the details of the proposed method is discussed step by step and with some examples. Computational results of applying the proposed hybrid USGA algorithm and two other algorithms from the literature to solve different test problems are illustrated in section 5. Finally, conclusion and future research directions are presented in section 6.

2. Literature Review

Metaheuristics has been frequently used to solve mixed-model assembly line balancing problem. This section reviewed some of the recent researches which tackled simple, two-sided, and U-shaped assembly line balancing problems solved mostly by metaheuristics.

Haq et al. (2006) by using obtained solutions from the modified ranked positional weight method as initial population, presented a hybrid genetic algorithm to solve mixed-model assembly line balancing problem which minimizes the number of workstations. By minimizing the balance delay, the smoothness index between stations and the smoothness index within stations for a given cycle time, the mixed-model assembly line balancing problem has been solved using a multi-objective ant colony optimization approach (Yagmahan, 2011). Hamzadayi and Yildiz (2012) presented a priority-based genetic algorithm (PGA) for balancing mixed-model U-shape assembly line with parallel workstations and zoning constraints. Moreover, a simulated annealing based fitness evaluation approach (SABFEA) is developed to calculate the fitness function easier and more effectively. Simulated Annealing algorithm has been applied to solve the mixed-model U-line assembly line balancing problem with parallel workstations in a just-in-time (JIT) production system. Moreover, to increase the efficiency of the proposed approach, some policies for labor assignment have been set (Manavizadeh et al., 2013). Imperialist competition algorithm (ICA) is also applied for solving simple mixed-model assembly line and multi-objective U-type assembly line problems (Moradi & Zandieh, 2013); (Nourmohammadi et al., 2013). In the study of Akpinar and Baykasoğlu (2014b), a multi colony hybrid bees algorithm is applied to solve the mixed-model assembly line balancing problem of type I with some particular features of the real world problems. Yuan et al. (2015) hybridized honey bee mating optimization algorithm with simulated annealing algorithm and proposed a hybrid approach for balancing Mixed-model two-sided assembly lines. Fattahi and Samouei (2016) presented a multi-objective PSO for mixed model assembly line balancing problem. The author considered the task times depended on the worker's skill level when the task must be executed manually. It means that if a high-skilled worker assigned to a task, the task time and the cycle time could be reduced. A modified particle swarm optimization algorithm with negative knowledge has been also proposed by Delice et al. (2017) for solving the mixed-model two-sided assembly line balancing problem. Gansterer and Hartl (2018) evaluated the performance of genetic algorithm (GA), tabu search (TS), and differential evolution (DE) for solving one- and two-sided assembly line balancing problem. Authors also considered three types of constraints, namely zoning constraints, positional constraints, and synchronous constraints. Fattahi and Askari (2018) applied a Multi objective harmony search (MOHS) algorithm to solve mixed-model assembly line (MMAL) problem in a stochastic environment which aims to minimize total utility work cost, total idle cost, and total production rate variation cost, simultaneously. A discrete version of artificial fish swarm algorithm is developed by Zhong et al. (2019) to solve the two-sided assembly line balancing problem. The objective of the problem is to minimize the construction cost and the number of stations. Rabbani et al. (2019) after selecting and prioritizing customer orders

using ANP procedure, and validating the mathematical model by GAMS, used GA and PSO to solve mixed model two-sided assembly lines (MM2SAL) problem. The authors revealed that GAMS is an appropriate software for solving small sized problems, and for large sized problems the performance of GA is better than PSO. Yemane et al. (2020) to improve the productivity of production line, modeled assembly line balancing problem by combining manual line balancing techniques with computer simulation. To evaluate the performance of model, it has been applied to a case study in garment industry.

In addition to the work of Andres which considered sequence dependent setup times in the assembly line balance problem, there are some other researches which solved assembly line balance problem with setups. Özcan and Toklu (2010) proposed a mixed integer program (MIP) to solve two-sided assembly line balance problem with setups (TALBPS). The proposed method minimizes the number of mated-stations as the primary objective and it minimizes the number of stations as a secondary objective for a given cycle time. Seyed-Alagheband et al. (2011) similar to the work of Andres, considered sequence dependent setup times for type II of general assembly line balancing problem where the challenge is to find the minimum cycle time for a predefined number of work stations. They developed a mathematical model and a novel simulated annealing (SA) algorithm to solve the problem and employed the Taguchi method as an optimization technique for tuning the parameters of proposed algorithm. Yolmeh and Kianfar (2012) presented a hybrid genetic algorithm to solve setup assembly line balancing and scheduling problem (SUALBSP) and the operators and parameters of GA are calibrated via a multifactor analyze of variance (ANOVA). Akpinar and Baykasoğlu (2014a) formulated the sequence dependent setup times between tasks in type-I mixed-model assembly line balancing problem and developed a mixed-integer linear mathematical programming for this problem. Li et al. (2019) developed a linear programming model for the two-sided robotic assembly line balancing problem considering sequence-dependent setup times and robot setup times. Moreover, a set of metaheuristics applied to solve the problem and the good performance of two variants of artificial bee colony algorithm and migrating bird optimization algorithm has been shown.

The effect of learning phenomenon in production lines has been evaluated recently by some researchers. Cohen et al. (2006) showed that, under homogeneous learning and relatively smaller lot sizes, unequal allocation of work to stations is better than balanced allocation. Then, developed a non-linear programming model for solving the problem of determining optimal allocation of work to the stations of an assembly line. Toksarı et al. (2008) proved that the simple assembly line balancing problem and U-type line balancing problem with the consideration of learning effects remains polynomially solvable. Toksarı et al. (2010) introduced simultaneous effects of learning and linear deterioration into assembly line balancing problem and developed a mixed nonlinear integer programming model. Hamta et al. (2013) by considering the learning effect on worker(s) performance and sequence dependent setup times, proposed a new method based on the combination of particle swarm optimization (PSO) algorithm with variable neighborhood search (VNS) to solve the single-model assembly lie balancing problem.. In order to investigate the role of learning in the rebalancing of assembly lines with repetitive tasks, Lolli et al. (2017) proposed a cost-based stochastic balancing heuristic by using a time-dependent learning curve. Koltai and Kalló (2017) explored the effects of an exponential learning function on the operation of simple assembly lines and presented an algorithm to determine throughput-time of a production run.

3. Problem Definition and the Mathematical Programming Model

Mixed-model assembly line balancing problem with setups (MMALBPS) which is studied in this paper consists of assigning a set of tasks for a set of models to an ordered sequence of workstations, such that the precedence constraints between tasks are maintained, the setup times between tasks for all models are considered, the number of workstations and variation of workload are minimized. In addition, some other real situations are considered in this study.

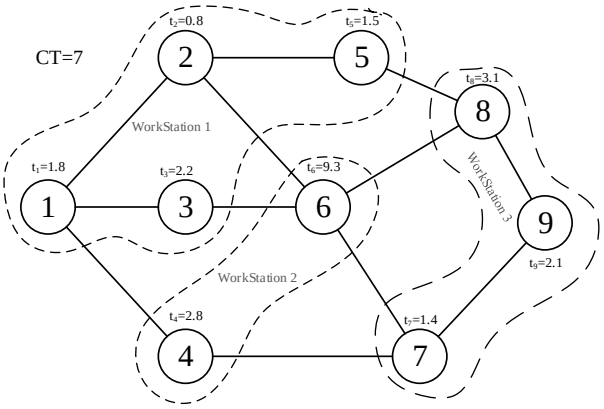
Learning effect causes to reduction in processing times of activities in the workstations. For an example, by processing one job after the other the skills of the workers continuously improve, e.g. the ability to perform setups faster, to deal with the operations of the machines and software or to handle raw materials, components or similar operations of the jobs at a greater pace. So the processing time of tasks in each station will be calculated using learning curve introduced by Biskup (1999), in which the operation time of task i with a learning effect if assigned to position p is defined as:

$$t_{ir} = t_i p^{\alpha} \quad (1)$$

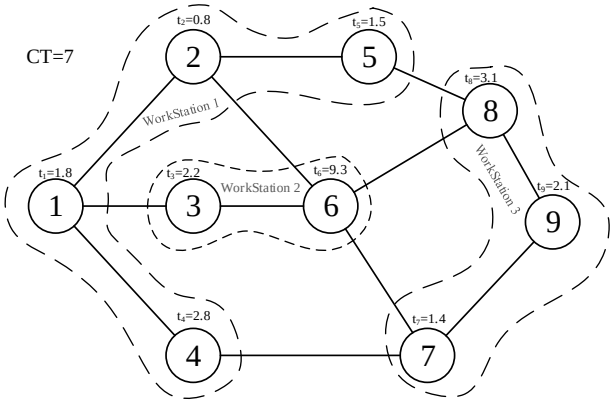
Where $\alpha(= \log_2 S \leq 0)$ is the learning effect when s is the learning rate.

In the following, some examples are explained to describe other realistic considered constraints. Workstations can be expanded and used as parallel workstations, when the time of some tasks is higher than cycle time. There is another real condition in assigning tasks to workstations which is called zoning constraints. Sometimes a certain task should be assigned to a specific workstation or due to the some limitations, assigning a task to a workstation is impossible. So, a feasible solution is a task assignment that has considered mentioned constraints. These constraints and situations are explained by a real case of producing two models in a line. The combined precedence diagram and assignment of tasks to workstations are shown in Fig. 1 and Fig. 2.

a. a feasible solution for assigning tasks to workstations



b. an infeasible solution by considering zoning constraints



c. an infeasible solution that turns into a feasible solution by considering learning effect

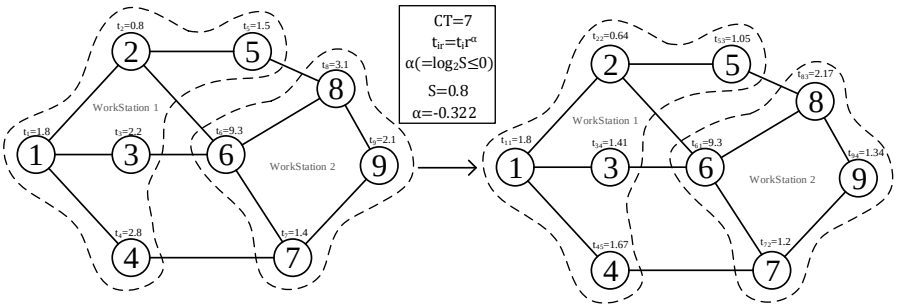


Fig. 1. Feasible and infeasible assignment of precedence diagram

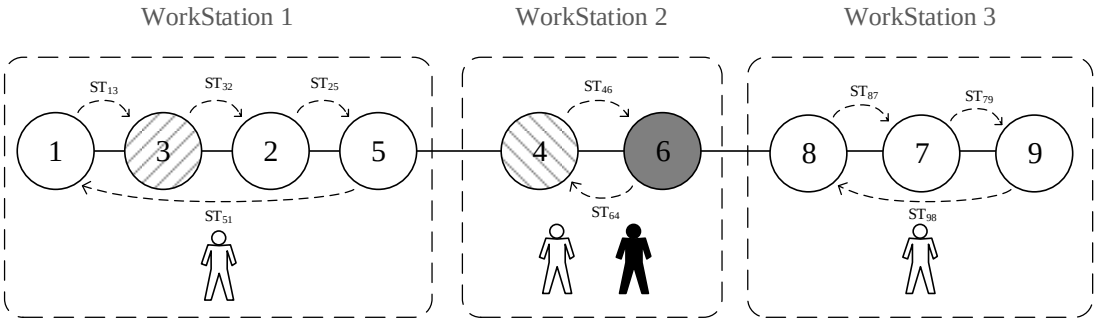


Fig. 2. Tasks ordering and workstations

Some constraints should be considered during the assignment of tasks to workstations in the diagram shown in Fig. 1. Task times in Fig. 1 are sum of the processing times of each model on the particular task. The processing times for the models *A* and *B* are shown in Table 1 where t_A and t_B indicate task times for models *A* and *B*. Since performing task 3 requires a special technology existed in workstation 1, task 3 must be executed in workstation 1. Moreover, due to the lack of some instruments, task 4 cannot be performed in workstation 1. Therefore, the assignment in Fig.1.b is infeasible. For this model, cycle time is calculated as 7 minute (task times are in minutes). Obviously, process time of task 6 is higher than cycle time and existence of parallel workstation is crucial. So as it can be shown in Fig. 2, a worker is added to workstation 2 which can be considered as a parallel workstation. Ordering of tasks in workstation 1 is another remarkable point in Fig.2. Since there is no precedence between task 2 and task 3, two task ordering in this workstation is possible: $\{1-2-3-5, 1-3-2-5\}$. According to the setup times between tasks for both models which are shown in Table 2, the time of each arrangement of tasks is calculated as follows:

$$\{1-2-3-5\}: ST_{12}+ST_{23}+ST_{35}+ST_{51} = 0.81+1.49+0.7+072 = 3.72$$

$$\{1-3-2-5\}: ST_{13}+ST_{32}+ST_{25}+ST_{51} = 0.88+0.75+1.31+0.72 = 3.66$$

where $ST_{12} = ST_{12}^{A} + ST_{12}^{B} = 0.35 + 0.46 = 0.81$. According these calculations, performing of task 3 before task 2 will cause to time saving and it will provide better solution. Similarly performing task 8 before task 7 in workstation 3 causes better solution. Fig. 1. c shows an infeasible assignment of tasks that turns into a feasible assignment after consideration of learning effect due to the reduction of operation times of tasks. Moreover, learning effect causes less workstations in comparison to the solution of Fig. 1. a. As a result, a feasible solution should fulfill the capacity constraints, precedence constraints and zoning constraints and consider learning effect on processing times. Also, taking sequence dependent setup times into account among tasks of a workstation provides better solutions.

Table 1
Task times for model A and B.

Task	1	2	3	4	5	6	7	8	9
t_A	1	0.3	1	1.7	0.6	4.5	0.0	2	1
t_B	0.8	0.5	1.2	1.9	0.9	4.1	1.5	1.2	1.1

Table 2
Setup times for models A and B.

Model	Task	1	2	3	4	5	6	7	8	9
Setup times for model A	1	0.43	**0.35**	0.14	0.33	0.71	0.37	0.53	0.24	0.22
	2	0.15	0.59	0.71	0.70	0.61	0.56	0.54	0.79	0.71
	3	0.64	0.56	0.19	0.45	0.15	0.30	0.49	0.15	0.51
	4	0.06	0.67	0.58	0.79	0.79	0.40	0.43	0.07	0.23
	5	0.30	0.07	0.18	0.02	0.54	0.48	0.54	0.02	0.56
	6	0.35	0.67	0.24	0.49	0.10	0.38	0.07	0.44	0.35
	7	0.39	0.25	0.55	0.46	0.61	0.46	0.72	0.64	0.77
	8	0.39	0.36	0.58	0.75	0.03	0.11	0.17	0.23	0.39
	9	0.33	0.58	0.55	0.50	0.14	0.09	0.01	0.26	0.46
Setup times for model B	1	0.06	**0.46**	0.74	0.52	0.36	0.08	0.13	0.59	0.67
	2	0.07	0.48	0.78	0.67	0.70	0.15	0.49	0.56	0.73
	3	0.18	0.19	0.38	0.54	0.55	0.17	0.59	0.09	0.28
	4	0.35	0.60	0.56	0.46	0.18	0.23	0.37	0.65	0.12
	5	0.42	0.04	0.22	0.11	0.30	0.28	0.13	0.58	0.76
	6	0.10	0.59	0.69	0.01	0.63	0.02	0.39	0.38	0.73
	7	0.66	0.71	0.00	0.42	0.30	0.76	0.73	0.65	0.05
	8	0.78	0.30	0.36	0.22	0.36	0.28	0.35	0.21	0.74
	9	0.73	0.34	0.37	0.15	0.21	0.80	0.51	0.41	0.67

In a MMALB problem with N tasks and M models, the required cycle time C is computed by Eq. (2), and the overall proportion of the number of units of each model q_m is calculated by Eq. (3):

$$C = \frac{P}{\sum_{m\in M} D_m} \quad (2)$$

$$q_m = \frac{D_m}{\sum_{m\in M} D_m} \quad \forall m \in M \quad (3)$$

where P is the planning horizon and D_m is the demand for model m.

With regard to the consideration of learning effect in this study, the modified version of mathematical programming model which is presented by Vilarinho and Simaria (2002) for mixed-model assembly line balancing problem with parallel workstations and zoning constraints is shown in Fig. 1. Due to consideration of learning effect in this model, the operational times of tasks are calculated using Eq. (1) and the constraints (9a) are modified. In the following model x_{ik} is 1 if task i assigned to workstation k is 0 otherwise, r_k is 1 if workstation k can be replicated is 0 otherwise. ZP is the set of task pairs that must be assigned to the same workstation (compatible tasks) and ZN is the set of task pairs that cannot be performed on the same workstation (incompatible tasks). F_i is the set of tasks that cannot be performed before task i is completed. S is the work station index that the last task is assigned, S' is the total number of workstations including replicas, s_{km} is the idle time of workstation k due to model m and S_k is the total proportional idle time of workstation k which due to considering the impact of learning phenomenon is calculated by Eq. (4) as follows:

$$S_k = C - \sum_{m=1}^{M}\sum_{i=1}^{T} t_{irm} = C - \sum_{m=1}^{M}\sum_{i=1}^{T} t_{im}p^{\alpha} \quad (4)$$

where T is the number of tasks that are assigned to workstation k, p is the position of task i in workstation k.

$$minz = \sum_{k=1}^{s} k.x_{Nk} + \frac{S'}{S'-1}\sum_{m=1}^{M} q_m \sum_{k=1}^{S'}\left(\frac{s_{km}}{\sum_{l=1}^{S'} s_{lm}} - \frac{1}{S'}\right)^2 \frac{M}{S'(M-1)}\sum_{k=1}^{S'}\sum_{m=1}^{M}\left(\frac{q_m s_{km}}{S_k} - \frac{1}{M}\right)^2 \quad (5)$$

subject to :

$$\sum_{k=1}^{S} x_{ik} = 1 \qquad i = 1,\dots,N \quad (6)$$

$$\sum_{k=1}^{S} x_{ak} - \sum_{k=1}^{S} x_{bk} \le 0 \qquad a \in N, b \in F_a \quad (7)$$

$$\sum_{k=1}^{S} x_{ak} - \sum_{k=1}^{S} x_{bk} = 0 \qquad (a,b) \in ZP \quad (8)$$

$$x_{ak} + x_{bk} \le 1 \qquad (a,b) \in ZN, k = 1,\dots,S \quad (9)$$

$$\sum_{i=1}^{N} t_{im}.p^{\alpha}.x_{ik} + s_{km} = C[1 + r_k(MAXP - 1)] \qquad k = 1,\dots,S, m = 1,\dots,M,\ p = 1,\dots,N, \alpha \le 0 \quad (10a)$$

$$r_k \le \sum_{i:\exists t_{im} > \gamma C; m=1,\dots,M} x_{ik} \qquad k = 1,\dots,S, 0 \le \gamma \le 100\% \quad (10b)$$

$$\mathcal{M}.r_k \ge \sum_{i:\exists t_{im} > \gamma C; m=1,\dots,M} x_{ik} \qquad k = 1,\dots,S, 0 \le \gamma \le 100\% \quad (10c)$$

$$s_{km} \ge 0 \qquad k = 1,\dots,S, m = 1,\dots,M \quad (11a)$$

$$x_{ik} \in [0,1] \qquad k = 1,\dots,S, i = 1,\dots,N \quad (11b)$$

$$r_k \in [0,1] \qquad k = 1,\dots,S \quad (11c)$$

The objective function (5) minimizes the number of workstations and balances the workload such that the first term minimizes the index of the workstation to which the last task is assigned, thus minimizing the number of workstations. The second and the third term balance the workload between the workstations and the workload within the workstations. Constraints (6) ensure that each task is assigned to only one workstation of the station interval, constraints (7) ensure that no successor of a task is assigned to an earlier station than that task. Constraints (8) and (9) indicate compatibility zoning constraints and incompatibility zoning constraints. The set of constraints (10) ensure that each workstation time capacity is not exceeded, the maximum number of replicas of a workstation (*MAXP*) is not exceeded and only workstations where the processing time of the tasks assigned to it, for at least one model, exceeds a certain proportion ($\gamma\%$) of the cycle time can be duplicated ($\mathcal{M}$ is a very large positive integer). The set of constraints (11) define the decision variables domains.

4. The proposed Unconscious Search Algorithm

Unconscious search algorithm is a metaheuristic based on the theory of psychoanalysis proposed by Freud. According to the theory of psychoanalysis, the human psyche is made up of two main parts, the conscious and unconscious. Consciousness is the subject's immediate apprehension of mental activity. The unconscious is a domain out of reach of the consciousness and includes one's basic instincts, repressed feelings, and thoughts. The theory of psychoanalysis maintains that mental disorders are a consequence of an unbalanced relationship between the conscious and the unconscious (Ardjmand et al., 2014).

Freud found out there are a cause leads to imbalance relationship between the conscious and unconscious called "resistance". The resistance prevents remembering of an event happened in the past of the patient and now is the cause of the mental disorders. In the psychoanalytic psychotherapy proposed by Freud for curing the mental disorders of patient, psychoanalyst has to reveal the contents of the unconscious of the patient by surmounting the resistance of the conscious. It leads to returning the missed balance between the conscious and unconscious of the patient and treating the mental disorders. For this purpose the psychoanalyst asks the patient to talk about a desired subject. Then psychoanalyst according to the conversation of the patient, instructs a subject and asks the patient to talk about the new subject. Thus the psychoanalyst gets experiences about the patient's mind and could instruct better subjects to achieve the unconscious. Since the resistance prevents revealing the unconscious in this process, the psychoanalyst must recognize the impact of resistance and surmounts it by giving the patient the right subject and right direction to follow in free association. The unconscious search algorithm mimicked this process to propose a new method for solving complex optimization problems.

The US has been used for some engineering problems such as pressure vessel design (Ardjmand & Amin-Naseri, 2012) and uncapacitated facility location problem (Ardjmand et al., 2014), but has been never applied to solve assembly line balancing problem. Since the original versions of the US are continuous and inappropriate for solving assembly line balancing problem, in this study, a modified version of US hybridized by genetic algorithm operators (USGA) is applied for MMALBS. The steps of USGA is depicted by Fig. 3

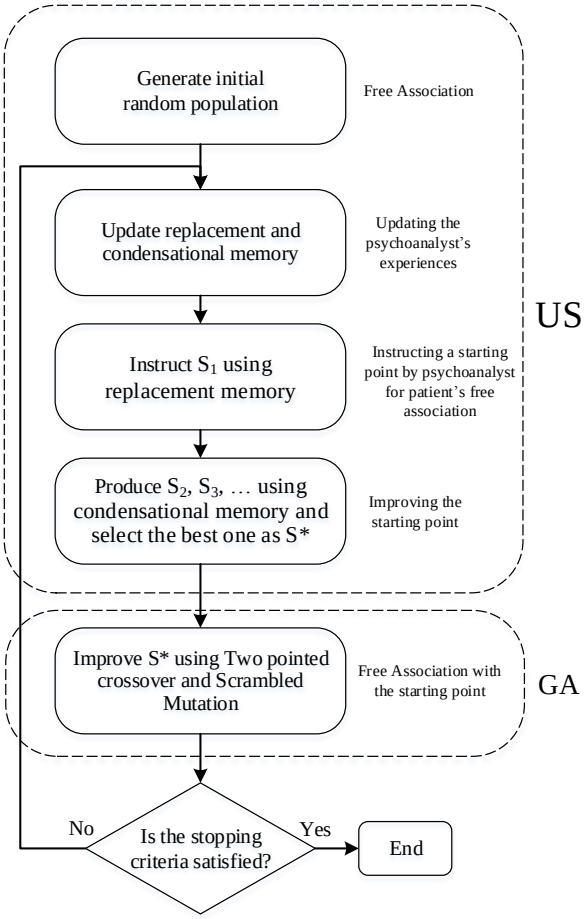


Fig. *3*. Flowchart of USGA

The algorithm starts by generating a pre-defined number of feasible solutions and arranging them based on their values. Then, displacement memory which indicates psychoanalyst's experience and perception about the unconscious of the patient, is filled according to generated solutions. Now that psychoanalyst has gathered information about the unconscious of the patient, he/she can instruct a subject to say anything that comes to patient's mind which is equal to generating a new feasible solution using displacement memory. Then, the patient can talk freely about the mentioned subject which this

freely talking is termed as "free association". Performing a local search around the instructed solution is similar to the free association of the patient. After improving the instructed solution during search process, the direction of conducted changes is traced and saved in condensational memory. In order to overcome the resistance of patient's conscious mind, psychoanalyst uses condensational memory and makes changes in generated solution to create other feasible solution as a starting point for patient's free association. This process repeated for a pre-defined number of iterations to reveal the unconscious of patient by psychoanalyst. The application of US in addressing MMALBPS is proposed in the next sub-sections.

4.1. Generating the initial solutions

In any optimization problem a representation pattern that is usually a vector of decision variables should be defined. In balancing problems this pattern is usually considered as an *N*-dimensional vector which is a permutation of tasks number.

At first, the psychoanalyst asks the patient to talk freely about a desired topic. In fact $|MM|$ random initial solutions ($|MM|$ is the size of measurement matrix) are generated which represent patient's associations. According to precedence constraints, generating a random order of tasks is impossible and a procedure that produces random feasible solutions should be used. Assume that there is a problem with *N* tasks and a $N \times N$ precedence matrix contains of binary numbers. In each step, the columns by all zero amounts represent the tasks without any priority, which means these tasks can be assigned to workstations. If there are more than one task to be assigned, one of them is chosen randomly and is removed from the precedence matrix. This process continued until all the tasks are assigned. Measurement matrix contains the sorted set of $|MM|$ feasible solutions and can be formulated as follow:

$$MM = \{\boldsymbol{P} | C(P_q) < C(P_{q+1}), q = 1,2,\cdots,|MM|\} \quad (12)$$

After generating initial solutions, psychoanalyst has to value each member of MM by use of "translation function" which maps the value of the solution's objective function into a range $(\alpha, 1-\alpha)$ for $\alpha \in (0,1)$. The applied translation function is a sigmoid function defined as follow:

$$f_{t_I}(P_q) = \frac{1}{1 + e^{a(C(P_q))+b}} \quad (13)$$

where a and b can change in each iteration and calculate the proximity of solutions in MM to the optimum solution.

As it was mentioned, the translation function f_{t_I} is designed to map the value of the worst and the best member of MM into α and $(1-\alpha)$. Due to the probability of changing the best and the worst members of MM, parameters a and b have to be calculated in each iteration as follow:

$$a = \frac{2\left(ln\left(\frac{1-\alpha}{\alpha}\right)\right)}{C(P_{worst}) - C(P_{best})} \quad (14)$$

$$b = \left(\frac{C(P_{best}) + C(P_{worst})}{C(P_{worst}) - C(P_{best})}\right)\left(ln\left(\frac{\alpha}{1-\alpha}\right)\right) \quad (15)$$

where P_{best} and P_{worst} denote the best and the worst member of MM.

4.2. Solution quality measure

For quantifying the members of MM by translation function, the algorithm has to use a fitness function which measures each solution's quality. The fitness function which is indicated by Eq. (5) is used as the objective function problem.

4.3. Updating the displacement memory

By means of translation function, psychoanalyst quantifies the members of MM based on their objective functions. Then he\she summarizes the information of MM's members in displacement memory Π. Because of the difference between the representation of solutions in balancing problem with other problems, updating of displacement memory is different and is modified by some changes. In a balancing problem with N tasks, displacement memory is a $N \times N$ matrix. Π can be defined as follow:

$$\Pi = \{\Pi_I, \Pi_E\} \quad (16)$$

in which

$$\Pi_I = \left\{\pi_{I_{ij}} : i = 1,2,\cdots,N,\ \ j = 1,2,\cdots,N\right\} \quad (17)$$

$$\Pi_E = \left\{\pi_{E_{ij}} : i = 1,2,\cdots,N,\ \ j = 1,2,\cdots,N\right\} \quad (18)$$

When the quality of the instructed subject by psychoanalyst for patient's conversation is worse than the

worst member of MM, the instructed solution is saved as a bad solution that leads to high level of resistance and a penalty is considered for this kind of solutions. Π_E is related to this kind of solutions. In fact, displacement matrix consists of two $N \times N$ matrixes, one of them for members of MM, and one of them for solutions that are worse than the worst member of MM. The updating process of both matrixes are same and can be seen in Algorithm 1.

Algorithm 1. The process of updating the displacement memory

Input: fitness function $C(\ )$, Solution P_m, worst member of measurement matrix P_{worst},
Begin
  For all members of measurement matrix
    For all tasks
      If $C(P_m) < C(P_{worst})$
        $\pi_{I_{(P_m(j),P_m(j+1))}} = f_{t_I}(C(P_m))$
      Else if $C(P_m) > C(P_{worst})$
        $\pi_{E_{(P_m(j),P_m(j+1))}} = f_{t_I}(C(P_m))$
      End
    End
  End
End
Output: displacement memory Π updated by P_m

In other word, displacement memory maps the sequence of tasks in all members of MM in the last performed iterations MS (Memory Size). For an example as it shown in Fig. 4, task 2 is performed after task 1 in the first member of MM, so the second column of first row of the displacement memory is filled by $f_{t_I}(C(P_1))$.

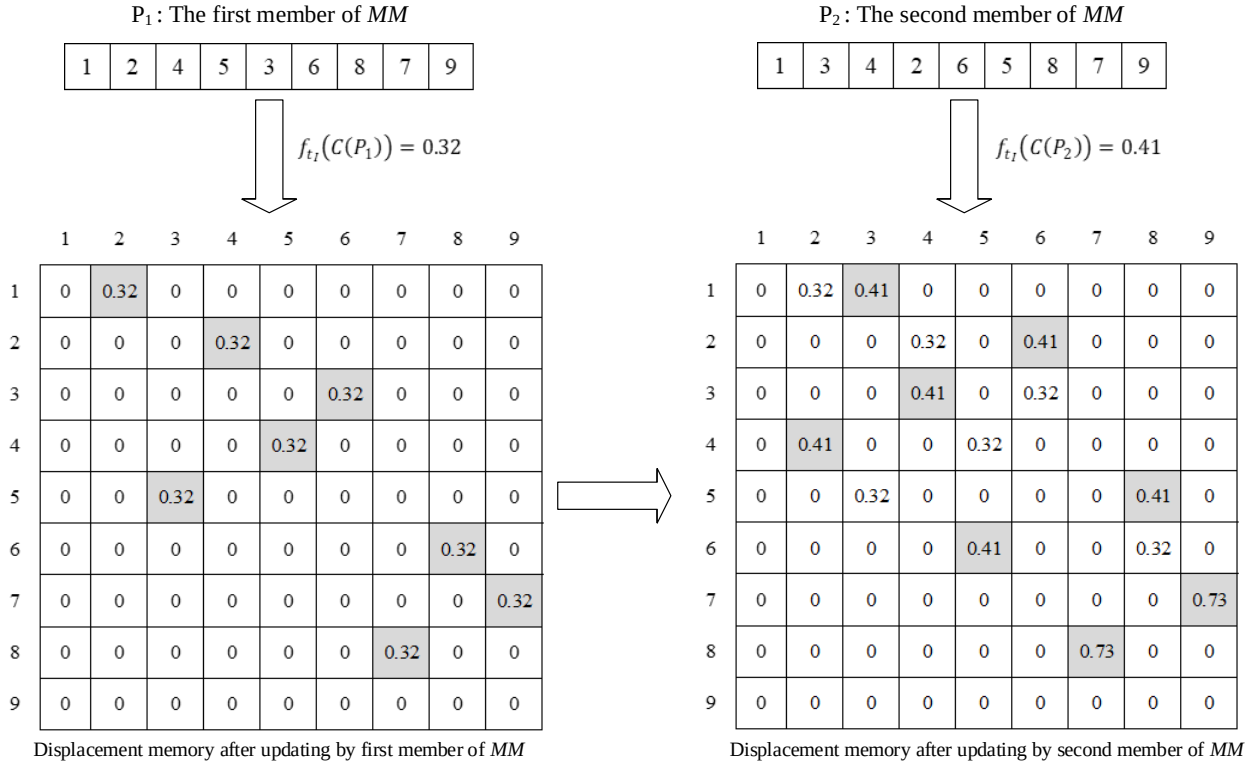


Fig. 4. Process of updating the displacement memory

4.4. Instructing a subject for conversations of patient

After updating the displacement memory a new solution called S_1 can be constructed by psychoanalyst in which the first possible task is selected randomly. Then, according to assigned tasks the set of tasks without precedence determined and by use of Eq. (19) probability of selecting the next task is calculated.

$$Prob(S_1(i) = j) = \frac{\frac{\pi_{I_{ij}}}{1 + \left(\pi_{E_{ij}}\right)^{\beta}}}{\sum_{j=1}^{N} \frac{\pi_{I_{ij}}}{1 + \left(\pi_{E_{ij}}\right)^{\beta}}} \quad (19)$$

where $Prob$ is the probability function and β is a constant and is one the parameters of algorithm.

4.5. Updating the condensational memory

In order to overcome the resistance, psychoanalyst has to define the direction of the resistance for revealing the unconscious of the patient. In the algorithm, condensational memory Π' defines the direction of the resistance. Like displacement memory, condensational

memory is created by two $N \times N$ matrixes. One of them for solutions belong to MM, and the other one for solutions with fitness function higher than the worst member of MM. Condensational memory can be defined as follow:

$$\Pi^{'} = \{\Pi_I', \Pi_E'\} \quad (20)$$

in which

$$\Pi_I' = \left\{\pi_{I_{ij}}' : i = 1,2,\cdots,N,\ \ j = 1,2,\cdots,N\right\} \quad (21)$$

$$\Pi_E' = \left\{\pi_{E_{ij}}' : i = 1,2,\cdots,N,\ \ j = 1,2,\cdots,N\right\} \quad (22)$$

Condensational memory is filled just in cases that a change is made by local search process on the instructed solution by the psychoanalyst. The process of updating the condensational memory is shown in Algorithm 2.

Algorithm 2. The process of updating the condensational memory

Input: Improved solution S_{imp}, Solution S, Sigmoid function $f_{t_I}()$,
Begin
  For all tasks
    Task=$S_{imp}(i)$;
    Position= find$(S = Task)$;
    If $S(Position + 1) \neq S_{imp}(i + 1)$
      $\pi'_{(Task,S_{imp}(i+1))} = \pi'_{(Task,S_{imp}(i+1))} + f_{t_I}\left(C(S_{imp})\right)$
    end
  End
End
Output: Condensational memory Π′ updated by S

For more clarity, the process of updating the condensational memory is explained by an example in Fig. 5.

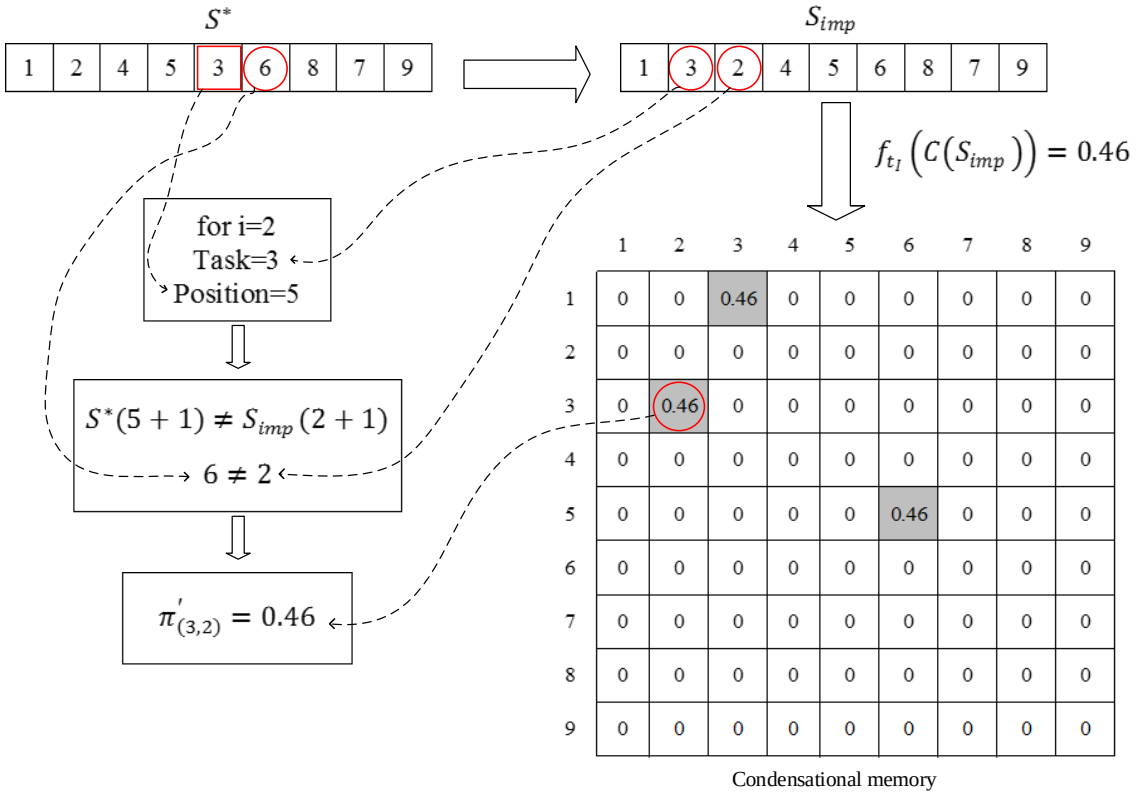


Fig. 5. Process of updating the condensational memory

4.6. Generating more solutions by use of condensational memory

After updating the condensational memory, in order to overcome the resistance, psychoanalyst tries to give the best starting point and direction to patient by instructing solutions based on the condensational memory. By use of the condensational memory the first instructed solution S_1 is modified through the right direction in order to overcome the resistance, and other solutions $S_2, S_3, \ldots$ are generated. This step due to existence of precedence constraints is a little complex. In each step for assigning the next task according to the last dedicated task, two set of tasks are existed: *feasible tasks* and *probable tasks*. Feasible tasks are tasks that considering precedence diagram are possible to be assigned, and probable tasks are defined by cells with positive amounts in the row of the last dedicated task in the condensational memory which means sequence of these tasks in the solution will cause better results. If tasks existed in both feasible tasks and probable tasks, one of them will be selected according to the amounts of the mentioned row of the condensational memory as follow: if the last assigned task called i, the values $v_{ij} = \frac{\pi_{I_{ij}}}{1+\pi_{E_{ij}}}$ will be calculated for positive amount of i'th row in the condensational memory. Then, a random number ψ will be generated. For first amount that meet the condition $\psi > v_{(i,j-1)}$, the task j will be selected for next assignment. Otherwise, when there is no intersection between feasible tasks and probable tasks, there are two possible action. If the assignment of the dedicated task in previous solution after the task considered in this step be possible, the mentioned task will be selected. Otherwise, one of the feasible tasks will be selected randomly. After generation of $S_2, S_3, \ldots$ the best one is chosen and called as S^*. The procedure of producing solutions by use of condensational memory is shown in Fig. 6 and an example of producing these solutions is explained in Fig. 7.

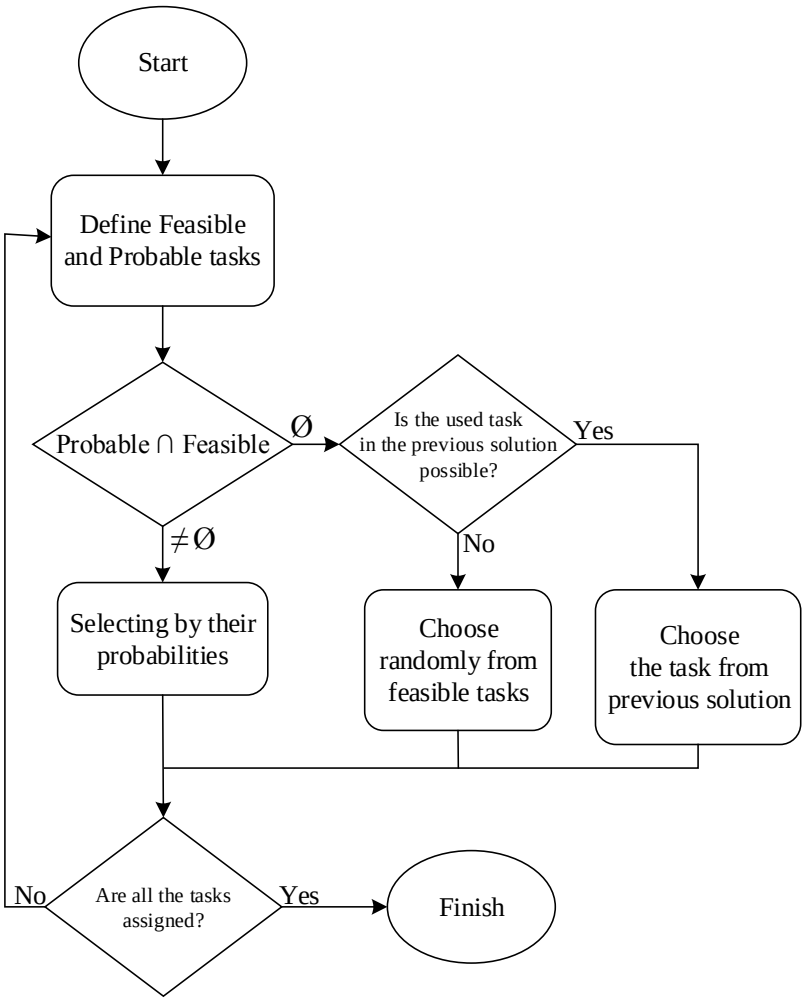


Fig. 6. Procedure of producing solutions by use of condensational memory

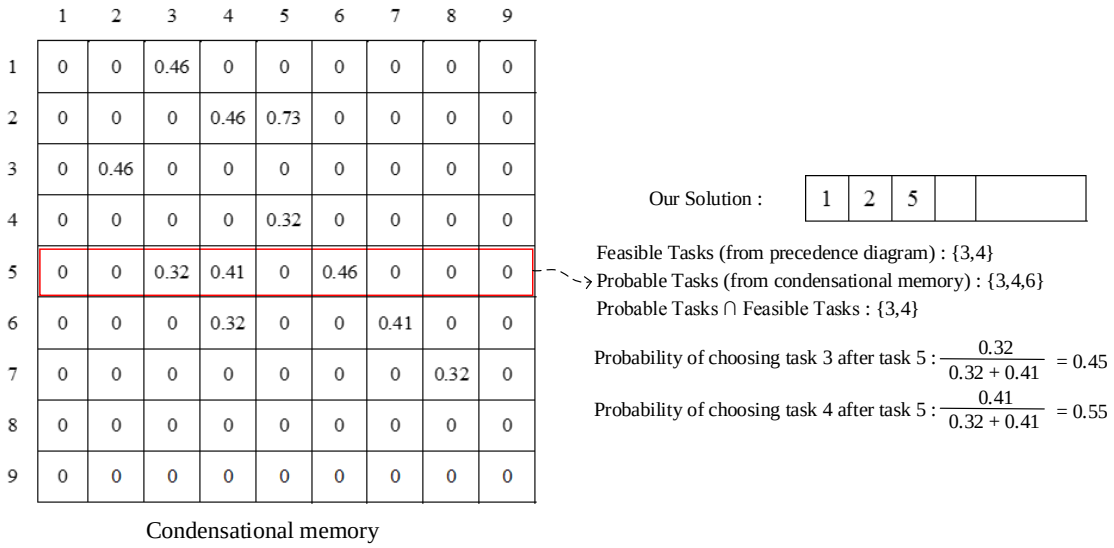


Fig. 7. Example of producing solutions by use of condensational memory

4.7. Free association by patient

S^* is the best instructed solution by psychoanalyst that will be given to patient for free association. In the algorithm, free association of the patient is like a local search progress. *Two point crossover* and *scramble mutation* are operators of ACO-GA algorithm which has been presented for solving the balance problem (AkpıNar et al., 2013). In this study these operators are used instead of a local search for free association of the patient. More explanations of these operators are available in original paper.

The result of these two operators will be S_{imp} that is an improved solution of S^* and obviously $C(S_{imp}) \leq C(S_*)$. If the quality of S_{imp} be better than the quality of the worst member of MM, MM will be updated and parameters a and b need to be recalculated by Eq. (14-15). The process of generation of solutions using displacement and condensational memory will be repeated for a predefined number of iterations to reveal the unconscious or equally to reach the best solution of the problem.

5. Computational experiments

To demonstrate the efficiency and robustness of the proposed USGA algorithm in addressing MMALBPS, its results are compared with ACOGA, which has been proposed by hybridizing operators of GA algorithm with ACO (AkpıNar et al., 2013), and a simple GA with proposed operators in ACOGA. For doing the comparison a set of problems is used which main characteristics of these problems are shown in Table 3. N, M and $C_{original}$ denote the number of tasks of the combined precedence diagram, the number of models, and the cycle time of assembly line that is used in the original paper. Since by use of some of the original cycle times the problem doesn't need parallel workstations, in this study cycle times are considered differently.

Table 3
Main characteristics of the test problems.

	Problem no.	N	M	$C_{original}$	C	Problem name
Small size	1	8	2	20	15	Bowman (1960)
	2	8	3	20	15	Bowman (1960)
	3	11	2	10	7	Gökcen and Erel (1998)
	4	11	3	10	7	Gökcen and Erel (1998)
Medium size	5	25	2	10	10	Vilarinho and Simaria (2002)
	6	25	3	10	10	Vilarinho and Simaria (2002)
	7	30	2	10	10	Akpınar and Bayhan (2011)
	8	30	3	10	10	Akpınar and Bayhan (2011)
Large size	9	35	2	60	32	Gunther, Johnson, and Peterson (1983)
	10	35	3	60	32	Gunther et al. (1983)
	11	45	2	55	30	Kilbridge and Wester (1961)
	12	45	3	55	30	Kilbridge and Wester (1961)
	13	70	2	176	140	Tonge (1960)
	14	70	3	176	140	Tonge (1960)

Original examples did not have setup times and for using these examples for this study, setup times are considered in three level like what AkpıNar et al. (2013) has done in his paper as follows:

- For low variability, the matrix of setup times are generated randomly according to a uniform discrete distribution $U[0,0.25 * (\min t_i)]$.
- For medium variability, the matrix of setup times are generated randomly according to a uniform discrete distribution $U[0,0.5 * (\min t_i)]$.
- For high variability, the matrix of setup times are generated randomly according to a uniform discrete distribution $U[0,0.75 * (\min t_i)]$.

All algorithms GA, ACOGA and USGA are coded in MATLAB2018a and the computational results are obtained by running algorithms 10 times over each test problem with their three setup time variability levels on a Windows 7 platform using Intel Core i7-2670, 2.20 Ghz and 4 GB RAM system. The computational results are shown separately in Tables 4, 5 and 6 based on the levels of setup time variability. The comparison is conducted in terms of total number of workstations concluding replicas (S'), so the minimum, maximum and average of them are presented in Tables 4, 5 and 6. Also, the average of number of workstations $(S_{average})$, the fitness function calculated by Eq. (5), the weighted line efficiency (WE) and the computational time (CPU) are shown in the results tables. The formula of the weighted line efficiency is given by Eq. (23), which takes into account not only the task times but only the sequence dependent setup times between tasks. Hence, the weighted line efficiency is computed by considering idle times of the workstations instead of considering the tasks times. It is noted that maximizing line efficiency is equivalent to minimizing the number of workstations (NWS) for a given cycle time (AkpıNar et al., 2013).

$$WE = \sum_{i=1}^{M} \left[q_m \frac{S \times C - \sum_{i=1}^{S} Idle_{SM}}{S \times C} \right] \qquad (23)$$

where q_m is the overall proportion of the number of units of model m being assembled, $Idle_{SM}$ is the idle time of workstation S due to model m, and C is the given cycle time.

As it can be seen from Tables 4, 5 and 6, almost all algorithms have same performance in case of small size problems, however, in 1 of 4 problems the minimum number of workstations obtained by USGA is lower than the values obtained by GA and ACOGA. In case of medium and large size problems in almost all instances except just one (problem 10 with medium variability), USGA outperforms GA and ACOGA, and the obtained solutions have better fitness function and lower number of workstations. For more details, the performance of USGA (the minimum number of the total workstations) is superior to GA and ACOGA in 60% (6 of 10 problems) in case of low variability setup times, 50% (5 of 10 problems) in case of medium variability setup times, and 80% (8 of 10 problems) in case of high variability setup times. For better perception, Fig. 8 shows the minimum number of total workstations of large size problems with high level of setup time variability. It must be noted that, the obtained fitness function and weighted line efficiency by USGA is higher than values obtained by GA and ACOGA. The calculated weighted line efficiencies of large size problems with high level of setup time variability are shown is Fig. 9. Also it is clear that USGA has slower speed in comparison to pure GA, due to extra computational functions, but has lower computational time in comparison to ACOGA. According to these results, the performance of USGA obviously is superior to pure GA and ACOGA.

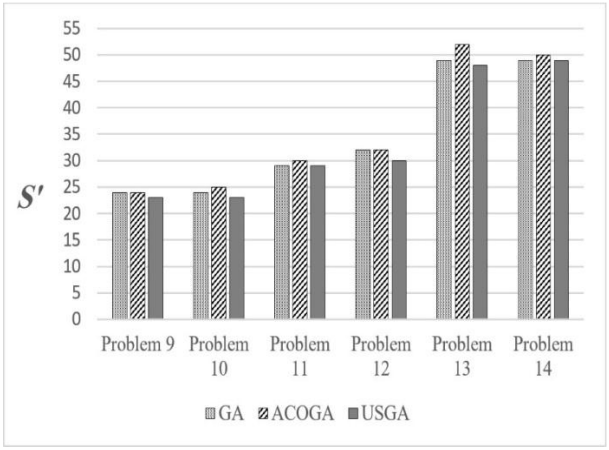


Fig. 8. The minimum number of workstations for large size problems with high level of setup time variability

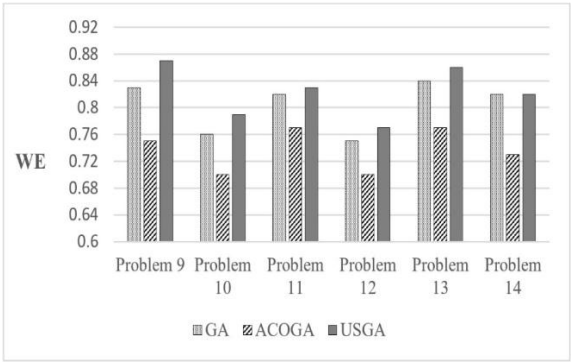


Fig. 9. The weighted line efficiency of large size problems with high level of setup time variability

Table 4
Computational results with low variability of setup times

Problem no.		GA							ACOGA							USGA						
		S'			S_{Avg}	F	WE	CPU	S'			S_{Avg}	F	WE	CPU	S'			S_{Avg}	F	WE	CPU
		Avg	Min	Max					Avg	Min	Max					Avg	Min	Max				
Small size	1	9	9	9	7	10.40	0.68	1.89	9	9	9	7	10.40	0.68	1.72	9	9	9	7	10.40	0.68	0.15
	2	10	10	10	7	11.44	0.73	1.90	10	10	10	7	11.44	0.73	1.75	10	10	10	7	11.44	0.73	0.22
	3	11	11	11	8	12.18	0.84	2.19	12.8	12	14	8.8	13.96	0.73	2.11	10.6	**10**	13	7.5	11.75	0.86	0.54
	4	10.5	10	11	7.5	11.95	0.82	2.12	12.6	11	13	8.8	14.05	0.69	2.14	10.2	10	11	7.2	11.64	0.82	0.70
Medium size	5	19.3	19	20	14.1	20.54	0.83	3.25	21.1	21	22	15.9	22.31	0.78	3.94	19.1	19	20	13.8	20.32	0.84	5.05
	6	19.6	19	20	14.6	21.08	0.78	3.25	21	20	22	15.8	22.49	0.71	3.89	19.2	19	20	14.1	20.68	0.81	3.80
	7	20.7	20	21	16.2	21.90	0.82	5.42	22.4	21	23	17.6	23.61	0.75	6.88	19.9	**19**	20	15.4	21.10	0.86	6.50
	8	21.8	21	23	17.6	23.30	0.71	5.58	23	22	24	18.4	24.50	0.66	6.93	20.9	**20**	21	16.6	22.40	0.74	6.57
	9	23.6	23	24	16.9	25.62	0.85	6.07	25.9	23	27	18.8	27.77	0.79	8.48	23.7	23	24	17.1	25.63	0.85	7.41
	10	24	23	25	17.3	25.49	0.78	6.41	25.8	24	27	18.7	27.29	0.73	7.92	23.7	23	24	17	25.19	0.77	7.29
Large size	11	29.4	29	30	22.9	30.64	0.82	7.92	31.2	31	32	25.1	32.45	0.78	10.88	28.7	**28**	29	22.5	29.94	0.85	9.82
	12	30.4	30	31	23.7	31.90	0.77	7.75	31.4	30	32	25.2	32.91	0.74	10.92	29.1	**29**	30	23.1	30.61	0.79	9.61
	13	47.6	47	49	37.7	65.42	0.83	12.10	53	52	54	42.3	55.98	0.76	18.59	47.4	**46**	48	38.5	50.61	0.86	17.00
	14	48.2	47	50	37.2	49.72	0.80	12.00	52.4	51	53	42.4	53.92	0.73	18.75	47.4	**46**	48	37.7	48.92	0.83	15.80

Table 5
Computational results with Medium variability of setup times

Problem no.		GA							ACOGA							USGA						
		S'			S_{Avg}	F	WE	CPU	S'			S_{Avg}	F	WE	CPU	S'			S_{Avg}	F	WE	CPU
		Avg	Min	Max					Avg	Min	Max					Avg	Min	Max				
Small size	1	9	9	9	7	10.40	0.68	1.89	9	9	9	7	10.40	0.68	1.73	9	9	9	7	10.40	0.68	0.17
	2	10	10	10	7	11.44	0.73	1.88	10	10	10	7	11.44	0.73	1.73	10	10	10	7	11.44	0.73	1.88
	3	12.3	12	13	8.3	13.45	0.77	2.12	13	11	14	9.1	14.18	0.74	2.15	12	11	13	8.3	13.17	0.77	0.56
	4	12.5	12	13	8.5	13.94	0.70	2.15	13.1	13	14	9.1	14.55	0.66	2.12	11.4	**11**	13	8.2	12.85	0.75	0.56
	5	20.4	20	21	15.2	21.64	0.79	3.27	21.5	20	22	16.4	22.72	0.77	3.88	19.7	**19**	21	14.2	20.92	0.82	4.50
Medium size	6	21.3	20	22	16.4	22.79	0.73	3.29	21.9	21	23	16.8	23.39	0.69	4.00	20	20	20	14.9	21.48	0.76	5.58
	7	21.8	21	23	17.5	23.01	0.79	5.68	23.7	23	24	18.7	24.91	0.71	6.96	21.5	21	22	16.9	22.71	0.78	6.83
	8	22.3	22	23	18.1	23.80	0.70	5.50	24	22	25	19.1	25.50	0.63	6.94	21.7	**21**	22	17.5	23.20	0.72	6.63
	9	24.1	23	25	17.3	26.08	0.84	6.15	25.7	24	27	18.5	27.50	0.80	7.95	23.8	23	24	17	25.80	0.85	7.46
	10	24.1	**23**	25	17.2	25.59	0.78	6.20	26.4	24	27	19.3	27.89	0.72	7.90	24	24	24	17	25.48	0.74	7.03
Large size	11	30.5	30	31	24.5	31.75	0.81	8.20	32	31	33	25.4	33.23	0.76	10.78	29.4	**29**	30	23	30.64	0.84	9.87
	12	32	31	33	26.1	33.51	0.75	8.86	32.5	32	34	25.7	34.01	0.70	10.50	29.6	**29**	30	23.3	31.10	0.78	12.63
	13	49.2	48	51	38.1	68.98	0.81	12.18	52.7	50	54	42.5	55.70	0.77	18.68	47.3	**47**	48	38	50.50	0.84	16.02
	14	48.8	47	50	37.6	50.32	0.79	12.24	53	52	54	42.7	54.52	0.73	18.81	47.7	47	48	38.2	49.22	0.80	15.83

Table 6
Computational results with High variability of setup times

Problem no.		GA							ACOGA							USGA						
		S'			S_{Avg}	F	WE	CPU	S'			S_{Avg}	F	WE	CPU	S'			S_{Avg}	F	WE	CPU
		Avg	Min	Max					Avg	Min	Max					Avg	Min	Max				
Small size	1	9	9	9	7	10.40	0.68	1.88	9	9	9	7	10.40	0.68	1.72	9	9	9	7	10.40	0.68	0.17
	2	10	10	10	7	11.44	0.73	1.89	10.1	10	11	7	11.54	0.72	1.73	10	10	10	7	11.44	0.73	1.89
	3	12.3	12	13	8.3	13.45	0.77	2.11	13.1	12	14	9.1	14.28	0.71	2.14	12.2	12	13	8.2	13.33	0.77	0.77
	4	13	13	13	9	14.45	0.68	2.26	13.1	13	14	9.1	14.55	0.66	2.14	12.6	**12**	13	8.6	14.04	0.72	0.58
Medium size	5	20.7	20	21	15.7	21.93	0.80	3.38	22.1	21	23	17	23.32	0.75	3.86	19.6	**19**	20	14.7	20.83	0.83	4.98
	6	22	21	23	17.3	23.50	0.71	3.33	22.7	22	23	17.8	24.20	0.67	3.96	21	**20**	22	15.8	22.49	0.74	4.88
	7	22.2	22	23	17.9	23.42	0.74	5.59	24.3	23	26	19.3	25.51	0.70	7.04	21.8	**21**	22	17.3	23.01	0.77	6.47
	8	23.2	23	24	19	24.71	0.67	5.56	25	24	26	20	26.50	0.62	6.97	22.7	**22**	23	18.5	24.20	0.69	6.68
	9	24.5	24	25	17.5	26.47	0.83	6.14	26.4	24	28	19.2	28.31	0.78	7.97	23.9	**23**	24	17	25.88	0.85	7.36
	10	24.8	24	25	17.8	26.29	0.76	6.28	27.1	25	28	19.6	28.59	0.70	7.97	24.3	**23**	25	17.4	25.79	0.79	7.25
Large size	11	30.6	29	31	24.5	31.85	0.82	9.70	31.7	30	34	24.7	32.93	0.77	11.12	29.9	29	30	23	31.13	0.83	9.93
	12	32.9	32	34	26.9	34.41	0.75	10.24	32.9	32	34	26.4	34.41	0.70	10.57	30.2	**30**	31	23.7	31.70	0.77	10.93
	13	49.9	49	52	38.4	55.64	0.84	12.29	52.7	52	54	42.5	55.69	0.77	18.58	49.5	**48**	51	39	54.46	0.86	16.08
	14	50.6	49	53	38.8	52.12	0.82	12.28	52.8	50	54	42.7	54.32	0.73	18.60	50.7	49	51	39.5	52.22	0.82	15.91

6. Conclusion

This study tried to hybridize unconscious search algorithm with GA for solving mixed-model assembly line balancing problem. Also some realistic conditions have been considered such as zoning constraints, parallel workstations, sequence dependent setup times, and effect of learning phenomenon on operational times of tasks. The performance of the proposed algorithm (USGA) has been compared with the hybrid ACOGA and pure GA over 14 problems with three different levels of sequence dependent setup times. The experimental results reveal that the performance of USGA outperforms ACOGA and GA, and the results are more efficient solutions especially in large size problems.

For future studies, the pure unconscious search algorithm can be applied for other complex optimization problems. Also as a future research direction, the proposed hybrid USGA algorithm is a good choice for solving other types of assembly line balancing problems such as two-sided and U-type assembly line balancing problems.

References

Akpınar, S., & Bayhan, G. M. (2011). A hybrid genetic algorithm for mixed model assembly line balancing problem with parallel workstations and zoning constraints. *Engineering Applications of Artificial Intelligence, 24*(3), 449-457.

AkpıNar, S., Bayhan, G. M., & Baykasoglu, A. (2013). Hybridizing ant colony optimization via genetic algorithm for mixed-model assembly line balancing problem with sequence dependent setup times between tasks. *Applied Soft Computing, 13*(1), 574-589.

Akpinar, Ş., & Baykasoğlu, A. (2014a). Modeling and solving mixed-model assembly line balancing problem with setups. Part I: A mixed integer linear programming model. *Journal of Manufacturing Systems, 33*(1), 177-187.

Akpinar, Ş., & Baykasoğlu, A. (2014b). Modeling and solving mixed-model assembly line balancing problem with setups. Part II: A multiple colony hybrid bees algorithm. *Journal of Manufacturing Systems, 33*(4), 445-461.

Andres, C., Miralles, C., & Pastor, R. (2008). Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. *European Journal of Operational Research, 187*(3), 1212-1223.

Ardjmand, E., & Amin-Naseri, M. R. (2012). Unconscious search-a new structured search algorithm for solving continuous engineering optimization problems based on the theory of psychoanalysis. In *Advances in swarm intelligence* (pp. 233-242): Springer.

Ardjmand, E., Park, N., Weckman, G., & Amin-Naseri, M. R. (2014). The discrete Unconscious search and its application to uncapacitated facility location problem. *Computers & industrial engineering, 73*, 32-40.

Biskup, D. (1999). Single-machine scheduling with learning considerations. *European Journal of Operational Research, 115*(1), 173-178.

Bowman, E. H. (1960). Assembly-line balancing by linear programming. *Operations Research, 8*(3), 385-389.

Buxey, G. (1974). Assembly line balancing with multiple stations. *Management science, 20*(6), 1010-1021.

Cohen, Y., Vitner, G., & Sarin, S. C. (2006). Optimal allocation of work in assembly lines for lots with homogenous learning. *European Journal of Operational Research, 168*(3), 922-931.

Delice, Y., Aydoğan, E. K., Özcan, U., & İlkay, M. S. (2017). A modified particle swarm optimization algorithm to mixed-model two-sided assembly line balancing. *Journal of Intelligent Manufacturing, 28*(1), 23-36.

Fattahi, P., & Askari, A. (2018). A Multi-objective mixed-model assembly line sequencing problem with stochastic operation time. *Journal of Optimization in Industrial Engineering, 11*(1), 157-167.

Fattahi, P., & Samouei, P. (2016). A Multi-Objective Particle Swarm Optimization for Mixed-Model Assembly Line Balancing with Different Skilled Workers. *Journal of Optimization in Industrial Engineering, 9*(20), 9-18.

Gansterer, M., & Hartl, R. F. (2018). One-and two-sided assembly line balancing problems with real-world constraints. *International Journal of Production Research, 56*(8), 3025-3042.

Gokcen, H., & Erel, E. (1997). A goal programming approach to mixed-model assembly line balancing problem. *International Journal of Production Economics, 48*(2), 177-185.

Gökċen, H., & Erel, E. (1998). Binary integer formulation for mixed-model assembly line balancing problem. *Computers & industrial engineering, 34*(2), 451-461.

Gunther, R. E., Johnson, G. D., & Peterson, R. S. (1983). Currently practiced formulations for the assembly line balance problem. *Journal of Operations Management, 3*(4), 209-221.

Hamta, N., Ghomi, S. F., Jolai, F., & Shirazi, M. A. (2013). 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), 99-111.

Hamzadayi, A., & Yildiz, G. (2012). A genetic algorithm based approach for simultaneously balancing and sequencing of mixed-model U-lines with parallel workstations and zoning constraints. *Computers & industrial engineering, 62*(1), 206-215.

Haq, A. N., Rengarajan, K., & Jayaprakash, J. (2006). A hybrid genetic algorithm approach to mixed-model assembly line balancing. *The International Journal of Advanced Manufacturing Technology, 28*(3-4), 337-341.

Hyun, C. J., Kim, Y., & Kim, Y. K. (1998). A genetic algorithm for multiple objective sequencing problems in mixed model assembly lines. *Computers & Operations Research, 25*(7), 675-690.

Kilbridge, M. D., & Wester, L. (1961). A heuristic method of assembly line balancing. *Journal of Industrial Engineering, 12*(4), 292-298.

Koltai, T., & Kalló, N. (2017). Analysis of the effect of learning on the throughput-time in simple assembly lines. *Computers & industrial engineering, 111*, 507-515.

Li, Z., Janardhanan, M. N., Tang, Q., & Ponnambalam, S. (2019). Model and metaheuristics for robotic two-sided assembly line balancing problems with setup times. *Swarm and Evolutionary Computation, 50*, 100567.

Lolli, F., Balugani, E., Gamberini, R., & Rimini, B. (2017). Stochastic assembly line balancing with learning effects. *IFAC-PapersOnLine, 50*(1), 5706-5711.

Manavizadeh, N., Hosseini, N.-s., Rabbani, M., & Jolai, F. (2013). A Simulated Annealing algorithm for a mixed model assembly U-line balancing type-I problem considering human efficiency and Just-In-Time approach. *Computers & industrial engineering, 64*(2), 669-685.

Moradi, H., & Zandieh, M. (2013). An imperialist competitive algorithm for a mixed-model assembly line sequencing problem. *Journal of Manufacturing Systems, 32*(1), 46-54.

Mosheiov, G. (2001). Scheduling problems with a learning effect. *European Journal of Operational Research, 132*(3), 687-693.

Nourmohammadi, A., Zandieh, M., & Tavakkoli-Moghaddam, R. (2013). An imperialist competitive algorithm for multi-objective U-type assembly line design. *Journal of Computational Science, 4*(5), 393-400.

Özcan, U., & Toklu, B. (2010). Balancing two-sided assembly lines with sequence-dependent setup times. *International Journal of Production Research, 48*(18), 5363-5383.

Rabbani, M., Aliabadi, L., & Farrokhi-Asl, H. (2019). A Multi-Objective Mixed Model Two-Sided Assembly Line Sequencing Problem in a Make–to-Order Environment with Customer Order Prioritization. *Journal of Optimization in Industrial Engineering, 12*(2), 1-20.

Seyed-Alagheband, S., Ghomi, S. F., & Zandieh, M. (2011). 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), 805-825.

Thomopoulos, N. T. (1967). Line balancing-sequencing for mixed-model assembly. *Management science, 14*(2), B-59-B-75.

Thomopoulos, N. T. (1970). Mixed model line balancing with smoothed station assignments. *Management science, 16*(9), 593-603.

Toksarı, M. D., İşleyen, S. K., Güner, E., & Baykoç, Ö. F. (2008). Simple and U-type assembly line balancing problems with a learning effect. *Applied Mathematical Modelling, 32*(12), 2954-2961.

Toksarı, M. D., İşleyen, S. K., Güner, E., & Baykoç, Ö. F. (2010). Assembly line balancing problem with deterioration tasks and learning effect. *Expert systems with Applications, 37*(2), 1223-1228.

Tonge, F. M. (1960). A heuristic program for assembly line balancing.

Vilarinho, P. M., & Simaria, A. S. (2002). A two-stage heuristic method for balancing mixed-model assembly lines with parallel workstations. *International Journal of Production Research, 40*(6), 1405-1420.

Yagmahan, B. (2011). Mixed-model assembly line balancing using a multi-objective ant colony optimization approach. *Expert systems with Applications, 38*(10), 12453-12461.

Yemane, A., Gebremicheal, G., Hailemicheal, M., & Meraha, T. (2020). Productivity Improvement through Line Balancing by Using Simulation Modeling. *Journal of Optimization in Industrial Engineering, 13*(1), 153-165.

Yolmeh, A., & Kianfar, F. (2012). An efficient hybrid genetic algorithm to solve assembly line balancing problem with sequence-dependent setup times. *Computers & industrial engineering, 62*(4), 936-945.

Yuan, B., Zhang, C., Shao, X., & Jiang, Z. (2015). An effective hybrid honey bee mating optimization algorithm for balancing mixed-model two-sided assembly lines. *Computers & Operations Research, 53*, 32-41.

Zhong, Y., Deng, Z., & Xu, K. (2019). An effective artificial fish swarm optimization algorithm for two-sided assembly line balancing problems. *Computers & Industrial Engineering, 138*, 106121.

This article can be cited: Asadi-Zonouz, M. Khalili, M. & Tayebi, H. (2021).
A Hybrid Unconscious Search Algorithm for Mixed-model Assembly Line Balancing Problem with SDST, Parallel Workstation and Learning Effect.
Journal of Optimization in Industrial Engineering. 13 (2), 123-140.

http://www.qjie.ir/article_673184.html
DOI: 10.22094/JOIE.2020.579974.1605