Research Article GU J Sci 34 (1): 162-178 (2021) DOI: 10.35378/gujs.690131

Gazi University

Journal of Science

http://dergipark.gov.tr/gujs

Type-2 Assembly Line Balancing with Workload Smoothing Objective: A Reactive Tabu Search Algorithm

Murat ARIKAN*

Gazi University Department of Industrial Engineering, 06570, Ankara, Turkey

Highlights

- Type-2 simple assembly line balancing problem with a workload smoothing objective was considered.
- A reactive tabu search algorithm was suggested to tackle the problem.
- The algorithm's performance was compared with six previously developed meta-heuristics.
- Proposed algorithm presented a higher performance over the existing meta-heuristics.

Article Info

Received: 17/02/2020
Accepted: 17/06/2020

Keywords

Assembly line balancing,
SALBP-2
Workload smoothing
Reactive tabu search
Meta-heuristics

Abstract

This study focusses on a two objective type-2 simple assembly line balancing problem. Its primary objective is minimizing the cycle time, or equivalently, maximizing the production rate of the line. Minimization of the workload imbalance among workstations is considered as the secondary objective. Since the problem is known to be intractable, a reactive tabu search algorithm is proposed for the solution. Although tabu search is a well-known meta-heuristic search procedure, based on the detailed literature survey, there is not a reactive tabu search algorithm to solve the investigated problem. Furthermore, the algorithm utilizes a sequence oriented solution representation which is usually applied by population heuristics such as genetic algorithms and differential evolution algorithms. The performance of the algorithm is tested on several benchmark problems taken from the open literature by comparing both objective values with those of previously developed four particle swarm optimization (PSO) algorithms and two multi-objective genetic algorithms (MOGA). The computational results show that the proposed approach presents a quite encouraging success over the existing meta-heuristics.

1. INTRODUCTION

Assembly lines, which were first introduced by Henry Ford in automobile production, are common manufacturing processes used in high-volume mass production. An assembly line stands for a flow type production system in which the workstations, where tasks are performed, are aligned in series. As the workpieces move along the line, they visit the workstations successively by means of a material handling system, such as a belt conveyor [1]. A well-known principal problem concerning assembly lines is the assembly line balancing problem (ALBP) which is defined as the assignment of tasks that compose the final product, to the workstations to optimize a certain performance criterion. These tasks have to satisfy some constraints when assigned to workstations. The first constraint is the necessity that the work content, i.e. total time of the assigned tasks, of each workstation should not surpass the cycle time. The cycle time is determined by the production rate of the line, which expresses how many products are to be produced within a certain time period. In order to maintain the desired production speed, one product must have been produced at the end of each cycle. The second constraint is that the assignment should not violate the precedence order of tasks. To assign a task to a workstation, its predecessors must have previously been assigned. Another constraint is that each task have to be allocated to a single workstation.

Simple assembly line balancing problem (SALBP), which is classified as NP-hard, is usually categorized into two main subclasses: (i) SALBP-1 (type-1) aims to minimize the number of workstations for a predetermined cycle time, and (ii) SALBP-2 (type-2) tries to minimize the cycle time for a predetermined

* e-mail: marikan@gazi.edu.tr

number of workstations [2]. Simple assembly line balancing problems may be complemented with a secondary objective such as workload smoothing to avoid an unbalanced task assignment between the workstations. It is a frequent practice to assign approximately same workloads to workstations in order to assure equivalent work, hence free time, to each worker. Otherwise, causing unfair working conditions among the workers cannot be avoided [3]. Here, SALBP-2 version of the problem, which takes workload smoothing as a secondary objective into account, is addressed. Nowadays, it still holds the attention of researchers and can be accepted as a valid starting point to solve the re-balancing problem [4].

When the SALBP-2 literature is examined, it can be realized that many researchers first deal with the type 2 assembly line balancing problem using the methods based on repetitive solution of the type 1 problem. In these methods, several cycle times are considered consecutively to check whether it is possible to assign all tasks to a certain number of workstations or not. When such a technique is used, the lower and/or upper limit values for cycle time are computed and the solution is obtained usually with strategies that solve the type-1 problem in a sequential manner, by increasing the cycle time at a certain rate starting from the lower limit [5, 6] or by updating the lower limit and upper limit values in respect of particular rules [7]. Later, approaches that directly solve the type-2 assembly line balancing problems are also suggested. Klein and Scholl [8] developed the branch-and-bound method, SALOME-2, which is an adaptation of SALOME-I and uses a novel enumaration method called Local Lower Bound Method. Uğurdağ et al. [9] proposed a two-phase heuristic based on integer programming for the type-2 assembly line balancing problem. In the first phase, a starting feasible solution is found by the heuristic procedure developed, while in the next phase this starting solution is enhanced using a simplexlike algorithm. Liu et al. [10] proposed two heuristic algorithms which first produce an initial solution using a bi-directional assignment technique, next further improve the acquired initial solution by interchanging tasks among workstations. Kılınçcı [11] presented a petri nets based heuristic, Blum [12] proposed an iterative beam search algorithm. La Scalia et al. [4] suggested a fuzzy binary linear programming model and solution algorithm that can be used when the problem has fuzzy job processing times. Azizoğlu and İmat [13] proposed a branch and bound algorithm to deal with the workload smoothing problem for single model assembly lines in which the workstation number is fixed and the cycle time is pre-specified. Kılınçcı [14] presented a new Petri-nets based algorithm to tackle the type-2 simple assembly line balancing problem in which the firing order is used as a priority rule by implementing backward procedure for task assignment.

Apart from these studies, with the appreciation of the success of the meta-heuristic approaches in the solution of NP-hard problems, they have also been started to be used to solve assembly line balancing problems of type-2. Heinrici [15] presented two algorithms, one simulated annealing (SA) and one tabu search (TS), for the type-2 problem and compared their performance on some test problems from the open literature. Scholl and Voß [16] presented a TS algorithm with several optional elements and showed that the procedure they developed yields much better results than the available constructive methods. Kim et al. [17] adressed various types of simple assembly line balancing problems including of type-2 and suggested a genetic algorithm (GA) for the solution. Nearchou presented differential evolution algorithms (DEA) with objectives, cycle time minimization [18], cycle time and balance delay time/workload smoothness index minimization [19] and compared their performances with two previously developed GAs. Nearchou [20] introduced a new method based on Particle Swarm Optimization (PSO). In the method, two criteria such as cycle time minimization and maximization of workload smoothing are considered simultaneously. Comparisons between the algorithm and two existing multi-objective population heuristics have shown that the proposed approach has a promising high performance. Zacharia and Nearchou [21] proposed a multi-objective genetic algorithm for another version of simple assembly line balancing problem of type-2 with fuzzy task operation times. Zheng et al. [2] developed an advanced ant colony optimization algorithm, called station ant colony optimization, to solve the problem. They confirm the efficiency and stability of the algorithm by comparing it with the literature in 23 samples included in nine examples. Mozdgir et al. [22] proposed a DEA that aims to minimize the workload smoothness index. Zacharia et al. [23] introduced a GA for solving the SALBP-2 for the assembly line of a robotic arm. Triki et al. [24] presented a GA which is hybridised with a local search procedure to solve an extension of SALBP-2 named 'Task Restrictions Assembly Line Balancing Problem' of type 2. Güden and Meral [25] proposed an adaptive SA approach which can be used to solve any deteriministic ALBP including SALBP-2. Their computational experiments on several SALB test problems from the literature showed that for most of the intances optimal solutions are obtained. Zhang et al. [26] developed a novel version of the DEA that directly deals with integer variables to tackle the problem and compared its performance with the DEA and GA algorithms available in the literature. Nearchou and Omirou [27] investigated ten versions of differential evolution approach which differ in the way they explore the search space, on type-1 and type-2 simple assembly line balancing problems. They executed an extensive computational study over a large set of test instances from the literature. Arıkan [28] developed a TS algorithm which uses a diversification strategy based on residence frequencies to solve the type-2 simple assembly line balancing problem with a workload smoothing objective. Akpınar [29] developed a large neighbourhood search algorithm to solve SALBP-2 and evaluated the achievement of the algorithm on a set of test problems. Researchers looking for a more detailed information about the assembly line balancing problems and related studies can refer to Sivasankaran and Shahabudeen [30] and Boysen et al. [31].

Here, a reactive tabu search (RTS) algorithm is suggested for the simple assembly line balancing problem of type-2 with a secondary objective as workload smoothing. Although there are several TS algorithms developed to address assembly line balancing problems in the literature [15, 16, 28, 32, 33], as far as we know, there is not a tabu search algorithm based on reactive tabu search. Another difference of the presented algorithm from the TS algorithms developed in the literature for the same problem (except for [28]) is that it utilizes a sequence oriented solution representation which is usually applied by population heuristics such as GA [17] and DEA [18, 19]. The success of the RTS algorithm is evaluated on several benchmarking instances by comparing cycle time and workload smoothing objective values with those of previously developed four particle swarm optimization (PSO) algorithms and two multi-objective genetic algorithms (MOGA).

The remainder of the paper is structured as follows: Section 2 gives the problem description and formulation. In Section 3, details of the solution methodology are explained. Section 4 discusses results of the computational study performed over test instances taken from the literature. Lastly, conclusions and directions for future research are presented in Section 5.

2. PROBLEM DESCRIPTION AND FORMULATION

The simple assembly line balancing problem deals with a production of a single model and there is a set of tasks belonging to that model whose priorities with respect to each other are determined by a precedence graph. The precedence graph given in Figure 1 (Bowman 1960) consists of 8 tasks. While the numbers inside nodes indicate the task numbers, outside ones refer the task times. According to the precedence graph, task 3, and 4 must be accomplished prior to task 6 can start, task 2 has to be finished in order to perform tasks 3, and 4. The problem investigated in this study addresses the problem of assigning the tasks to workstations by minimizing first, the cycle time, then the workload imbalance. In other words, it is the simple assembly line balancing problem of type-2 with workload smoothing objective. Although there are several functions defined for workload smoothness in the literature [3, 34], we use the total absolute difference of workstation loads from the average workload (*TAD*) as given below [3]:

$$TAD=\sum_{j=1}^{m}\left|W_j-\frac{\sum_{i=1}^{n}t_i}{m}\right| \quad . \tag{1}$$

Here, m, t_i, W_j, and $\sum_{i=1}^{n}t_i$ represent the number of workstations, processing time for task i, workload at workstation j, and total work content, respectively.

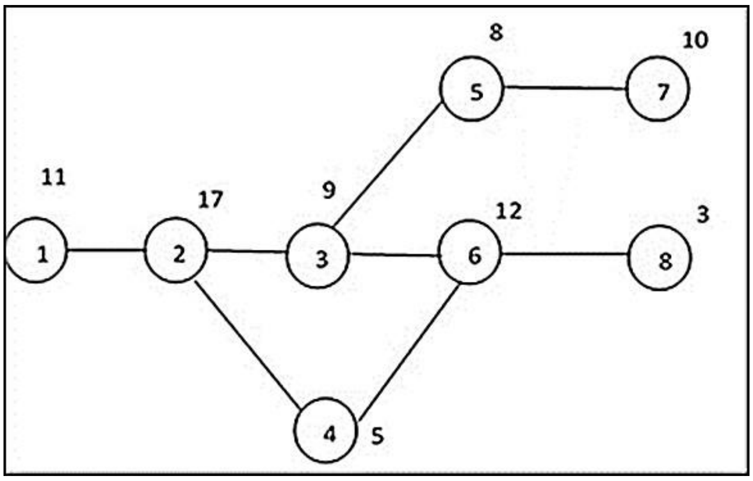

Figure 1. *Precedence graph of Bowman (1960) problem [35]*

The average workload per worker is calculated as $\sum_{i=1}^{n} t_i/3$=25,00 for an assembly line with 3 workstations associated with the precedence graph in Figure 1. In order to make a fair assignment to each worker, it is aimed to minimize the total absolute deviation of workstation loads from the average work content. In Table 1, two altenative task assignments are given and the associated total workload imbalances are calculated. Although the cycle times of two assignments are 28, the smaller total imbalance of the second indicates the workstation loads are closer to each other.

Table 1. *Two alternative task assignments and their corresponding objective values in three workstationed Bowman problem*

	Assignment 1			Assignment 2		
Workstations	Assig. tasks	Worksta. load (W_j)	$\left\|W_j-\frac{\sum_{i=1}^{n} t_i}{m}\right\|$ ($\|W_j$-25$\|$)	Assig. Tasks	Worksta. load (W_j)	$\left\|W_j-\frac{\sum_{i=1}^{n} t_i}{m}\right\|$ ($\|W_j$-25$\|$)
1	1, 2	28	3,00	1, 2	28	3,00
2	3, 4, 6	26	1,00	3, 4, 5	22	3,00
3	5, 7, 8	21	5,00	6, 7, 8	25	0,00
Cycle time		28			28	
Total Imbalance ($\sum_{j=1}^{m}\left\|W_j-\frac{\sum_{i=1}^{n} t_i}{m}\right\|$)			8,00			6,00

Mathematical representation of the problem and related notation are given below [28]:

Notation

n	task number (i=1,…,n)
m	workstation number (j=1,…,m)
t_i	processing time of task i
T	total work content ($\sum_{i=1}^{n} t_i$)
CT_{min}	a lower bound for cycle time
CT_{max}	an upper bound for cycle time
PT_i	set of tasks which precede task i
ST_i	set of tasks which succeed task i
$E_i(CT_{max})$	earliest possible workstation for task i, given a value of CT_{max} ($E_i(CT_{max})=\left\lceil\frac{t_i+\sum_{k\in PT_i} t_k}{CT_{max}}\right\rceil$, (Pastor and Ferrer [36]))
$L_i(CT_{max})$	Latest possible workstation for task i, given a value of CT_{max} ($L_i(CT_{max})=m+1-\left\lceil\frac{t_i+\sum_{k\in ST_i} t_k}{CT_{max}}\right\rceil$, (Pastor and Ferrer [36])
FS_i	set of workstations which task i can be assigned (determined by the [E_i, L_i]

	calculations for each task)
FT_j	set of tasks which can be assigned to workstation j (determined by the [E_i, L_i] calculations for each task)
PR	Set of pairs of tasks such that there is an immediate precedence between them
x_{ij}	1, if and only if task i is assigned to workstation j; 0, otherwise
CT	cycle time
O_j	positive deviation amount from the average workload on workstation j
U_j	negative deviation amount from the average workload on workstation j

Formulation

$$\text{Min } Z_1=CT \tag{2}$$

$$\text{Min } Z_2=\sum_{j=1}^{m}\left(O_j+U_j\right) \tag{3}$$

Subject to

$$\sum_{j\in FS_i} x_{ij}=1 \qquad \forall i \tag{4}$$

$$\sum_{i\in FT_j} t_i . x_{ij}\leq CT \qquad \forall j \tag{5}$$

$$\sum_{j\in FT_j} j . x_{ij}\leq \sum_{k\in FT_j} j . x_{kj} \qquad \forall(i,\, k)\in PR \tag{6}$$

$$\sum_{i\in FT_j} t_i . x_{ij}+U_j-O_j= {}^{T}\!/_{m} \qquad \forall j \tag{7}$$

$$CT\leq Z_1 \tag{8}$$

$$x_{ij}\in\{0,1\} \qquad \forall i,\ \forall j\in FS_i \tag{9}$$

$$CT\geq 0 \tag{10}$$

$$U_j,\ O_j\geq 0 \qquad \forall j. \tag{11}$$

Objective (2) is used to minimize the cycle time, while objective (3) helps to smooth workloads among workstations as defined in Equation (1). Constraint (4) ensures that each task is assigned to a unique station. Constraint (5) guarantees that workloads of each workstation do not exceed the cycle time. Constraint (6) assures that the precedence relationships between tasks are not violated. Constraint (7) specifies the deviations from the average workload for each workstation. Constraint (8) restrains the cycle time with the outcome of objective (2). Constraints (9), (10), and (11) define the variable types and non-negativity conditions.

First, the problem is composed of the constraints (4), (5), (6), (7), (9), (10), (11) and solved under the objective (2) to obtain the minimum cycle time for the assembly line. Then, this obtained cycle time is added as constraint (8) below the model and solved again using the workload smoothing objective (3). Consequently, the most balanced task allocation is attained while cycle time is minimized or synonymously production speed is maximized.

Due to the NP-hard structure of the considered problem, it is difficult to reach an optimal solution with the optimal seeking techniques such as the mixed integer programming model given above when the number of tasks increases. In such cases, it is a common practice to use meta-heuristic techniques to achieve optimal or near optimal results. In this study, a reactive tabu search algorithm is developed for the solution. In the following section, solution methodology and the associated decisions are explained.

3. REACTIVE TABU SEARCH ALGORITHM

TS is a meta-heuristic which directs a local search method to investigate the solution space behind local optimality [37]. TS makes use of the history records gathered while performing the search. The technique is initialized with a starting solution and iteratively tries to achieve solutions with higher quality. At each iteration, a subset is searched among the neighbours of the current solution and the best neighbor is recorded as the current solution, even if it has a poorer objective value. In this way, it becomes possible to escape from getting caught in a local optima. However, the algorithm may cycle as described above. TS prohibits last few movements to restrain cycling and to conduct the search to the new spaces. These movements are identified as tabu and recorded in a structure known as tabu list. An appropiriate size of this list is called as tabu tenure and is crucial for the achievement of the algorithm [38]. RTS is an extension of TS and suggests a simple structure for modifying the tabu tenure according to the properties of the optimization problem. The solutions encountered while searching and their iteration numbers are stored in memory, once the solution is updated with a new move, it can be checked if it is replicated before and the interval between two replications can be calculated. The basic fast "reaction" mechanism augments the tabu tenure when solutions reoccur frequently. This is followed by a slower reduction mechanism. If the current search space does not need a large tabu tenure, it is decreased. These two mechanisms are complemented with an additional Long Term Memory diversification step, called Escape mechanism, when there is an evidence that the search path is restrained in a bounded portion of the search space. The escape phase comes into play when too many solutions frequently reappeared [38]. An elementary escape procedure involves a series of random steps which are carried out starting from the current solution.

In order to implement reactive tabu search algorithm to the considered problem, a number of choises should be made. Those choises are detailed below:

3.1. Encoding

One of the fundamental decisions that affects the success of meta-heuristic techniques like tabu search is determining how to represent the solution. In this study, unlike the simulated annealing and tabu search algorithms developed before, a sequence-oriented solution representation is used which usually applied in evolutionary algorithms proposed for the same problem. In the sequence-oriented solution representation, all tasks are written in the order that the tasks are appointed to workstations. If the order of elements in a sequence-oriented representation does not violate the precedence restrictions, it is defined as a feasible solution. For instance, when the precedence graph in Figure 1 is considered, the solution represented by task sequence <1,2,4,3,5,7,6,8> is feasible. Only the feasible solutions are explored by the developed RTS algorithm.

3.2. Decoding

When the number of workstations is given, the cycle time can be obtained using the technique developed by Kim et al. [17]. The steps of the method are as follows:

1) Calculate an initial base cycle time $BCT = T/m$, which equals to the theoretical minimum cycle time.
2) Pack the tasks up to the $(m - 1)$st workstation by using the cycle time *BCT*. Then assign the rest to the mth workstation.
3) Compute W_j, $(j=1, 2, \ldots, m)$ and W_j^{+}, $(j=1, 2, \ldots, m-1)$. Here, W_j denotes the workload of jth workstation and W_j^{+} stands for the potential workload of workstation j which is the sum of W_j and the operation time of the first task appointed to the $(j + 1)$st workstation.
4) Determine $\mathrm{DT}=\max\{W_j \mid j=1, 2, \ldots, m\}$ and $\mathrm{BCT}=\min\{W_j^{+} \mid j=1, 2, \ldots, m-1\}$. If $DT \leq BCT$ then stop and identify DT as the minimum cycle time. Otherwise, go to Step 2.

3.3. Initial Solution

Either heuristic or random procedure which guarantees the feasibility, can be used to form an initial solution. A method of creating a random and feasible sequence is given below (Kim et al., [17]).

1) Compose an initial list of tasks that have no predecessors, and form an empty array which will represent the initial solution at the end.
2) If the list is empty then stop, else go to Step 3.
3) Choose a task randomly from the list, and add it to the array.
4) Update the current list by deleting the chosen task and by appending every immediate successor of the task if all the immediate predecessors of the successor are already in the array. Go to Step 2.

3.4. Objective Function

The mathematical model has two objectives as the minimization of cycle time and the minimization of workload imbalace, i.e. the sum of absolute differences between workstation contents and average workload. First, the objectives are expressed in a proportional manner and combined as in Equation (12).

$$\text{Minimize } \propto.\left({CT}/{CT_{min}}\right)+\beta.\left({\sum_{j=1}^{m}\left(U_j+O_j\right)}/{t_{sum}}\right) \quad . \tag{12}$$

Here, CT_{min} is the minimum possible cycle time. The first portion of the combined objective corresponds to cycle time and the second portion to workload imbalance. α and β correspond to the weights of objectives. Preliminary experiments show that for the weights α=500 and β=100, the priority of Z_1 over Z_2 is well achieved.

3.5. Neighbourhood Generation Mechanism

Insert and swap moves are employed to generate neighbour solutions from the current one. Insert move refers inserting a task to a different position in the sequence, swap move corresponds to interchanging two tasks. First, the move to be undertaken is decided randomly. Probabilities of insert and swap moves are taken as 0.5 and 0.5, respectively. When a move type is determined, whole neighbourhood of the current solution is explored by applying the same move. Afterwards, the solution part from which the move will be carried out, is established. The move is performed either by selecting any task from the whole sequence or only by considering the tasks in the sequence which are corresponding to the maximal loaded workstation. The related probabilities are decided to be 0.05 and 0.95, respectively. It is possible to state the cycle time by implementing the decoding method given in Section 3.2. Hence, the workstation with the maximum content and the tasks assigned to this workstation can be recognised in any sequence-oriented solution. For each task that is the source of the movement, all candidate positions, which can be performed without deteriorating the feasibility, are specified and recorded to a list. If the move to be carried out concerns the whole solution, only one task is determined randomly to investigate its neighbourhood and all candidates in the list are considered to update the current solution. Otherwise, if the move concerns the tasks assigned to maximal loaded workstation, current solution is updated by exploring the neighbourhood of each pertinent task. Furthermore, in this case, the candidate positions list for each task involves the positions concerning the tasks which are not assigned to the maximal loaded workstation. Sometimes, because of the precedence constraints, no task can be moved to the positions corresponding to a workstation other than the one with the maximum work content in the sequence. In such a case, again the move is performed from any task in the whole sequence. Insert and swap moves are illustrated in Figure 2.

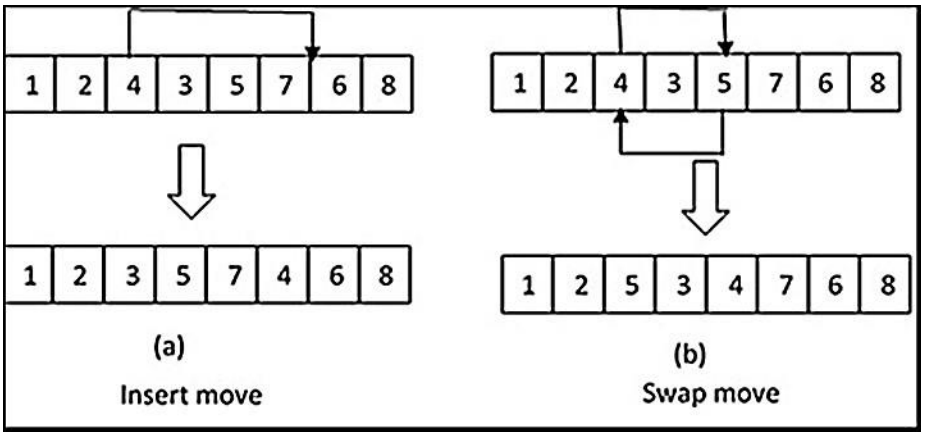

Figure 2. *Insert and swap moves to produce neighbour solutions*

3.6. Tabu Lists and Tabu Tenures

In order to monitor the tabu status of a move, a tabu list of size (n × n) is used which stores their starting iterations. When a move is performed, tabu start times of all tasks whose positions changed are updated. For example, considering the insert move in Figure 2(a), tabu start times of tasks 4, 3, 5, and 7 along with the positions 3, 4, 5, and 6, respectively, are revised. Return of those tasks to corresponding positions is forbidden for a number of iterations which depends on the current tabu tenure. For this purpose, after a move is performed starting time of the tabu situation of the associated positions and tasks with respect to move type are updated as follows:

```
1. Insert move: Task in position k1 is inserted into position k2
    If k1<k2 then begin
      for k:=k1 to k2 do
      begin
      t ← task in position k;
      tabustart[k, t]:=current iteration;
      end;
    end else begin
      for k:=k2 to k1 do
      begin
      t ← task in position k;
      tabustart[k, t]:=current iteration;
      end;
    end;
2. Swap move: Task in position k1 is exchanged with the task in position k2
      t1← task in position k1;
      t2 ← task in position k2;
      tabustart[k1, t1]:=current iteration;
      tabustart[k2, t2]:=current iteration;
```

Additionally, a move is classified as tabu according to the move type if the conditions below are satisfied:

Insert move: Task *t* in position *k1* is inserted into position *k2*,

$$\textit{current iteration} \leq \textit{tabustart}[k2, t] + \textit{tabu tenure} \quad (13)$$

Swap move: Task *t1* in position *k1* is exchanged with task *t2* in position *k2*,

$$\textit{current iteration} \leq \textit{tabustart}[k1, t2] + \textit{tabu tenure} \ \wedge \\ \textit{current iteration} \leq \textit{tabustart}[k2, t1] + \textit{tabu tenure}. \quad (14)$$

3.7. Aspiration Criteria

The simplest form of aspiration criterion is used. A tabu move is adopted if it improves the best solution acquired so far.

3.8. Reaction Mechanisms

RTS uses two reaction mechanisms to maintain a balance between the intensification and diversification of the search. One of those mechanisms controls the tabu tenure, the other one manages the escape procedure. The interval between the revisited solutions and their frequency are monitored to activate these mechanisms. They are summarized as follows:

1) Tabu tenure (*tt*) is dynamically adjusted. If a solution is replicated within a predetermined number of iterations (*CYCLE_MAX*), it is an indication that the search has entered a cycle. In this case, tabu tenure is augmented by a priorly determined factor *INC* where *INC*>1 to avoid short-term cycles that would result in further repetitions. This is combined with a slower reduction mechanism. If there is no repetitions for a adequately long time period, tt is reduced by a priorly determined factor *DEC* where 0<*DEC*<1. This time period corresponds to a moving average (*movAvg*) of observed cycles computed concurrent to the search procedure. Moreover, there is another case in which the tabu tenure is decreased. When tabu tenure increases so much that all possible moves become tabu and none of them meets the aspiration criterion then the tabu tenure is reduced.
2) In case a solution occurred more than *REP* times (*REP* is an predefined parameter), then that solution is classified as a frequently-repeated solution. If the number of frequently-repeated solutions is greater than *CHAOS* (*CHAOS* is a predefined parameter), then it can be stated that the search path is constrained in a bounded segment of the search space. In order to skip out of this space an escape procedure is performed. The standard escape strategy is to execute randomly a number of moves depending on the average gap, in terms of iteration number, between repeated solutions. In this study, standard escape procedure is replaced by a controlled simulated annealing (SA) as Voß and Fink [39] proposes. In our SA implementation, SA parameters containing the initial and final temperatures, cooling factor and number of moves at each temperature level are determined as 100, 30, 0.90, and 1, respectively. Furthermore, only insert moves are used in the escape procedure.

After a series of preliminary experiments the parameters used in the reaction mechanisms are decided as given in Step 0 of the algorithm in Section 3.11.

3.9. Hash Function and Hashing Structures

RTS must be able to effectively identify repeated solutions. However, it is not possible for an efficient algorithm to save and compare all previously visited solutions. Therefore, repetitions are recognized by a computer science technique known as hashing. A two-level open hashing structure is used for the detection process [40]. An effective hash structure should minimize the possibility of collision which means that two different solutions have the same hash function value [41]. Hence, two non-identical task sequences should not be considered as if they were the same. Otherwise, RTS algorithm may work incorrectly. The utilized two-level hashing produces very small probability of collision and described below.

In the first level of hashing, the solutions are stored in the buckets of the hash table whose locations in the array are obtained by a function based on their cycle times (*CT(S)*). The function *f(S)*, given below, appoints a solution to a bucket in the hash table

$$f(S)=[CT(S)] \bmod k\ . \tag{15}$$

Here, $0\leq f(S)\leq k-1$ and hash table is a *k*-dimensional array. In our study, we set *k*=1009 because it is a prime number and does not require excessive storage space. Since multiple solutions may have the same cycle time, only *f(S)* value in the hash table is not sufficient to distinguish between individual solutions. To cope with this deficiency, the additional values associated with each solution are kept in the buckets of the hash table as illustrated in Figure 3. The location of each solution in the array is determined by *f(S)* value. *TAD*, *hv*, *rept no*, and *last found* refer to the total absolute deviation of workstation loads, hashing value, repetition number, and iteration number where the associated solution was last encountered, respectively.

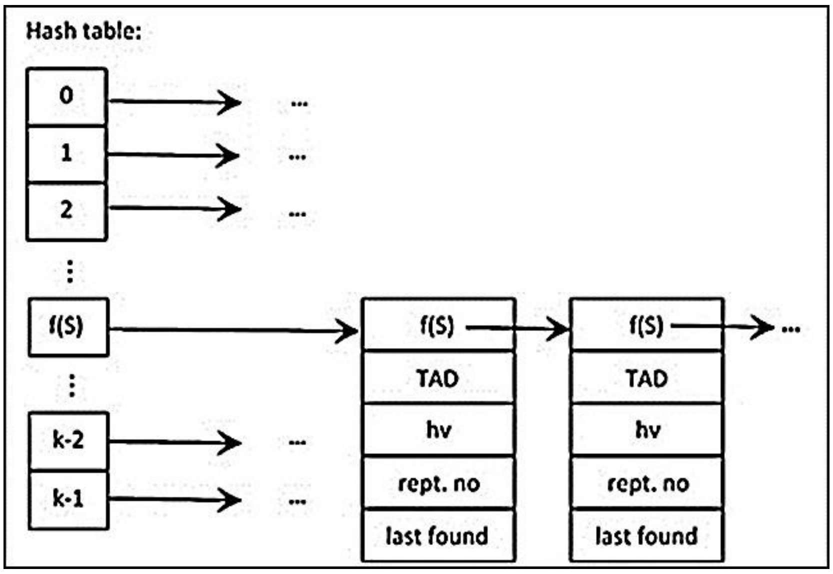

Figure 3. *Two-level open hashing structure*

f(S) links all the solutions with the same *f(S)* in one bucket. If two solutions have the same address in the hash table, then their *TAD* and hashing values (*hv*) are compared successively. If these two values are also the same, it is concluded that these two solutions are identical and the solution is revisited. Hence, *rept no* and *last found* values are updated. Otherwise, if *TAD* is different or *TAD* is same but *hv* does not match they are different solutions, and the new one is linked at the end of the associated bucket. Hashing value is a transformation of each solution sequence into an integer. This is done by the following hash function where *Z[i]* is a random number of task *i* in the range [1, 4300]

$$hv(S)=\sum_{i=0}^{n+1} Z[i]Z[i+1] \ . \tag{16}$$

Z[n+1] is set equal to *Z[1]*. Upper bound for *Z[i]* is determined as 4300 to avoid that the hashing value exceeds longint variable range in Pascal Programming language for any test problem considered.

3.10. Stopping Rule

Two termination criteria are used to stop the algorithm. First one is associated with the iteration number, second one with the run time. The algorithm is terminated either when a particular number of iterations elapsed without improving the current best solution (*iterlim*) or when a predetermined run time is exceeded. The second termination criterion is introduced to restrict run times for large sized problems. *Iterlim* is set proportional to the problem size and achieved by multiplying the number of tasks in the associated problem with 200 (n×200). Run time is set equal to 900 CPU seconds. The algorithm is terminated depending on the stopping rule that occurs priorly.

3.11. Steps of the Algorithm

RTS algorithm steps are as the following.

<u>Step 0:</u> Initialization of the parameters and the tabu list

- *tt*=1; *REP*=2; *CHAOS*=3; *INC*=1.1; *DEC*=0.99; *CYCLE_MAX*=50; *movAvg*=0;

<u>Step 1:</u> Find an initial solution (*S*), save it as the best solution and as the current solution ($S_{best}=S$, $C(S_{best})=C(S)$, $S_{cur}=S$, $C(S_{cur})=C(S)$)

<u>Step 2:</u> Investigate the neighbourhood of the current solution

```
    • Select the move type to be performed
    • Select the solution part from which the move is executed
    • If the move is executed from any position in the sequence then
        o Select any task
        o Without deteriorating the feasibility determine all the candidate positions in the task
          sequence and assign to a list
        o Determine the best solution in the neighbourhood ($S'$) by performing all the candidate
          moves in the list
        o Update the current solution ($S_{cur}=S'$, $C(S_{cur})=C(S')$)
   Else
    • Starting from the first one, for each task assigned to the workstation with the maximum work
      content, repeat the following.
        o Without deteriorating the feasibility determine all the candidate positions in the task
          sequence which do not correspond to the workstation with the maximum work content and
          assign to a list
        o Determine the best solution in the neighbourhood ($S'$) by performing all the candidate
          moves in the list
    • Update the current solution ($S_{cur}=S'$, $C(S_{cur})=C(S')$)
Step 3: Search the current solution ($S_{cur}$) in the hash table
    • steps_since_last_change=steps_since_last_change+1
    • If $S_{cur}$ is found then
        o GAP=currentiter-$S_{cur}$^.lastfound;
        o $S_{cur}$^.reptno=$S_{cur}$^.reptno+1;
        o $S_{cur}$^.lastfound=currentiter;
        o If GAP<CYCLE_MAX then
            ▪ movAvg=0.1*GAP+0.9*movAvg;
            ▪ tt=tt*INC;
            ▪ steps_since_last_change=0;
        o If $S_{cur}$^.reptno>=REP then
            ▪ chaono=chaono+1;
            ▪ If chaono=CHAOS then
                - chaono=0;
                - execute_escape=true;
        o Goto Step 5;
Step 4: Add the current solution to the hash table
    • If steps_since_last_change>movAvg then
        o tt=tt*DEC;
        o steps_since_last_change=0;
        o Goto Step 6;
Step 5: Perform Escape procedure if required
    • If execute_escape=true then
        o Call ESCAPE Procedure;
        o execute_escape=false;
Step 6: Update the best solution, tabu lists and current iteration
    • If $C(S_{cur})<C(S_{best})$ then $S_{best}=S_{cur}$; $C(S_{best})=C(S_{cur})$;
    • Update the tabu list;
    • currentiter=currentiter+1;
    • If stopping rule is satisfied then STOP else goto Step 2;
```

4. EXPERIMENTAL STUDY

The performance of the RTS algorithm is evaluated on a set of benchmark problems which are available on the Web at http://www.assembly-line-balancing.de/salbp/benchmark-data-sets-1993/, by comparing its results against six meta-heuristic algorithms which are previously developed for the considered problem.

Data set includes seven problems with task numbers differing between 29 and 111: Buxey (29,8,7,14), Sawyer (30,8,7,14), Gunther (35,10,6,15), Kilbridge (45,9,3,11), Tonge (70,23,3,25), Arcus1 (83,20,3,22), Arcus2 (111,25,3,27). The numbers in the parenthesis indicate the task numbers, the number of test instances contained in the relevant ALBP, minimum and maximum number of workstations of test instances in the relevant ALBP, respectively. This means, for example, for Buxey problem 8 different test instances with workstation numbers between 7 and 14, are considered. The RTS algorithm is coded in TurboPascal programming language and all experimentation are executed on a Pentium Intel Core2 Quad 2.67 GHz computer with a 4 GB RAM memory. In order to secure the impartiality of the experiments, the algorithm was run 10 times with different random seeds on each test instance with the parameter values given in the previous sections and the solution quality was averaged over all instances of each test problem. That is, RTS algorithm was run (8+8+10+9+23+20+23)x10=1010 times in total.

Existing meta-heuristics which the achievement of the suggested RTS algoritm is compared, include four versions of PSO algorithm developed by Nearchou [20], a Pareto-niched GA presented by Kim et al. [17] to deal with multi objective (MO) SALBPs (MOGA1), and a Pareto weight-sum GA proposed by Murata et al. [42] to tackle MO flow-shop scheduling problems (FSSPs) (MOGA2). The results of the existing meta-heuristics are taken from the Nearchou [20]. Nearchou [20], who dealt with type-2 assembly line balancing problem considering the same secondary objective function as in our study, takes 7 test problems into account. Therefore, the same problems are addressed in this study.
Table 2 gives the comparison of the RTS algorithm against the existing meta-heuristics. The first and second column of the table indicates the test problems and their sizes. The information provided by the remaining columns are explained below:

best.c%dev= the average relative deviation from optimum/best in percentage; calculated by $(\frac{c\text{-}c^*}{c^*})\times 100$ where c^* is the actual optimal or actual best cycle time known so far and c is the best cycle time attained by a particular meta-heuristic,

avg.c%dev= the average relative deviation from optimum/best in percentage; calculated by $(\frac{avg.c\text{-}c^*}{c^*})\times 100$ where *avg.c* is the average cycle time attained by a particular meta-heuristic

max.c%dev= the maximum relative deviation from optimum/best in percentage; calculated by $(\frac{max.c\text{-}c^*}{c^*})\times 100$ where *max.c* is the worst cycle time attained by a particular meta-heuristic

MAD= mean absolute deviation of workstation loads from the average workload which is calculated by dividing the total deviation given in Equation (1) to the number of workstations ($MAD=\frac{1}{m}\sum_{j=1}^{m}\left|W_j\text{-}\frac{\sum_{i=1}^{n}t_i}{m}\right|$).

In Nearchou's [20] study, *avg.c%dev* and *max.c%dev* values of the related meta-heuristics are reported. In addition to those values, here, we also reported average deviations of best cycle times from the optimum (*best.c%dev*) to comment about the robustness of the algorithm. In Table 2, it can be seen that the best performance in terms of the primary objective (cycle time) was acquired by the proposed RTS algorihm. Even the *max.%devs* of the RTS algorithm are better than the avg.*c%devs* of the existing meta-heuristics. In regard with the secondary objective (workload smoothing), RTS algorithm achieved the highest performance in 6 out of the 7 test problems. Only, in Arcus1, RTS came 3rd with a *MAD* of 76,49 behind PSO3 with a *MAD* of 63,89 and MOGA1 with a *MAD* of 75,26.These observations are summarized in Figures 4 and 5. Figure 4 illustrates the avg.c%devs obtained by each meta-heuristic. Additionally, average of worst cycle times (*max.c%dev*) for the proposed RTS algorithm are also displayed. Figure 5 demonstrates the average *MAD* values. For each separate benchmark problem, last two columns from the right in Figure 4 and last column from the right in Figure 5 are associated with the results of the RTS algorithm. Success of the RTS algorithm can be clearly seen.

Another criterion used when evaluating the success of meta-heuristic algorithms is the robustness of the algorithm. When Table 2 is examined, it is seen that the difference between the *best.c%dev* and *max.c%dev* values obtained by the RTS algorithm on the test problems are quite small. The largest gap belongs to Arcus1 with 0.32% (the corresponding *max.c%dev* and *best.c%dev* values are 0.44% and 0.12%, respectively). For the Kilbridge problem, this difference is 0. This means that in 9 instances dealt with for the Kilbridge problem, the optimal result was achieved in all of the 10 trials with different random numbers. As a result, the RTS algorithm is quite robust.

Table 2. *Comparison of the outcomes acquired on benchmark problems (results for the existing meta-heuristics are taken from Nearchou [20])*

Problem	N	Method	*best.c%dev*	*avg.c%dev*	*max.c%dev*	*MAD*
Buxey	29	PSO1	-	1,96	7,14	1,13
		PSO2	-	5,88	12,00	1,72
		PSO3	-	1,55	4,00	1,29
		PSO4	-	5,01	12,00	1,47
		MOGA1	-	8,79	17,86	2,71
		MOGA2	-	7,86	22,22	2,13
		RTS	**0,00**	**0,19**	**0,27**	**0,87**
Sawyer	30	PSO1	-	2,24	3,85	1,14
		PSO2	-	6,62	12,00	1,65
		PSO3	-	1,60	3,85	1,12
		PSO4	-	5,61	12,00	1,52
		MOGA1	-	1,90	7,14	1,13
		MOGA2	-	3,23	7,69	1,13
		RTS	**0,00**	**0,24**	**0,27**	**0,72**
Gunther	35	PSO1	-	0,87	5,00	2,73
		PSO2	-	3,88	7,94	3,46
		PSO3	-	0,64	5,00	3,45
		PSO4	-	3,92	10,00	3,11
		MOGA1	-	8,84	20,37	5,04
		MOGA2	-	2,64	11,11	3,74
		RTS	**0,00**	**0,02**	**0,19**	**2,18**
Kilbridge	45	PSO1	-	0,80	1,79	0,77
		PSO2	-	1,64	4,35	1,11
		PSO3	-	0,68	1,79	0,50
		PSO4	-	1,78	4,84	1,06
		MOGA1	-	10,47	16,13	8,30
		MOGA2	-	0,94	1,79	0,73
		RTS	**0,00**	**0,00**	**0,00**	**0,30**
Tonge	70	PSO1	-	2,91	9,54	5,56
		PSO2	-	5,99	14,69	8,69
		PSO3	-	2,03	7,65	6,01
		PSO4	-	5,82	14,12	7,66
		MOGA1	-	1,60	8,02	3,25
		MOGA2	-	4,14	10,90	4,66
		RTS	**0,04**	**0,09**	**0,14**	**1,06**
Arcus1	83	PSO1	-	1,42	4,22	86,03
		PSO2	-	3,20	7,13	120,98
		PSO3	-	0,70	2,07	63,89
		PSO4	-	3,04	7,23	101,63
		MOGA1	-	0,73	1,67	75,26
		MOGA2	-	6,78	15,69	358,34
		RTS	**0,12**	**0,26**	**0,44**	**76,49**
Arcus2	111	PSO1	-	3,79	8,37	197,24
		PSO2	-	6,67	15,31	308,36
		PSO3	-	2,02	8,07	90,01
		PSO4	-	5,77	13,21	277,12
		MOGA1	-	1,88	6,74	76,21
		MOGA2	-	10,85	21,29	749,35
		RTS	**0,10**	**0,19**	**0,39**	**22,65**

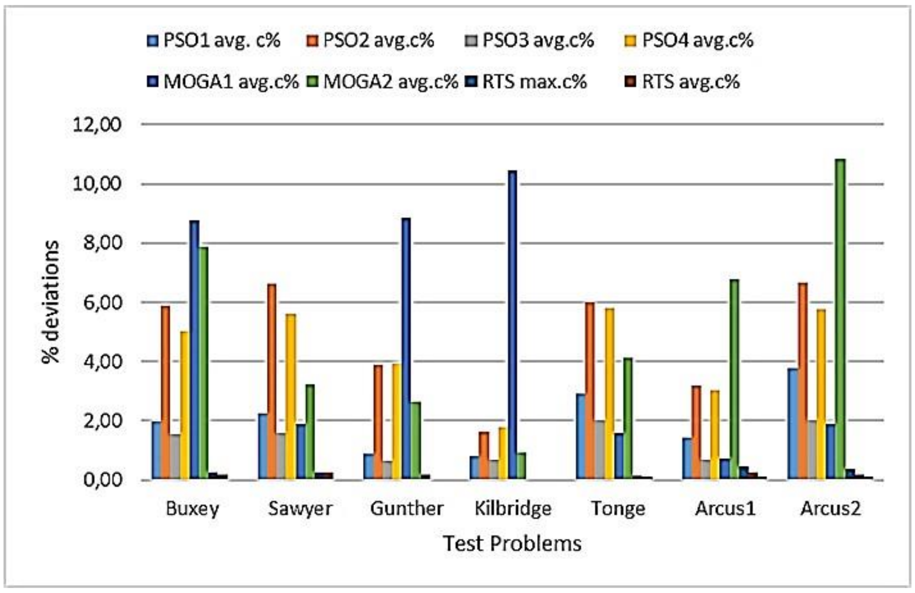

Figure 4. *Comperative results for the cycle times*

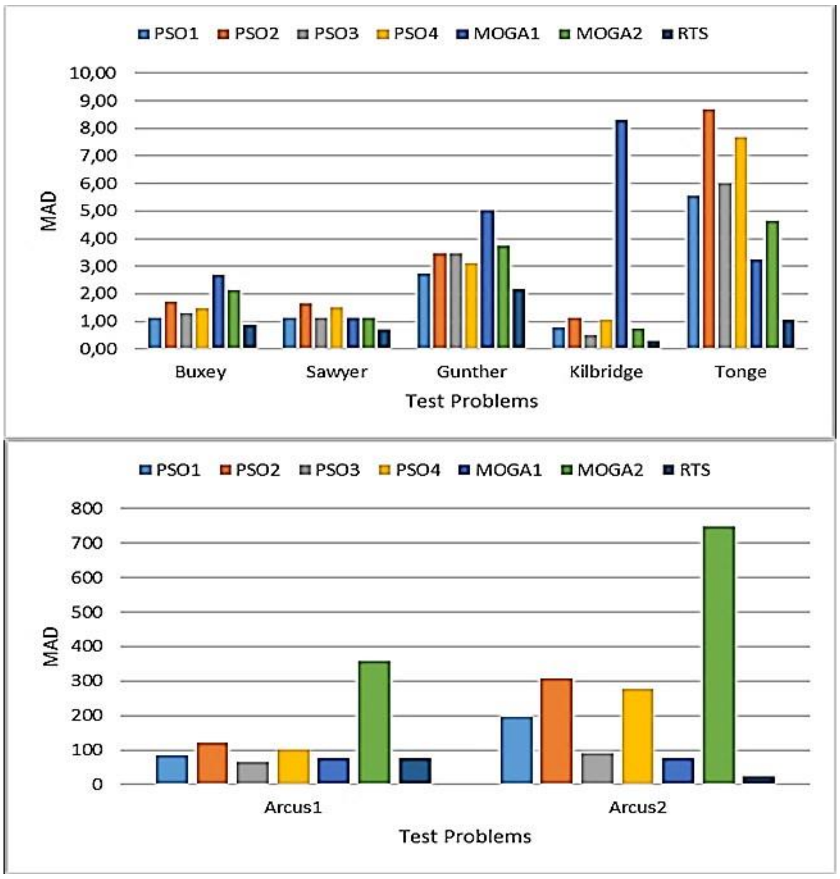

Figure 5. *MAD values obtained over the considered test problems*

5. CONCLUSION

In this study, type-2 simple assembly line balancing problem with a secondary objective as workload smoothing is tackled by a RTS algorithm. The most significant differences of the suggested algorithm from the other TS algorithms in the literature which are proposed for the considered problem, are twofold: (1) It is based on reactive tabu search and (2) it employs a sequence oriented solution representation which is usually applied by population heuristics. The only tabu search algorithm that uses this type of solution representation belongs to Arıkan [28]. Proposed algorithm's performance is demonstrated on 7 well-known benchmark problems taken from the open literature by comparing the results for both objectives with those of previously developed four particle swarm optimization (PSO) algorithms and two multi-objective genetic algorithms (MOGA). The experimental outcomes show that the proposed approach surpasses the performance of the existing meta-heuristics.

For future research, the algorithm can be designed to apply to other forms of ALBPs. Moreover, the effect of different solution representations on tabu search and other meta-heuristic techniques can be investigated.

CONFLICTS OF INTEREST

No conflict of interest was declared by the author.

REFERENCES

[1] Boysen, N., Fliedner, M. and Scholl A., "A classification of assembly line balancing problems", Eur. J. Oper. Res., 183: 674-93, (2007).

[2] Zheng, Q., Li, M., Li, Y. and Tang, Q., "Station ant colony optimization for the type 2 assembly line balancing problem", Int. J. Adv. Manuf. Tech., 66: 1859-1870, (2013).

[3] Rachamadugu, R. and Talbot, B., "Improving the equality of workload assignments in assembly lines", Int. J. Prod. Res., 29 (3): 619-633, (1991).

[4] La Scalia, G., Rosa, M., Giuseppe, A., Mario E., "Solving type-2 assembly line balancing problem with fuzzy binary linear programming", J. Intell. Fuzzy Syst., 25: 517-524, (2013).

[5] Mastor, A.A.," An experimental investigation and comparative evaluation of production line balancing techniques", Manage. Sci. 16(11): 728-746, (1970).

[6] Gehrline, W.V. and Patterson, J.H., "Sequencing for assembly lines with integer task times", Manage. Sci., 21(9): 1064-1070, (1975).

[7] Hackman, S.T., Magazine, M.J. and Wee, T.S., "Fast, effective algorithms for simple assembly line balancing problems", Oper. Res., 37: 916-924, (1989).

[8] Klein, R. and Scholl, A., "Maximizing the production rate in simple assembly line balancing – A branch and bound procedure", Eur. J. Oper. Res., 91: 367-385, (1996).

[9] Uğurdağ, H.F., Rachamadugu, R. and Papachristou, C.A., "Designing paced assembly lines with fixed number of stations", Eur. J. Oper. Res., 102: 488-501, (1997).

[10] Liu, S.B., Ong, H.L. and Huang, H.C., "Two bi-directional heuristics for the assembly line type II problem", Int. J. Adv. Manuf. Tech., 22: 656-661, (2003).

[11] Kılınçcı, Ö., "A petri-net based heuristic for simple assembly line balancing problem of type 2", Int. J. Adv. Manuf. Tech., 46: 329-338, (2010).

[12] Blum, C., "Iterative beam search for simple assembly line balancing with a fixed number of work stations", Stat. Oper. Res. T., 35(2): 145-164, (2011).

[13] Azizoğlu, M. and İmat, S., "Workload smoothing in simple assembly line balancing", Comput. Oper. Res., 89: 51-57, (2018).

[14] Kılınçcı, Ö., “Petri-net based algorithm for maximizing production rate in assembly lines”, J. Fac. Eng. Archit. Gaz., 35(2): 753-763, (2020).

[15] Heinrici, A., “A comparison between simulated annealing and tabu search with an example from the production planning”, In: Dyckhoff H, Derigs U, Salomon M, Tijms HC, editors. Operations Research Proceedings 1993, Berlin: Springer-Verlag, 498-503, (1994).

[16] Scholl, A. and Voß, S., “Simple Assembly Line Balancing – Heuristic Approaches”, J. Heuristics, 2: 217-244, (1996).

[17] Kim, Y.K.., Kim, Y.J. and Kim, Y., “Genetic algorithms for assembly line balancing with various objectives”, Comput. Ind. Eng., 30(3): 397-409, (1996).

[18] Nearchou, A.C., “Balancing large assembly lines by a heuristic based on differential evolution method”, Int. J. Adv. Manuf. Tech., 34: 1016-1029, (2007).

[19] Nearchou, A.C., “Multi-objective balancing of assembly line by population heuristics”, Int. J. Prod. Res., 46(8): 2275-2297, (2008).

[20] Nearchou, A.C., “Maximizing production rate and workload smoothing in assembly lines using particle swarm optimization”, Int. J. Prod. Econ., 129: 242-250, (2011).

[21] Zacharia, P.T. and Nearchou, A.C., “Multi-objective fuzzy assembly line balancing using genetic algorithms”, J. Intell. Manuf., 23: 615-627, (2012).

[22] Mozdgir, A., Mahdavi, I., Badeleh, I.S. and Solimanpur, M., “Using the Taguchi method to optimize the differential evolution algorithm parameters for minimizing the workload smoothness index in simple assembly line balancing”, Math. Comput. Model., 57: 137-151, (2013).

[23] Zacharia, P.T., Tsirkas, S.A., Kabouridis, G. and Giannopoulos, G.I., “Planning the construction process of a robotic arm using a genetic algorithm”, Int. J. Adv. Manuf. Tech., 79: 1293-1302, (2015).

[24] Triki, H., Mellouli, A., Hachicha, W. and Masmoudi, F., “A hybrid genetic algorithm approach for solving an extension of assembly line balancing problem”, Int. J. Comput. Integ. M., 29(5): 504-519, (2016).

[25] Güden, H. and Meral, S., “An adaptive simulated annealing algorithm-based approach for assembly line balancing and a real-life case study”, Int. J. Adv. Manuf. Tech., 84: 1539-1559, (2016).

[26] Zhang, H., Yan, Q., Liu, Y. and Jiang, Z., “An integer-coded differential evolution algorithm for simple assembly line balancing problem of type 2”, Assembly Autom., 36(3): 246-261, (2016).

[27] Nearchou, A.C., and Omirou, S.L., “Assembly Line Balancing Using Differential Evolution Models”, Cybernet. Syst., 48(5): 436-458, (2017).

[28] Arıkan, M., “A tabu search algorithm for the simple assembly line balancing problem of type-2 with workload balancing objective”, J. Fac. Eng. Archit. Gaz., 32(4): 1169-1179, (2017).

[29] Akpınar, Ş., “Large neighbourhood search algorithm for type-II assembly line balancing problem”, Pamukkale University Journal of Engineering Sciences, 23(4): 444-450, (2017).

[30] Sivasankaran, P. and Shahabudeen, P., “Literature review of assembly line balancing problems”, Int. J. Adv. Manuf. Tech., 73: 1665-1694, (2014).

[31] Boysen, N., Fliedner, M. and Scholl, A., “Assembly line balancing: Which model to use when?”, Int. J. Prod. Econ., 111: 509-528, (2008).

[32] Chiang, W.C., “The application of a tabu search metaheuristic to the assembly line balancing problem”, Ann. Oper. Res., 77: 209-227, (1998).

[33] Lapierre, S.D., Ruiz, A. and Soriano, P., “Balancing assembly lines with tabu search”, Eur. J. Oper. Res., 168: 826-837, (2006).

[34] Driscoll, J. and Thilakawardana, D., “The definition of assembly line balancing difficulty and evaluation of balance solution quality”, Robot. Cim-Int. Manuf., 17: 81-86, (2001).

[35] Scholl, A., “Data of assembly line balancing problems”, Shriften zur Quantitativen Betriebwirtschaftslehre 16/93, TH Darmstadt, (1993).

[36] Pastor, R. and Ferrer, L., “An improved mathematical program to solve simple assembly line balancing problem”, Int. J. Prod. Res., 47(11): 2943-2959, (2009).

[37] Glover, F. and Laguna, M., “Tabu Search”, Boston, Kluwer Academic Publishers, (1997).

[38] Battiti, R. and Tecchiolli, G., “The reactive tabu search”, ORSA Journal on Computing, 6(2): 126-140, (1994).

[39] Voß, S. and Fink, A., “A hybridized tabu search approach for the minimum weight vertex cover problem”, J. Heuristics, 18(6): 869-876, (2012).

[40] Carlton, W.B. and Barnes, J.W., “Solving the traveling salesman problem with time windows using tabu search”, IIE Trans., 28: 617-629, (1996).

[41] Woodruff, D.L. and Zemel, E., “Hashing vectors for tabu search”, Ann. Oper. Res., 41: 123-137, (1993).

[42] Murata, T., Ishibuchi, H. and Tanaka, H., “Multi-objective genetic algorithm and its application to flowshop scheduling”, Comput. Ind. Eng., 30(4): 957-968, (1996).