Journal of Quality Engineering and Production Optimization

Vol. 7, No. 1, Winter & Spring 2022

http://jqepo.shahed.ac.ir

Research Paper

DOI: 10.22070/JQEPO.2021.13856.1175

Landscape Analysis and a Hybrid Iterated Local Search (HILS) for Solving Simple Assembly Line Balancing Problem type 2 (SALBP2)

Somaye Ghandi [1*], Nafiseh Ghazavi [1]

[1] *Department of Industrial Engineering, Faculty of Engineering, University of Kashan, Kashan, Iran*

**** Corresponding Author:*** *Somaye Ghandi (Email: n.ghazavi9053@gmail.com)*

Abstract *– The Assembly Line Balancing (ALB) problem is one of the subproblems of the Assembly Planning(AP) problem and is defined as the process of partitioning the assembly operations into a set of tasks and assigning them to assembly workstations such that all workstations approximately have equal times. The most basic model in ALB is the Simple Assembly Line Balancing Problem for type 2 (SALBP2). The mentioned problem is an NP-hard problem and thus many researchers in the field tried to find an effective and efficient solution for it. However, the fitness landscape of this problem has not been yet studied despite the existence of numerous works on solving it. In this article, different statistical correlation and distribution measures are used and calculated in order to analyze the fitness landscape of the SALBP2 problem for 44 test problems. The results reveal that the problem's landscape is approximately uniform based on the distribution of the locally optimal assembly sequences. Therefore, for obtaining an effective and efficient solution to SALBP2 a suitable Hybrid Iterated Local Search (HILS) is designated and used to solve a number of SALBP2 problems. Comparison results with the other approaches in the SALBP2 literature represent that the HILS produces the optimal or best known solutions on most problem instances, and it performs better than other algorithms.*

Keywords*– Assembly Line Balancing (ALB), Hybrid Iterated Local Search (HILS), Landscape Analysis, Multi-objective Optimization, Simple Assembly Line Balancing Problem for Type 2 (SALBP2).*

I. INTRODUCTION

The Assembly Line Balancing (ALB) problem consists of assigning *n* tasks *i* with times t_i to an ordered sequence of *m* assembly workstations *k* stations such that the precedence relations among the tasks are satisfied and the assembly times of all workstations are nearly equal.

In (Rashid et al., 2012) the studies in ALB that appeared in 2001–2011 and used soft computing approaches has been surveyed. Also, in (Battaïa and Dolgui, 2013)'s recent research on balancing flow lines within many different industrial contexts has been analyzed. Their survey covers about 300 studies on line balancing problems that appeared during 2007–2012. The ALB problem is NP-hard (Scholl and Becker, 2006), and is divided into two classes:

1. SALBP: Simple Assembly Line Balancing Problem is appropriate for single-sided assembly lines in which a unique model of a single product is created (Capacho Betancourt, 2007). SALB is categorized into some sub-problems. First, SALB for type 1 (SALBP1) minimizes the number of workstations (*m*) for a predefined

Manuscript Received:13- Dec -2021 & Revised: 18- March -2022 & Accepted: 30- April -2022
ISSN: 2423-3781

cycle time (*c*) (Sotskov et al., 2015). Second, type 2 (SALBP2) considers the minimization of cycle time with a given number of workstations (Sungur and Yavuz, 2015). It should be mentioned that problems such as general resources-constrained assembly line balancing problem (GRCALBP) which minimize cycle time and resource usage for a given number of stations (Alakaş, 2021) are an extension of SALBP2. The next problem is SALB for type E (SALBP–E) which maximizes the line efficiency (*E*) by simultaneously minimizing *c* and *m* due to their mutual relationship (Esmaeilbeigi et al., 2015). Another class is SALB for type F (SALBP–F) which based on a given combination of *m* and *c* investigates the existence of a feasible line balance (Kriengkorakot and Pianthong, 2012).

2. GALBP: Generalized Assembly Line Balancing Problem is appropriate for balancing two-sided assembly lines which are more complex than the single–sided assembly lines. GALB is categorized into some sub-problems. First, two-sided assembly line balancing (2S–ALB) in which almost all operating lines consist of a pair of head-on workstations (Yadav et al., 2019, Sepahi and Naini, 2016). Second, the two-sided mixed-model assembly line balancing problem (MALB) produces several models of a basic product in an intermixed sequence (Razali et al., 2019; Liu et al., 2021; Buyukozkan et al., 2016). Another class is the U–line assembly line balancing problem (UALBP) in which a single product is produced in the stations that are arranged within a narrow U (Chantarasamai and Lasunon, 2021; Alavidoost et al., 2016).

In (Scholl and Becker, 2006) a comprehensive survey of the SALBP2 is completed. Also, A systematic review of hot topics and research themes in assembly line balancing during 1990–2017 is presented in (Eghtesadifard et al., 2020). Also, based on the literature review, the solution approaches to SALBP2 include three following categories:

(1) Exact- Only a few exact solution approaches have been developed which directly solve SALBP2. These procedures include Lower bounds (McNaughton, 1959; Klein and Scholl, 1996; Merengo et al.,1999), Branch and Bound (B&B) (Klein and Scholl, 1996; Li et al., 2021) and Logic-Based Benders Decomposition (Zohali et al., 2021) procedures.

(2) Heuristic- These approaches include Constructive procedures (Hackman et al.,1989) and a constraint programming approach (Pinarbasi et al., 2019).

(3) Metaheuristic- These approaches include Genetic algorithms (GA) (Anderson and Ferris, 1994; Watanabe et al., 1995), tabu search (TS) (Heinrici, 1994; Chiang, 1998), simulated annealing (SA) (Heinrici, 1994), Ant Colony Optimization (ACO) (Zheng et al., 2013), and Variable Neighbourhood Strategy Adaptive Search (VaNSAS) (Jirasirilerd et al., 2020).

The literature review revealed that several heuristic and metaheuristic approaches to SALBP2 have been proposed in the last two decades. However, to decide on a suitable metaheuristic for SALBP2, the fitness landscape characteristics of the problem must be investigated. Generally, fitness Landscape Analysis (FLA) is an efficient and important approach for finding the most appropriate metaheuristic for any problem (Eghtesadifard et al., 2020). FLA describes the amplitudes, correlation and distribution of the local optimal solutions to the problem.

In fact, in addressing any combinatorial optimization problem, no specific algorithm can perform better than all other algorithms. It is necessary to mention that although the FLA has been done for various issues and problems in articles such as (Malan and Engelbrecht, 2014; Beham et al., 2017; Jana et al., 2017; Wang et al., 2017), no research has been performed for analyzing the fitness landscape of the SALBP2 Problem. The only article which considered the fitness landscape of the SALBP is (Nourmohammadi et al., 2019) in which the landscape of SALBP1 has been analyzed for a precedence graph with 7 tasks. Based on the literature review, the contributions and/or innovations of this paper are as follows:

1- The landscape of the SALBP2 problem has not been studied and analyzed so far despite the large number of metaheuristics applied to solve it. On the other hand, the landscape properties affect the effectiveness of used metaheuristics for solving the problem. Therefore, in this paper, some statistical measures including correlation and distribution measures are used to analyze the landscape of the SALBP2 problem for the first time in the literature.

2- A hill–climbing approach is developed and run on population *U* which is a set of 500 starting assembly line balances in order to generate the Population *O* which is a set of 500 local optimums.

3- The correlation and distribution measures are calculated and analyzed for 14 sample assembly line balancing type 2 test problems with various workstations. Altogether 44 test problems have been analyzed.

4- Based on the distribution and correlation measures analysis, the problem's nature is specified and an appropriate Hybrid Iterated Local Search (HILS) algorithm is implemented for the SALB2 problem.

The rest of this paper is organized as follows: the fitness landscape of the SALB2 problem is analyzed in Section 2. In Section 3, the proposed HILS algorithm is described and illustrated for the SALBP2. In Section 4, computational results and discussion are presented. Finally, conclusions and summarizing remarks are prepared in Section 5.

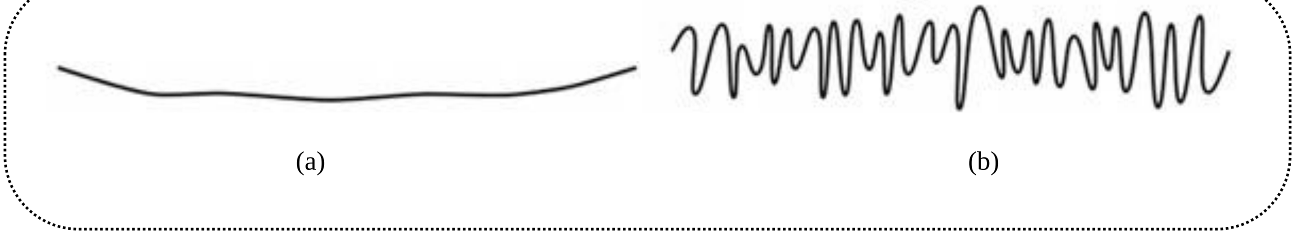
II. SALB2 PROBLEM FITNESS LANDSCAPE ANALYSIS

The properties of the landscape must be considered in applying metaheuristics to a problem. Because the search space of that problem influences the efficiency and effectiveness of the metaheuristic to address the considered problem. These properties are the representation of the solution, neighborhood generation, and objective function. The search space of the problem could be assumed as a floor from a geographical point of view. Therefore the landscape of a problem is comprised of valleys, plains, cliffs, valleys, plateaus, peaks, canyons, basins, etc. as shown in Fig. (1) (Talbi, 2009). Depending on the landscape's shape, specific types of metaheuristic algorithms will be more effective.

III. METHODOLOGY

This paper proposes a linear mathematical model for optimization decisions in production planning of AKBOR GRP Pipe Production Company which produces GRP pipes. AKBOR produces pipes from diameter 300-4200 mm with pressure range of 1-40 bar and stiffness range of 2500 N/m2 to 10000 N/m2. GRP pipes generally are being used in irrigation projects, sewage, and potable water projects. The better production planning will help company to produce more pipes with optimal costs. For this reason, paper makes the best multi period multi products model.

The flow of material and product are assumed as shown in figure 1. As figure 1, five raw materials as silica sand, glass fiber, resin, cobalt and styrene are purchased and entered to the raw materials warehouse. The raw materials are then fed into the production line, and the finished product is delivered to the final product warehouses. The final customer will receive the order from the final warehouse after purchasing the merchandise. It is considered to produce only a single product with different diameters from 300-4200 mm.



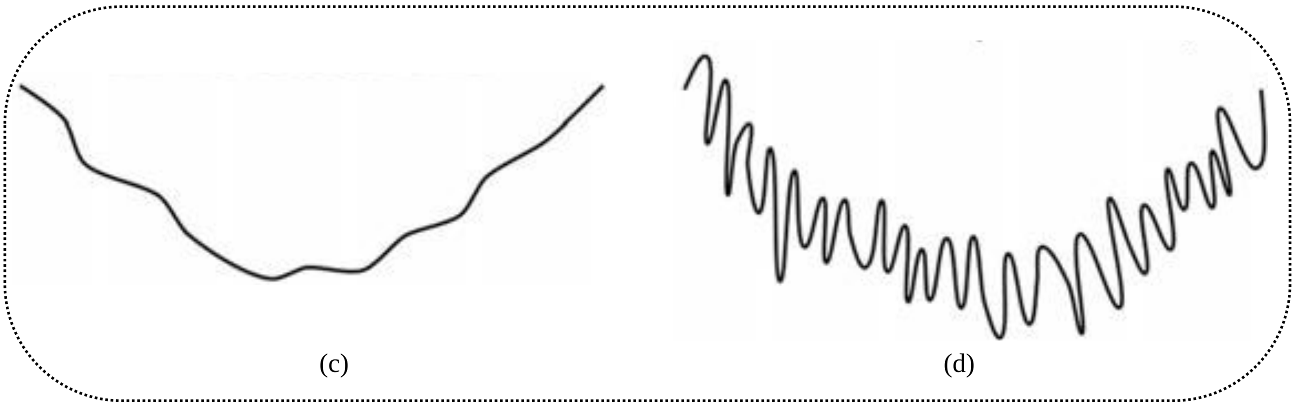


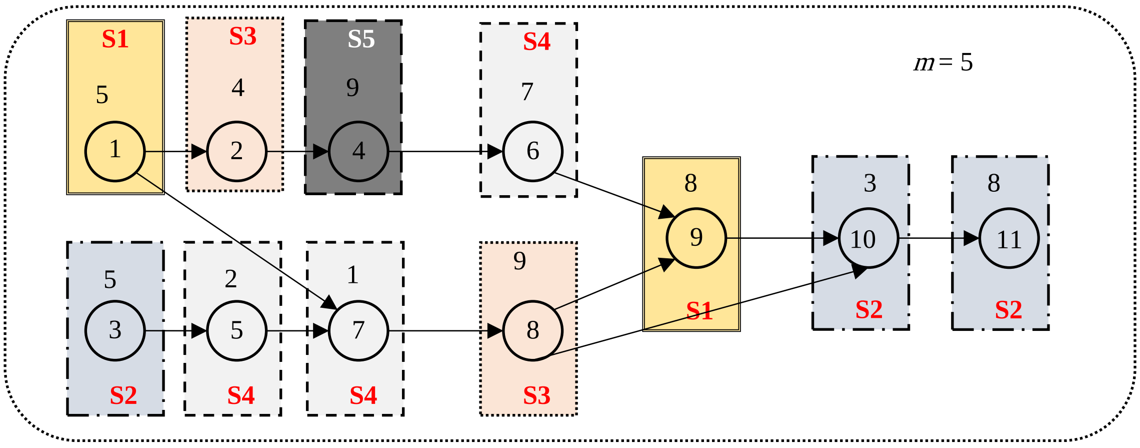
Fig 1. The geographical shape of different types of landscape, (a) Flat plain, (b) Rugged plain, (c) Basin valley, (d) Rugged valley (Talbi, 2009).

In this paper, indices such as distribution and ruggedness of the fitness function are computed and analyzed. The local optima's distribution in the landscape of the SALBP2 is explained by these indices. Based on the results, a concept about the landscape shape is formed which guides us in selecting the appropriate type of metaheuristic in the considered context. Landscape studies are based on 14 sample assembly line balancing type 2 test problems with various workstations. Altogether 44 test problems have been analyzed. The test problems are taken from http://www.assembly-line-balancing.de/sualbsp/data-set-of-scholl-et-al-2013, where many test problems are available for testing the developed approaches for solving the ALB problem. Also, a hill–climbing approach is developed and run on population U which is a set of 500 starting assembly line balances. For each starting solution in U, the Local Search is applied until achieving to a local optimum. Therefore, population O is generated which is a set of 500 local optimums. After that, statistical measures such as the correlation and distribution of local optimums in the search space of the SALB2 are calculated and finally, the fitness landscape analysis is performed based on the statistical measures for the studied test problems. The following subsections describe each component of the landscape analysis of the SALB2 Problem.

A. SALBP2 symbols

In order to manufacture a product, the total amount of work must be partitioned into a set of n tasks. Each task j needs a task time tj to Perform. Considering the precedence relationships between the tasks is essential due to the technological and organizational conditions. These elements are visualized and summarized by a precedence graph, and its corresponding Assembly Precedence Matrix (APM). In the assembly precedence graph each task is presented by a node. Also, the task times are considered as the node weights and the precedence constraints are the arcs of this graph. Also, the APM is an n×n matrix in which each entry is defined by a binary variable apmij which takes value 1 if task i be the predecessor of task j and takes 0 otherwise.

A precedence graph with n = 11 tasks and its corresponding APM is created by us and is shown in Fig. (2). The task times are between 1 and 9 (time units). For example, processing task 7 requires its direct predecessors (tasks 1 and 5) and its indirect predecessor (task 3) to be completed. On the other hand, task 7 must be completed prior to its direct successors (task 8) and its indirect successors (tasks 9, 10, and 11).



$$
\text{APM} = \begin{array}{c|ccccccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
3 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

Fig 2. Precedence graph and its corresponding APM of 11 tasks, 5 stations, and a solution for SALBP2 (created by us).

The notations and other nomenclature of the SALBP2 are represented in Table I. Based on this notations, the load of stations 1 and 4 is $S_1 = \{1, 9\}$ and $S_4 = \{5, 6, 7\}$ with the station times $t(S_1) = t_1 + t_9 = 5 + 8 = 13$, and $t(S_4) = t_5 + t_6 + t_7 = 2 + 7 + 1 = 10$. The optimal number of stations is $c^* = \lceil t_{sum}/m \rceil = \lceil 61/8 \rceil = 8$ and the cycle time for the represented solution in Fig. (2) is $c = \max(t(S_1), t(S_2), t(S_3), t(S_4), t(S_5)) = \max(13, 16, 13, 10, 9) = 16$. Also the idle time of stations 1 and 4 is $I_1 = c - t(S_1) = 16 - 13 = 3$ and $I_4 = c - t(S_4) = 16 - 10 = 6$.

Table I. Notations And Symbols Of The Salbp2.

N	Number of assembly tasks; index j = 1,…, n
tj	The time of task j
Pj (Pj*)	Set of direct (all) predecessors of task j
Fj (Fj*)	Set of direct (all) successors of task j
Sk	The load of station k (i.e., the set of tasks assigned to station k)
m	Number of assembly workstations;
t (Sk)	The time of station k; $t(S_k) = \sum_{j\epsilon S_k} t_j$, k = 1 , ... , m
c	Cycle time; $c = \max(t(S_k))$
c*	Optimal cycle time; $c^* = \lceil t_{sum}/m \rceil$, $t_{sum} = \sum_{j=1}^{n} t_j$
Ik	Idle time of station k; Ik = c – t(Sk)

B. SALBP2 encoding

In this article a solution s to the SALBP2 is represented by a row vector of pairs (π_i, w_i), in which $\pi_i \in \{1, 2, …, n\}$ is the i–th assembled task and $w_i \in \{1, 2, …, m\}$ is the workstation in which task π_i is assembled. Fig. (3) shows the priority list L_1 = [1, 3, 2, 5, 4, 7, 6, 8, 9, 10, 11] and the encoding {(1, 1), (3, 2), (2, 3), (5, 4), (4, 5), (7, 4), (6, 4), (8, 3), (9, 1), (10, 2), (11, 2)} of the solution in Fig. (2), which indicates that part π_1 (= 1) must be assembled at station 1, then part π_2 (= 3) must be assembled at station 2, and so on.

The total number of encodings for feasible and infeasible solutions for the SALB2 problem equals $n! \times (m)^n$. Because the number of sequences of n tasks is n! and the number of possible workstations to assemble the mentioned task is m. It is necessary to explain that two or more seemingly different encodings may all be related to the same solution and therefore the size of the whole search space of SALBP2 problem is smaller than $n! \times (m)^n$.

L$_1$	1	3	2	5	4	7	6	8	9	10	11											
, *Sol*$_1$	1	1	3	2	2	3	5	4	4	5	7	4	6	4	8	3	9	1	10	2	11	2

Fig 3. The priority list and encoding of the solution in Fig. (2).

C. First Solution Generation

In order to generate the starting solution in population U, two sets of Heuristic approaches are used with a probability of 0.5. These two sets are constructed as follows:

Approach 1: In this approach, a priority list of n tasks is generated based on the randomly determined sequence of tasks. Then in each iteration, the task in the first position of the priority list (e.g. task j) is assigned to the station with the minimum station time. In the case of a tie, the station with a smaller number is selected. For example, applying this approach to the randomly determined priority list L_1 = [3, 9, 1, 4, 11, 5, 7, 6, 10, 8, 2] for the presented assembly line in Fig. (2) leads to the solution ⟨(3, 1), (9, 2), (1, 3), (4, 4), (11, 5), (5, 1), (7, 3), (6, 3), (10, 1), (8, 2), (2, 5)⟩ as shown in Table II. It should be noted that the created solutions by this approach are usually infeasible because in generating a solution from a specific list, the precedence relationships between tasks do not take into account.

Table II. Approach 1 For Generating A Starting Solution From The Randomly Determined Priority List L1 = [3, 9, 1, 4, 11, 5, 7, 6, 10, 8, 2] For The Presented Assembly Line In Fig. (2).

Number of stations: m = 5 ***priority list: L1 = [3, 9, 1, 4, 11, 5, 7, 6, 10, 8, 2]***									
Iteration	***Candidate task***	***Station time***					***Candidate stations***	***Chosen Station***	
		1	***2***	***3***	***4***	***5***		***Number***	***Time***
1	3	0	0	0	0	0	{1, 2, 3, 4, 5}	1	t(s1) = 0 + t3 = 5
2	9	5	0	0	0	0	{2, 3, 4, 5}	2	t(s2) = 0 + t9 = 8
3	1	5	8	0	0	0	{3, 4, 5}	3	t(s3) = 0 + t1 = 5
4	4	5	8	5	0	0	{4, 5}	4	t(s4) = 0 + t4 = 9
5	11	5	8	5	9	0	{5}	5	t(s5) = 0 + t11 = 8
6	5	5	8	5	9	8	{1, 3}	1	t(s1) = 5 + t5 = 7
7	7	7	8	5	9	8	{3}	3	t(s3) = 5 + t7 = 6
8	6	7	8	6	9	8	{3}	3	t(s3) = 6 + t6 = 13
9	10	7	8	13	9	8	{1}	1	t(s1) = 7 + t10 = 10
10	8	10	8	13	9	8	{2, 5}	2	t(s2) = 8 + t8 = 17
11	2	10	17	13	9	8	{5}	5	t(s5) = 8 + t2 = 12

Approach 2: In this approach, a priority list of n tasks is generated based on the randomly determined sequence of tasks. Then the station determination process is done for the random priority list. In each iteration of the station determination process (e.g. iteration i), the task in the first position of the priority list (e.g. task $\pi_{i} =$ j) is assigned to the station with the minimum station time (e.g. w_i = arg{min (t(S_k))}) if all direct predecessors of task j (i.e., all tasks in the set P_j) have been assigned prior to this task. Otherwise, the next task in the list is selected. For example, applying this approach to the randomly determined priority list L_1 leads to the solution ⟨(3, 1), (1, 2), (5, 3), (7, 4), (8, 5), (2, 4), (4, 3), (6, 1), (9, 2), (10, 4), (11, 4)⟩ as shown in Table III.

D. Operators for neighborhood Generation

In total 500 solutions are created using Approaches 1 and 2 as the initial population U for conducting the fitness landscape analysis. Then, a hill-climbing is performed on each initial solution to reach a local minimum. Four different operators are applied to the task list of the first solution to generate the neighboring task list. Then the station determination process is done for the newly generated task list. These neighborhood generation operators are as below:

Table III. Approach 2 For Generating A Starting Solution In U From The Randomly Determined Priority List L1 For The Presented Assembly Line In Fig. (2).

Number of stations: m = 5

priority list: L1 = [3, 9, 1, 4, 11, 5, 7, 6, 10, 8, 2]

Iteration	*Candidate task*	*Is candidate task assignable?*	*Station time*					*Candidate stations*	*Chosen Station*	
			1	*2*	*3*	*4*	*5*		*Number*	*Time*
1	3	Yes	0	0	0	0	0	{1, 2, 3, 4, 5}	1	t(s1) = 0 + t3 = 5
2	9	No						-		
	1	Yes	5	0	0	0	0	{2, 3, 4, 5}	2	t(s2) = 0 + t1 = 5
3	9	No						-		
	4	No						-		
	11	No						-		
	5	Yes	5	5	0	0	0	{3, 4, 5}	3	t(s3) = 0 + t5 = 2
4	9	No						-		
	4	No						-		
	11	No						-		
	7	Yes	5	5	2	0	0	{4, 5}	4	t(s4) = 0 + t7 = 1
5	9	No						-		
	4	No						-		
	11	No						-		
	6	No						-		
	10	No						-		
	8	Yes	5	5	2	1	0	{5}	5	t(s5) = 0 + t8 = 9
6	9	No						-		
	4	No						-		
	11	No						-		
	6	No						-		
	10	No						-		
	2	Yes	5	5	2	1	9	{4}	4	t(s4) = 1 + t2 = 5
7	9	No						-		
	4	Yes	5	5	2	5	9	{3}	3	t(s3) = 2 + t4 = 11
8	9	No						-		
	11	No						-		
	6	Yes	5	5	11	5	9	{1, 2, 4}	1	t(s1) = 5 + t6 = 12
9	9	Yes	12	5	11	5	9	{2, 4}	2	t(s2) = 5 + t9 = 13
10	11	No						-		
	10	Yes	12	13	11	5	9	{4}	4	t(s4) = 5 + t10 = 8
11	11	Yes	12	13	11	8	9	{4}	4	t(s4) = 8 + t11 = 16

(1) Exchange Operator: In this operator, two tasks of the list are selected randomly and their positions are exchanged. For example, assuming that the first solution of the SALBP2 is Sol_1 = ⟨(3, 1), (1, 2), (5, 3), (7, 4), (8, 5), (2, 4), (4, 3), (6, 1), (9, 2), (10, 4), (11, 4)⟩ by the corresponding task list L_1 = [3, 9, 1, 4, 11, 5, 7, 6, 10, 8, 2]. Two tasks 2 and 4 are randomly selected and the new list after applying the Exchange operator is L_2 = [3, 9, 1, 2, 11, 5, 7, 6, 10, 8, 4]. Doing the station determination process to list L_2 leads to the neighboring solution Sol_2 = ⟨(3, 1), (1, 2), (2, 3), (5, 4), (7, 5), (8, 5), (4, 4), (6, 3), (9, 1), (10, 2), (11, 2)⟩, which is shown in Fig. (4). The neighborhood's size of this operator is C(n , 2) $= n(n-1)/2$.

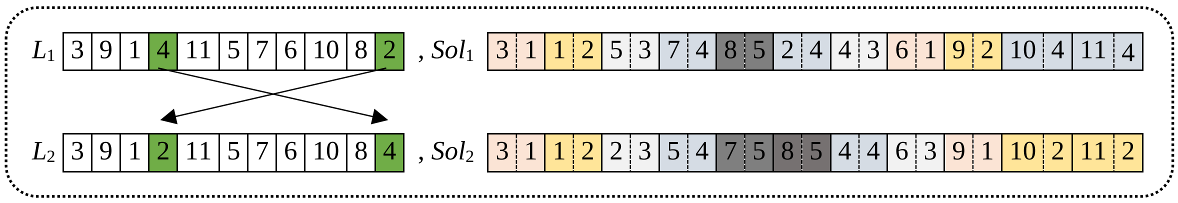

L_1 3 9 1 4 11 5 7 6 10 8 2 , Sol_1 3 1 1 2 5 3 7 4 8 5 2 4 4 3 6 1 9 2 10 4 11 4
L_2 3 9 1 2 11 5 7 6 10 8 4 , Sol_2 3 1 1 2 2 3 5 4 7 5 8 5 4 4 6 3 9 1 10 2 11 2

Fig 4. The Exchange Neighborhood Generation Operator.

(2) Insertion Operator: In this operator, a task in the list is randomly selected and relocated to a different position in the list. Finally, all tasks after the new position shift one position to the right. For example, for the first solution Sol_1 = ⟨(3, 1), (1, 2), (5, 3), (7, 4), (8, 5), (2, 4), (4, 3), (6, 1), (9, 2), (10, 4), (11, 4)⟩ by the corresponding task list L_1 = [3, 9, 1, 4, 11, 5, 7, 6, 10, 8, 2], the task 3 and the new position 6 are randomly selected and the new list after applying the Insertion operator is L_2 = [9, 1, 4, 11, 5, 3, 7, 6, 10, 8, 2]. Doing the station determination process to list L_2 leads to the neighboring solution Sol_2 = ⟨(1, 1), (3, 2), (5, 3), (7, 4), (8, 5), (2, 4), (4, 3), (6, 1), (9, 2), (10, 4), (11, 4)⟩)⟩, which is shown in Fig. (5). The size of the neighborhood for this operator is $n \times (n-1)$ since n tasks can be selected, and $(n-1)$ different positions can be selected as the relocation point.

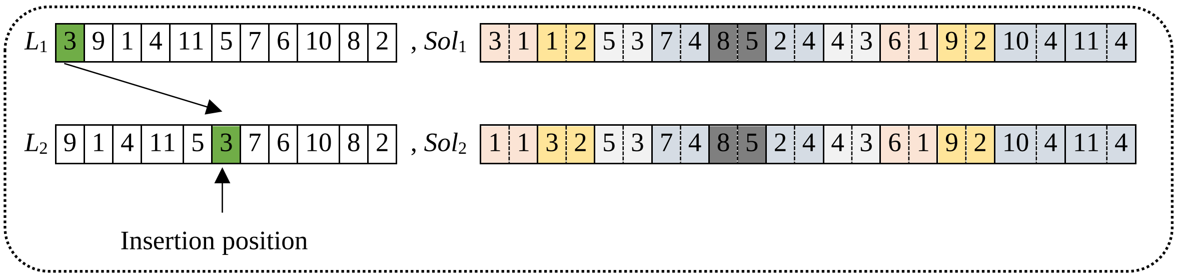

L_1 3 9 1 4 11 5 7 6 10 8 2 , Sol_1 3 1 1 2 5 3 7 4 8 5 2 4 4 3 6 1 9 2 10 4 11 4
L_2 9 1 4 11 5 3 7 6 10 8 2 , Sol_2 1 1 3 2 5 3 7 4 8 5 2 4 4 3 6 1 9 2 10 4 11 4
Insertion position

Fig 5. The Insertion Neighborhood Generation Operator.

(3) Inversion Operator: This operator randomly selects two tasks in the list and reverses the position of tasks between them in the list. For example, for the first solution Sol1 by the corresponding task list L1, two tasks 5 and 2 are randomly selected and the new list after applying the Inversion operator is L2 = [3, 9, 1, 4, 11, 2, 8, 10, 6, 7, 5]. Doing the station determination process to list L2 leads to the neighboring solution Sol2 = ⟨(3, 1), (1, 2), (2, 3), (4, 4), (6, 5), (5, 3), (7, 1), (8, 2), (9, 1), (10, 3), (11, 5)⟩)⟩, which is shown in Fig. (6). The size of the neighborhood for this operator is $\prod_{i=1}^{n-2}(n-i-1)$.

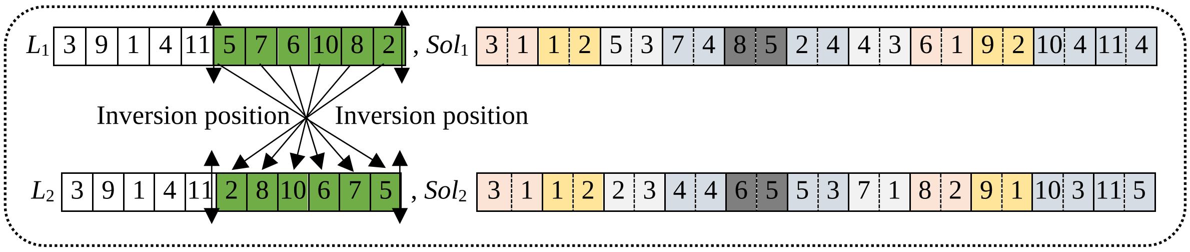
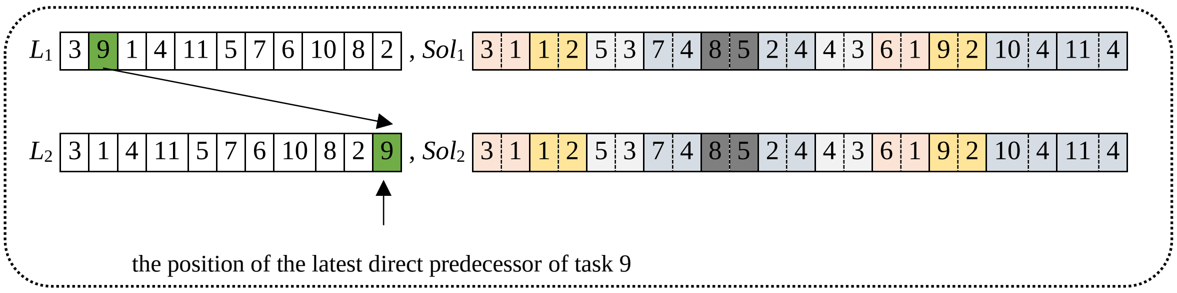

L_1 3 9 1 4 11 5 7 6 10 8 2 , Sol_1 3 1 1 2 5 3 7 4 8 5 2 4 4 3 6 1 9 2 10 4 11 4
Inversion position
Inversion position
L_2 3 9 1 4 11 2 8 10 6 7 5 , Sol_2 3 1 1 2 2 3 4 4 6 5 5 3 7 1 8 2 9 1 10 3 11 5

Fig 6. The Inversion Neighborhood Generation Operator.

(4) Cyclical shift Operator: In this operator, a task in the list is randomly selected and inserted into a different position after the position of its latest direct predecessor. For example, for the first solution Sol1 = ⟨(3, 1), (1, 2), (5, 3), (7, 4), (8, 5), (2, 4), (4, 3), (6, 1), (9, 2), (10, 4), (11, 4)⟩ by the corresponding task list L1 = [3, 9, 1, 4, 11, 5, 7, 6, 10, 8, 2], the task 9 is selected randomly and the new position of this task (e.g. position 11) is selected after the position of the latest direct predecessor (task 8). The new list after applying the Cyclical shift operator is L2 = [3, 1, 4, 11, 5, 7, 6, 10, 8, 2, 9]. Doing the station determination process to list L2 leads to the neighboring solution Sol2 = ⟨(3, 1), (1, 2), (5, 3), (7, 4), (8, 5), (2, 4), (4, 3), (6, 1), (9, 2), (10, 4), (11, 4)⟩ which is similar to the first solution, that is shown in Fig. (7). The size of the neighborhood for this operator is n × (n − 1), since n tasks can be selected, and at the best case (n − 1) positions (the positions after the latest direct predecessor) can be selected as the relocation position. In the best case, the selected task belongs to the tasks without any predecessors (i.e., elementary tasks).



L1 3 9 1 4 11 5 7 6 10 8 2 , Sol1 3 1 1 2 5 3 7 4 8 5 2 4 4 3 6 1 9 2 10 4 11 4
L2 3 1 4 11 5 7 6 10 8 2 9 , Sol2 3 1 1 2 5 3 7 4 8 5 2 4 4 3 6 1 9 2 10 4 11 4
the position of the latest direct predecessor of task 9

Fig 7. The Cyclical Shift Neighborhood Generation Operator.

It should be noted that the neighborhood generation operators are sequentially used in order to avoid repeated neighboring solutions. Also, the connectivity property is an important trait for a neighborhood generating operator and guarantees that through successive application of the operators and from any given solution, a particular solution could be reached. It can be shown that the connectivity property for any two solutions is established by our used neighborhood generation operators. The number of neighbors on a list is:

$$|N(s)| = max\left\{ n(n-1)/2 \text{و} n \times (n-1) \text{ و } \prod_{i=1}^{n-2}(n-i-1) \text{ و} n \times (n-1) \right\} \tag{1}$$

The generated assembly lines for a SALB2 problem must be feasible and a good quality. In the following subsections, the quality and feasibility of a solution are described.

1) Feasibility of a Solution

A solution s = ⟨(π1, w1), (π2, w2), …, (πn, wn)⟩ to a SALB2 problem is feasible if the task πi (∀i = 2,…, n) does not belong to the set of all predecessors of any already–assembled task πj (∀j < i). Put more formally, a solution s is feasible if:

$$\sum_{i=2}^{n}\sum_{j=1}^{i-1} apm_{\pi_i \text{و} \pi_j} = 0 \tag{2}$$

Based on relation (2), in a feasible solution, none of the predecessors of a particular task are assembled before the mentioned task. Also, in this article, the Percent of UnSatisfied Precedence Constraints (*PUSPC*) is considered as below:

$$PUSPC = \frac{USPC}{\binom{n}{2}} = \frac{USPC}{\frac{n(n-1)}{2}} \tag{3}$$

$$USPC = \sum_{i=2}^{n}\sum_{j=1}^{i-1} apm_{\pi_i.\pi_j} \tag{4}$$

in which *USPC* represents the 'UnSatisfied Precedence Constraints' and equals to the number of times that the precedence constraints between tasks are violated in a given solution and one of the precedences of a particular task is assembled after that task. Also, $\binom{n}{2}$ in relation (3) is the total number of precedence constraints which must be checked in order to determine the feasibility of a solution to the SALBP2. Therefore, dividing the number of unsatisfied precedence constraints by the total number of precedence constraints between tasks represents the percent of unsatisfied precedence constraints (PUSPC). It should be noted that for a feasible solution, USPC = 0 and respectively PUSPC = 0.

2) Quality of a Solution

The objective function is to minimize the Smoothness Index (SI) and cycle time (c). The smoothnessIndex which is expressed mathematically in (5) represents the line efficiency. The smaller *SI* means a closer processing time among workstations.

$$SI = \sqrt{\frac{\sum_{k=1}^{m}\left(c-t(S_k)\right)^2}{m}} \tag{5}$$

$$c = \max_{k=1....m} t(S_k) \tag{6}$$

In some works on SALBP2, infeasible solutions which violate precedence relations are discarded. But in this article to contain the feasibility of solutions in the objective function, the total fitness function is expressed as a weighted combination of the above objectives and is defined as:

$$Minimize\ f(s) = (1 + W_1 * PUSPC)(W_2 * SI + W_3 * c). \tag{7}$$

The first term in (7) minimizes the total number of unsatisfied precedence constraints among all checked precedence constraints for solution s, and the second term minimizes the total weighted sum of smoothness index and cycle time in the assembly line. In order to guide the search toward feasible solutions, the values of the factors are set as W1 = 5, W2 = 1, and W3 = 2.

F. Landscape Properties

In order to describe the landscape properties of the SALBP2, two types of statistical measures are used and calculated for 44 test problems in this paper. These types are defined as below:

(1) Distribution measures present the spatial situation of local solutions in the objective space and over the problem search space. These measures contain:

- The scatter of a population of solutions is represented by the Distribution measure. a population *P* has an average distance as:

$$dmm(P) = \frac{\sum_{s\in P}\sum_{t\in P,t\neq s} dist(s,t)}{|P|\cdot(|P|-1)} \tag{8}$$

where *dist*(*s*, *t*) is the Hamming distance of two solutions and || computes the size of population *P*. Also, a population *P* has an average normalized distance as:

$$Dmm(P) = \frac{dmm(P)}{diam(P)} \tag{9}$$

where diam(P) = max dist(s, t); ∀s, t ∈ P represents the population's diameter. The number of different values of solutions s and t is used to calculate the distance dist(s, t). Also, the maximum distance between two solutions is diam(P) = 2n because the solution encoding has 2n cells. A small Dmm shows that solutions are concentrated at a small area.

The variation of the average difference between *Dmm* of population *U* and population *O* is as:

$$\Delta_{Dmm} = \frac{Dmm(U) - Dmm(O)}{Dmm(U)} \tag{10}$$

- Entropy measures the scatter of the solutions and is an indicator of the diversity of solutions in the landscape of the problem. A novel entropy formulation for the SALBP2 is introduced in this article as follows:

$$ent(P) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{d=1}^{m} N_{ijd}\cdot(|P| - N_{ijd})}{mn^2\cdot(|P|-1)} \tag{11}$$

where N_{ijd} represents the number of solutions in *P* in which task *i* is assembled in the assembly workstation *k* and is in the position *j* of the sequence, or mathematically, $\pi_j = i$ and $w_j = m$. When $N_{ijd} = |P|$ we have *ent*(*P*) = 0. On the other hand, when N_{ijd} =1 we have *ent*(*P*) = 1. The variation of entropy between populations *U* and *O* is defined as:

$$\Delta_{ent} = \frac{ent(U) - ent(O)}{ent(U)} \leq 1 \tag{12}$$

According to the measures of Δ_{Dmm} and Δ_{ent}, three main topologies of local optima there are as shown in Fig. (8). For a 'Uniform' landscape, when a local search is applied to an initial solution, a nearby local optimum would rapidly be found. Since P–metaheuristics cover various solution clusters that are scattered across the search space, they are better choices In a 'Multi–massif' landscape. Finally, in a 'One–Massif' (or 'Massif Central') landscape, S–metaheuristics are more suitable than P–metaheuristics as they would waste less time searching relatively sparse areas of the space.

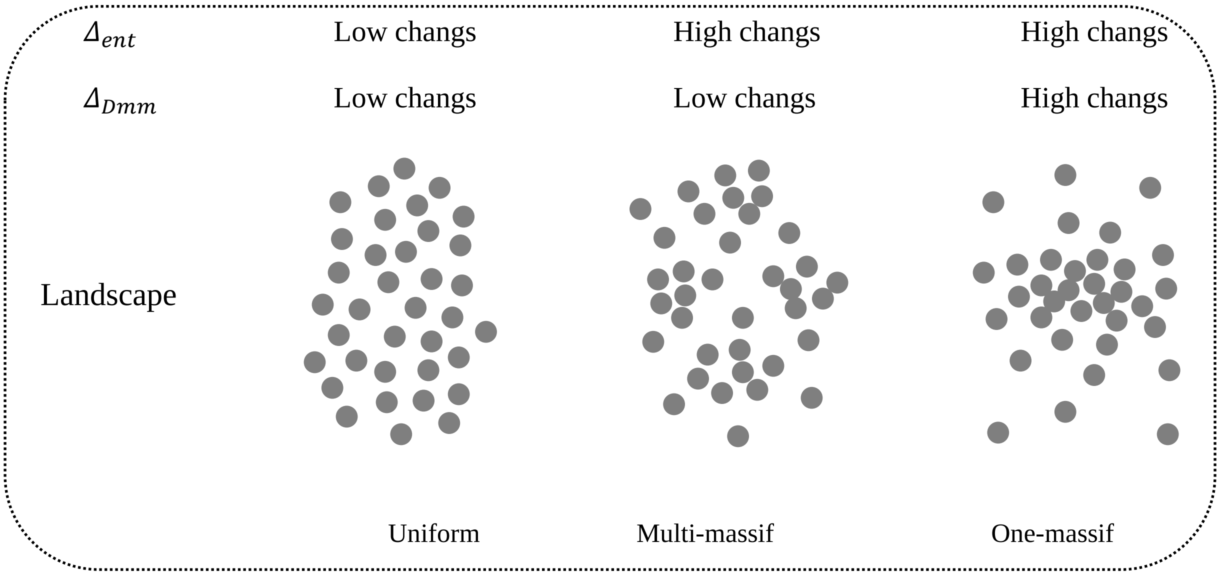


Fig 8. The Topology Of Local Optima In The Landscape According To The Entropy Variation And The Distribution Variation(Talbi, 2009).

- The difference between the fitness values of best and worst solutions in the population *P* is defined by the amplitude indicator *Amp*(*P*) as shown in (13). For a partly smooth search space, the value of *Amp*(*P*) is small.

$$Amp(P) = \frac{|P|\cdot(max_{s\in P} f(s) - min_{s\in P} f(s))}{\sum_{s\in P} f(s)} \geq 0 \tag{13}$$

The variation of amplitude between populations *U* and *O* is defined as:

$$\Delta_{Amp} = \frac{Amp(U) - Amp(O)}{Amp(U)} \leq 1 \tag{14}$$

Also 'Gap' of the population *O* of locally optimal solutions is the average distance between this population and the best-found solution (s^*) as defined below:

$$Gap(O) = \frac{\sum_{s \in O}(f(s) - f(s*))}{|O| \cdot f(s*)} \tag{15}$$

For a problem which is easy to solve, the Gap of the mentioned problem is small.

(2) Correlation measures analyze the correlation between the relative distance of the solutions and their fitness functions. The result of this analysis guides us to measure the ruggedness of the search space. These measures contain:

- The ruggedness of the landscape is defined by the Average length of the walks of a population *P*. The length of the walks between a final locally optimum solution and an initial random solution is long in a smooth landscape. On the other hand, in an unsmooth landscape the length of walks is shorter. For the population *O*, the average length of the walks is defined as:

$$Lmm(O) = \frac{\sum_{s \in O} l(s)}{|O|} \tag{16}$$

where $l(s)$ is the number of times the neighborhood operators are applied to an initial solution to reach to the final local optimal solution *s*.

- Autocorrelation function $\rho(d_H)$ investigates the effect of the distance between solutions on their fitness variation. For two neighboring solutions with different fitness values, we have $|\rho(1)| \approx 0$, and so the landscape is unsmooth. For *n* pairs of solutions with distance d_H, the autocorrelation function is as:

$$-1 \leq \rho(d_H) = \frac{\sum_{s,t \in S \times S, dist(s,t) = d_H}(f(s) - \bar{f})(f(t) - \bar{f})}{n.\sigma_f^2} \leq 1 \tag{17}$$

- Fitness Distance Correlation (FDC) analysis provides information about the correlation of the distance of a solution to the best–known or globally optimal solution and its fitness. At first, a population of *n* solutions are generated randomly and then the set of fitness function values $F = \{f_1, f_2, \ldots, f_n\}$ and the associated set of distances to the best–known solution $D = \{d_{H1}, d_{H2}, \ldots, d_{Hn}\}$ are computed. Afterward, the coefficient *r* (i.e., the fitness distance correlation) is computed as:

$$-1 \leq r = \frac{\frac{1}{n}\sum_{i=1}^{n}(f_i - \bar{f})(d_{Hi} - \overline{d_H})}{\sigma_f \cdot \sigma_{d_H}} \leq 1 \tag{18}$$

where σ_{dH} and σ_f are the standard deviation of the distance and fitness measures. For an 'misleading' problem we have $r \approx -1$. On the contrary, for an easy-to-solve problem, we have $r \approx 1$. The hardest situation arises when we have $|r| \approx 0$. For more information about the material in this subsection, one can refer to (Ghandi and Masehian, 2015).

G. Results of Landscape Analysis

Since the structure of a problem instance affects the fitness landscape of that problem, we implemented and used 14 different SALB2 test problems and 44 scenarios of the general SALB2 problem with various stations that are usually studied for SALBP2. For each scenario, the hill-climbing approach was applied 500 times to reach locally optimal solutions (in total, 44 * 500 = 22000 runs). The results of calculating the statistical measures for each scenario are reported in Table IV. The average values are shown below the values of that test problem in Table IV. Also, the average values of the used statistical measures in all scenarios are provided in the last row of this table. The critical analyses on the SALBP2 landscape are provided in the following subsections.

1) Distribution Measures

The small values of Δ_{ent} = 0.1547 and Δ_{Dmm} = 0.0918 imply that the locally optima are uniformly scattered over the search space. Also, the large value of Δ_{Amp} (= 0.8146) shows that the quality of locally optimal solutions is much better than initial solutions. Also the landscape is nearly plain. Also, the relatively small average amount of gap between the global solutions and the obtained local optimal solutions (nearly 5.95%) indicates that the neighborhood generation operators are suitable operators for SALBP2.

2) Correlation Measures

The relatively small amount of $Lmm(O)$ for many test problems of the SALBP2 (= 6.45 on average) indicates that this problem has a rugged landscape and therefore, a local optimum is encountered within a few search iterations. However, as Table IV indicates, the value of $Lmm(O)$ depends on the test problem and the considered m and c. In the studied scenarios, the 'Jaeschke test problem' has shorter walks compared to the other test problems which indicate that this test problem has a more rugged landscape than the others. Also, we calculate the autocorrelation measure for three Hamming distances of 2, 4, and 6. The results implied that on average, $\rho(2)$, $\rho(4)$, and $\rho(6)$ are almost equal. This confirms that the problem's landscape is rugged because the variation of fitness between 2–neighbors, 4–neighbors, and 6–neighbors are on average equal to the change between any two solutions. Based on the amount of the correlation coefficient r = 0.7839, the problem is easy to solve because the correlation between the distance of the solutions to the best-known solution and their fitness is strong.

Table IV. The Statistical Measures For 44 Salbp2 Instances.

Test Problem	*m*	*Distribution measures*							*Correlation measures*				
		Dmm(O)	*ΔDmm*	*ent(O)*	*Δent*	*Amp(O)*	*ΔAmp*	*Gap(O)*	*Lmm(O)*	*ρ(2)*	*ρ(4)*	*ρ(6)*	*r*
Bowman	3	0.1793	0.4612	2.8785	0.5675	0.1219	0.9385	0.0103	3.4240	0.5000	-0.3227	-0.1667	0.7958
	4	0.2382	0.3038	3.3966	0.4855	0.2927	0.8662	0.0279	4.5380	-0.1667	-0.3963	0.4820	0.8167
	5	0.2804	0.2127	4.0287	0.4013	0.3592	0.8384	0.0385	5.1240	-0.5000	0.3602	-0.3141	0.8391
Average		0.2326	0.3259	3.4346	0.4848	0.2579	0.8810	0.0256	4.3620	-0.0556	-0.1196	0.0004	0.8172
Buxey	3	0.3807	0.0460	0.5450	0.0551	0.1410	0.8961	0.0262	6.5260	0.5000	0.4962	0.5000	0.6893
	7	0.4263	0.0186	0.5644	0.0353	0.2908	0.7989	0.0516	7.9060	0.4993	0.4994	-0.3236	0.8828
	8	0.428	0.0172	0.5641	0.036	0.3428	0.7652	0.0699	7.6380	0.4999	-0.1354	0.3388	0.8840
	9	0.4313	0.0272	0.5643	0.0387	0.3721	0.7245	0.0733	7.4700	0.5000	-0.4618	0.5000	0.8891
Average		0.4166	0.0273	0.5595	0.0413	0.2867	0.7962	0.0553	7.3850	0.4998	0.0996	0.2538	0.8363
Gunther	3	0.3957	0.0212	0.3788	0.049	0.1596	0.8872	0.0570	7.5560	0.5000	0.4232	0.5000	0.5473
	6	0.4224	0.0272	0.3856	0.0378	0.3073	0.7852	0.0761	8.6280	0.5000	-0.4398	0.5000	0.8032
	7	0.4270	0.0300	0.3856	0.0399	0.3739	0.7656	0.0835	8.5320	0.5000	-0.0538	0.4995	0.8275
	8	0.4308	0.0247	0.3852	0.0403	0.4014	0.7201	0.0894	8.7060	0.5000	0.4986	-0.4998	0.8733
Average		0.4189	0.0258	0.3838	0.0417	0.3105	0.7895	0.0765	8.3555	0.5000	0.1070	0.2499	0.7628
Hahn	3	0.3576	0.143	0.1555	0.0913	0.0994	0.9376	0.0105	8.2940	0.5000	0.4954	0.5000	0.8149
	5	0.3878	0.0881	0.1621	0.0661	0.2964	0.8282	0.0154	7.7540	0.5000	0.4996	0.4279	0.9039
	6	0.3932	0.0896	0.1638	0.0588	0.3582	0.777	0.0095	7.2640	0.4986	0.4929	0.4959	0.9171
Average		0.3795	0.0969	0.1605	0.0721	0.2495	0.8476	0.0118	7.7707	0.4995	0.4960	0.4746	0.8786
Jackson	3	0.3407	0.0528	3.1302	0.1718	0.3142	0.8562	0.0066	3.3080	0.5000	0.5000	-0.1286	0.7065
	4	0.3539	0.0629	3.1586	0.1762	0.1631	0.9082	0.0228	4.8200	0.4800	-0.1667	-0.2331	0.7823
	5	0.3682	0.0376	3.1497	0.1852	0.4202	0.7839	0.0591	5.4560	0.5000	-0.0942	0.4560	0.8261

Test Problem	***m***	***Distribution measures***							***Correlation measures***				
		Dmm(O)	***ΔDmm***	***ent(O)***	***Δent***	***Amp(O)***	***ΔAmp***	***Gap(O)***	***Lmm(O)***	***ρ(2)***	***ρ(4)***	***ρ(6)***	***r***
Average		0.3543	0.0511	3.1362	0.1777	0.2992	0.8494	0.0295	4.5280	0.4933	0.0797	0.0314	0.7716
Jaeschke	3	0.2584	0.2069	3.2035	0.3919	0.4938	0.7943	0.0538	3.6540	0.4962	0.5000	-0.2457	0.8174
	4	0.2316	0.3427	2.6251	0.5075	0.5470	0.7619	0.003	3.4680	0.5000	-0.1450	-0.1461	0.8193
	5	0.2638	0.27	2.9314	0.4524	0.5848	0.7465	0.0480	4.2740	-0.1667	-0.1737	-0.4451	0.8715
	6	0.2875	0.1873	3.0493	0.4229	0.3452	0.8548	0.0002	2.5200	-0.1667	-0.1667	-0.0256	0.8913
Average		0.2603	0.2517	2.9523	0.4437	0.4927	0.7894	0.0263	3.4790	0.1657	0.0037	-0.2081	0.8499
Kilbridge	3	0.4002	0.0253	0.2348	0.0316	0.0519	0.957	0.0136	6.3220	0.4800	0.5000	0.5000	0.7078
	4	0.4088	0.0488	0.2374	0.0237	0.1764	0.8595	0.0381	8.0060	0.5000	0.5000	0.4296	0.5263
	5	0.4208	0.0266	0.2384	0.0223	0.2928	0.7713	0.0635	8.2460	0.5000	0.5000	0.3389	0.7503
	6	0.428	0.0274	0.2389	0.0221	0.3885	0.7099	0.1089	8.7740	0.4999	0.4388	0.4984	0.7262
	7	0.433	0.0263	0.2393	0.0214	0.5012	0.6658	0.1471	9.3600	0.5000	0.5000	0.5000	0.7740
Average		0.4182	0.0308	0.2377	0.0242	0.2822	0.7927	0.0742	8.1416	0.4959	0.4878	0.4534	0.6969
Lutz1	3	0.3528	0.1041	0.4248	0.0961	0.3139	0.8073	0.0150	7.8540	0.5000	0.4935	0.3944	0.7561
	4	0.3609	0.1006	0.4328	0.0809	0.1437	0.9365	0.0227	8.1800	0.5000	0.4991	-0.3491	0.8401
	8	0.3862	0.0745	0.4384	0.068	0.229	0.8748	0.0949	8.5360	0.4800	0.4994	0.4850	0.9087
Average		0.3666	0.0931	0.432	0.0816	0.6566	0.8728	0.0442	8.1900	0.4933	0.4973	0.1768	0.8349
Mansoor	3	0.3389	0.0518	3.3796	0.1431	0.3627	0.8264	0.0771	4.8820	0.4800	0.4900	-0.1667	0.7002
	4	0.3470	0.0870	3.3792	0.1275	0.4581	0.7429	0.0843	6.4820	0.5000	-0.1667	0.0079	0.7642
	5	0.3199	0.1714	3.0866	0.2068	0.3964	0.7778	0.0020	4.3380	0.5000	0.4800	0.5000	0.7733
Average		0.3353	0.1034	3.2818	0.1491	0.4057	0.7824	0.0545	5.2340	0.4933	0.2678	0.1085	0.7459
Mertens	3	0.3136	0.0597	6.3179	0.2914	0.4967	0.7530	0.0870	5.3780	0.4978	0.5000	-0.0746	0.7464
	4	0.3132	0.1452	6.2220	0.3241	0.7894	0.6667	0.0301	4.6800	-0.2012	0.4998	-0.5000	0.7673
	5	0.3288	0.0738	6.2193	0.3132	0.4506	0.8028	0.0990	5.0900	0.5000	0.2984	-0.4800	0.8107
Average		0.3185	0.0829	6.2198	0.3095	0.5789	0.7408	0.0720	5.0493	0.2655	0.4327	-0.3515	0.7748
Mitchell	3	0.3503	0.0712	0.9258	0.1294	0.2775	0.8365	0.1627	6.1420	0.5000	0.4618	0.4141	0.7823
	4	0.3619	0.0862	0.9360	0.1241	0.3860	0.7704	0.1672	6.2680	-0.1827	-0.5000	-0.0183	0.6532
	5	0.3698	0.0772	0.9335	0.1227	0.4106	0.7642	0.1484	6.1400	0.5000	0.5000	0.3087	0.8468
Average		0.3607	0.0782	0.9318	0.1254	0.3580	0.7904	0.1494	6.1733	0.2724	0.1539	0.2348	0.7608
Roszieg	3	0.3631	0.0464	0.6637	0.1210	0.1900	0.8863	0.0288	5.1380	0.5000	-0.4600	0.3290	0.7549
	4	0.3751	0.0631	0.6545	0.1358	0.2466	0.8467	0.0433	5.5880	-0.4956	-0.1873	0.3308	0.8146
Average		0.3691	0.0548	0.6591	0.1284	0.2183	0.8665	0.0361	5.3630	0.0022	-0.3237	0.3299	0.7832
Sawyer	3	0.3920	0.0276	0.5243	0.0330	0.1470	0.9066	0.0417	7.8360	0.5000	0.4923	0.3196	0.6029
	7	0.4313	0.0178	0.5334	0.0277	0.3377	0.7749	0.1552	8.3340	0.5000	0.4996	0.5000	0.7899
	8	0.4333	0.0073	0.5328	0.0270	0.3646	0.7430	0.1688	8.1060	-0.0340	0.5000	0.4539	0.8558
Average		0.4189	0.0176	0.5302	0.0292	0.2731	0.8082	0.1219	8.0920	0.3220	0.4973	0.4245	0.7495
Warnecke	3	0.4092	0.0108	0.1407	0.0333	0.0925	0.9291	0.0147	7.1740	0.5000	0.4917	0.4335	0.6437
Average		0.4020	0.0108	0.1407	0.0333	0.0925	0.9291	0.0147	7.1740	0.5000	0.4917	0.4335	0.6437
Total		0.3618	0.0918	1.6502	0.1547	0.3233	0.8146	0.0595	6.4474	0.3636	0.2282	0.1866	0.7839

III. HYBRID ITERATIVE LOCAL SEARCH (HILS) FOR THE SALBP2

The analysis of the search space of the SALBP2 problem showed that the local optima are uniformly distributed in the search space and the landscape of this problem is a rugged plain. Therefore, a single solution based metaheuristic algorithm is suitable for this problem. Also due to the small average gap (about 5.95%) and the relatively large average correlation coefficient (= 0.7839), the problem is easy to solve and therefore a simple single solution based metaheuristic is suitable to solve this problem. The Iterative Local Search (ILS) algorithm which has been successfully applied to solve optimization problems like the constrained clustering problem (González-Almagro et al., 2020), the vehicle routing problem (Penna et al., 2016; Belver et al., 2017; Wang et al., 2020) and the job shop scheduling problem (Nagata and Ono, 2018) has such requirements. Based on the powerful exploration capabilities of the ILS and its fine results in those domains, we selected it and used a set of 15 'priority rule-based constructive procedures' with the same probability of selection in order to solve the SALBP2. This newly generated algorithm which is called Hybrid Iterative Local Search (HILS) uses one of the priority rule-based constructive procedures to generate the initial solution and then uses local search in each iteration for discovering a local optimum. Also in the iterative phase, the Inversion operator is used to perturb the last local optimum and then a new local search is conducted from this perturbed solution. Thus, applying the constructive procedures leads to the exploitation of good solutions, and continually and adaptively applying perturbations by the Inversion operator leads to the exploration of new search areas.

In the following, the assumptions on the nature of the SALPB2 model and its solution approach are presented. The SALBP2 is one of the most–studied assembly line balancing problems in the literature. The aim of the SALBP2 is to minimize cycle time (c) due to a predefined number of stations (m). The following main characteristics are considered for this problem:

• One homogeneous product is Mass– produced;

• The production process of the product is known;

• The number of workstations (m) is fixed;

• The assembly times of tasks are integral and deterministic;

• The only assignment restrictions are the precedence constraints;

• The line layout is serial;

• The machines and workers of all stations are equal;

The main procedure and the Flowchart of the HILS method are presented in Fig. (9) and (10).

A. The HILS subroutines

The HILS algorithm requires two subroutines:

(1) Initial_Solution() to generate a starting point (sequence) for the search. We use a set of 15 'priority rule-based constructive procedures with the same probability of selection for constructing the initial solution for the HILS algorithm as shown in Table V. In all procedures, the task times or the precedence relations between tasks or a combination of them have been used for computing the priority values for different tasks. Then, a priority list is created based on the sorted priority values of tasks. Finally, the station determination process is done for the generated priority list. By using this process, a 'more feasible' initial solution is created.

(2) Local_Search() to improve the running solution by exploiting a part of the solution's neighborhood. A hill-climbing approach is utilized to an initial solution s_0 to move towards a local optimum s^* in the proposed HILS algorithm. In each iteration of the hill-climbing approach for obtaining a new solution, a neighborhood operator (Exchange, Insertion, or Cyclical shift operators with equal probability of selection) is applied to the running solution s. The neighboring solution is to replace the running solution in the case of improvement. If after E_{max} = 20 attempts no improvement was applied, the

running solution is considered as a local optimum. In this situation, the running optimum s^* is perturbed by applying the Inversion operator in order to find a neighboring solution. In the next iteration of the local search approach, this perturbed local optimum is considered the starting point. The algorithm terminates when the optimal solution to the test problem is found or the number of iterations reaches the *Maxiter*. Fig. (11) shows the pseudocode of the executed local search approach (Algorithm 2).

```
Algorithm 1 (Main). Hybrid Iterative_Local_Search ($APM, n, ATT, c^*, m, E_{max}, maxiter$)
Inputs:   $APM$ = The Assembly Precedence Matrix
          $n$ = Number of assembly tasks
          $ATT$ = The Assembly Task Times
          $c^*$ = Optimal Cycle Time of the assembly line
          $m$ = number of workstations
          $E_{max}$ = Maximum number of attempts within a local search iteration
          $maxiter$ = The maximum allowed number of iterations of the HILS algorithm
Output:   $s_{best}$ = Best solution found
--------------------------------------------------------------------------------
1.  $iter \leftarrow 0$               // Initializes counter iter for enumerating the number of elapsed iterations
2.  $s_0 \leftarrow$ Initial_Solution ($APM, n, ATT, m$) // Generates the initial solution
3.  $s_{best} \leftarrow s_0$                       // Records the best founded solution
4.  $f_{best} \leftarrow f(s_0)$                    // Records the best objective value obtained so far
5.  $s^* \leftarrow$ Local_Search ($s_0, E_{max}, , APM, ATT, m$) // Applies a hill-climbing local search
6.  $s_{lo} \leftarrow s^*$                         // Records the last founded local optimum
7.  While (($iter < maxiter$) and ($cbest > c^*$))
8.          $s_p \leftarrow$ Inversion_Operator($s^*$)      // Perturbs $s^*$ by applying Inversion Operator
9.          $iter = iter + 1$
10.         $s^* \leftarrow$ Local_Search($s_p, E_{max}$)     // Applies a local search on the perturbed solution $s_p$
11.         $s_{lo} \leftarrow s^*$                   // Updates the last local optimum
12.         If $f(s^*) < f_{best}$ Then
13.           $s_{best} \leftarrow s^*$               // Records the best founded solution
14.           $f_{best} \leftarrow f(s^*)$            // Calculated in (7)
15.           $cbest \leftarrow c$                  // Records the cycle time of the best found solution
16.         End
17. End
```

Fig 9. The Procedure Of The Hils Algorithm For The Salbp2.

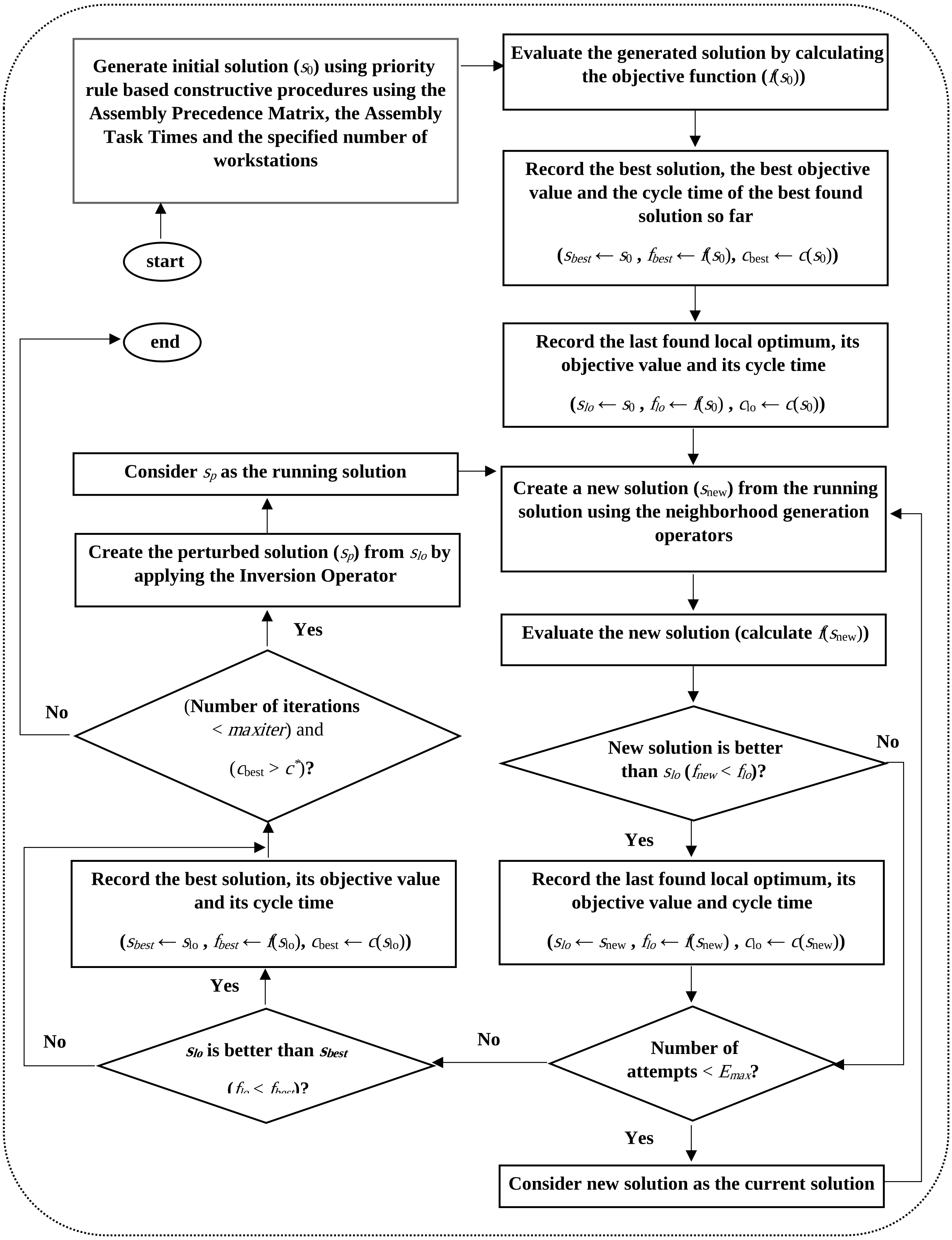

start
Generate initial solution (s_0) using priority rule based constructive procedures using the Assembly Precedence Matrix, the Assembly Task Times and the specified number of workstations
Evaluate the generated solution by calculating the objective function ($f(s_0)$)
Record the best solution, the best objective value and the cycle time of the best found solution so far
($s_{best} \leftarrow s_0$, $f_{best} \leftarrow f(s_0)$, $c_{best} \leftarrow c(s_0)$)
Record the last found local optimum, its objective value and its cycle time
($s_{lo} \leftarrow s_0$, $f_{lo} \leftarrow f(s_0)$, $c_{lo} \leftarrow c(s_0)$)
Create a new solution (s_{new}) from the running solution using the neighborhood generation operators
Evaluate the new solution (calculate $f(s_{new})$)
New solution is better than s_{lo} ($f_{new} < f_{lo}$)?
No
Yes
Record the last found local optimum, its objective value and cycle time
($s_{lo} \leftarrow s_{new}$, $f_{lo} \leftarrow f(s_{new})$, $c_{lo} \leftarrow c(s_{new})$)
Number of attempts < E_{max}?
Yes
Consider new solution as the current solution
No
s_{lo} is better than s_{best} ($f_{lo} < f_{best}$)?
Yes
No
Record the best solution, its objective value and its cycle time
($s_{best} \leftarrow s_{lo}$, $f_{best} \leftarrow f(s_{lo})$, $c_{best} \leftarrow c(s_{lo})$)
(Number of iterations < $maxiter$) and ($c_{best} > c^*$)?
Yes
No
Create the perturbed solution (s_p) from s_{lo} by applying the Inversion Operator
Consider s_p as the running solution
end

Fig 10. The Flowchart Of The Proposed HILS Algorithm For The SALBP2.

Table V. Priority Values For Generating The Initial Solution Of The Proposed Hils Algorithm.

Priority value :Definition / Ascending (A) or Descending (D)	*Generated priority list and its corresponding solution for presented assembly in Figure 2*
j : Task number / A	L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] Sol = ⟨(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 5), (7, 2), (8, 1), (9, 2), (10, 3), (11, 3)⟩
tj : Processing time / D	L = [4, 8, 9, 11, 6, 1, 3, 2, 10, 5, 7] Sol = ⟨(1, 1), (3, 2), (2, 3), (4, 4), (6, 5), (5, 3), (7, 1), (8, 2), (9, 1), (10, 3), (11, 5)⟩
tj : Processing time / A	L = [7, 5, 10, 2, 3, 1, 6, 11, 9, 8, 4] Sol = ⟨(3, 1), (5, 2), (1, 3), (7, 4), (2, 5), (4, 4), (6, 2), (8, 5), (9, 1), (10, 3), (11, 3)⟩
|Fj*| : Number of all followers / D	L = [1, 3, 2, 5, 4, 7, 6, 8, 9, 10, 11] Sol = ⟨(1, 1), (3, 2), (2, 3), (5, 4), (4, 5), (7, 4), (6, 4), (8, 3), (9, 1), (10, 2), (11, 2)⟩
|Fj| : Number of immediate followers / D	L = [1, 8, 2, 3, 4, 5, 6, 7, 9, 10, 11] Sol = ⟨(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 5), (7, 2), (8, 1), (9, 2), (10, 3), (11, 3)⟩
|Pj| : Number of immediate predecessors / D	L = [7, 9, 10, 2, 4, 5, 6, 8, 11, 1, 3] Sol = ⟨(1, 1), (3, 2), (2, 3), (4, 4), (5, 5), (7, 5), (6, 5), (8, 4), (9, 1), (10, 2), (11, 2)⟩
pwj = tj + $\sum_{h\epsilon F_j^*} t_h$: Positional weight / D	L = [1, 2, 3, 4, 5, 7, 8, 6, 9, 10, 11] Sol = ⟨(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (7, 5), (8, 5), (6, 2), (9, 1), (10, 3), (11, 3)⟩
pwj* = tj + $\sum_{h\epsilon F_j^*} pw_h^*$: Cumulated positional weight / D	L = [1, 3, 2, 5, 4, 7, 8, 6, 9, 10, 11] Sol = ⟨(1, 1), (3, 2), (2, 3), (5, 4), (4, 5), (7, 4), (8, 4), (6, 3), (9, 1), (10, 2), (11, 2)⟩
apwj = pwj*/|Fj*| : Average positional weight / D	L = [11, 1, 3, 2, 5, 4, 7, 8, 6, 9, 10] Sol = ⟨(1, 1), (3, 2), (2, 3), (5, 4), (4, 5), (7, 4), (8, 4), (6, 3), (9, 1), (10, 2), (11, 2)⟩
ubj = n + 1 $-\lceil pw_j/c \rceil$: Upper bound / A	L = [1, 2, 3, 4, 5, 7, 8, 6, 9, 10, 11] Sol = ⟨(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (7, 5), (8, 5), (6, 2), (9, 1), (10, 3), (11, 3)⟩
lbj = $\left\lceil \left(t_j + \sum_{h\epsilon P_j^*} t_h\right)/c \right\rceil$: Lower bound / A	L = [1, 2, 3, 5, 7, 4, 6, 8, 9, 10, 11] Sol = ⟨(1, 1), (2, 2), (3, 3), (5, 4), (7, 5), (4, 5), (6, 4), (8, 2), (9, 1), (10, 3), (11, 3)⟩
ub_nfj = ubj/|Fj*| : Upper bound / the number of all followers / A	L = [1, 3, 2, 5, 4, 7, 8, 6, 9, 10, 11] Sol = ⟨(1, 1), (3, 2), (2, 3), (5, 4), (4, 5), (7, 4), (8, 4), (6, 3), (9, 1), (10, 2), (11, 2)⟩
t_ubj = tj / ubj : Processing time divided by the upper bound / D	L = [4, 8, 9, 1, 6, 11, 3, 2, 10, 5, 7] Sol = ⟨(1, 1), (3, 2), (2, 3), (4, 4), (6, 5), (5, 3), (7, 1), (8, 2), (9, 1), (10, 3), (11, 5)⟩
slkj = ubj - lbj : Slack / A	L = [9, 10, 11, 1, 8, 2, 4, 6, 3, 5, 7] Sol = ⟨(1, 1), (2, 2), (4, 3), (6, 4), (3, 5), (5, 2), (7, 1), (8, 5), (9, 1), (10, 2), (11, 4)⟩
nf_slkj = |Fj*|/slkj : Number of all followers / the task slack / A	L = [11, 10, 6, 7, 8, 4, 9, 5, 2, 3, 1] Sol = ⟨(3, 1), (2, 2), (5, 3), (7, 4), (8, 5), (2, 4), (4, 3), (6, 1), (9, 2), (10, 4), (11, 4)⟩

```
Algorithm 2: Local_Search($s, E_{max}, APM, ATT, m$)
Output:   $s^*$ = A local optimum solution
1.  $ni = 0$                         //Initialize counter $ni$ of enumerating non-improving iterations
2.  While $ni < E_{max}$
3.      Generating a random $s' \in N(s)$   // Generate a neighbor by randomly applying a neighborhood operator
4.      If $f(s') < f(s)$ Then      // Calculated in (7)
5.          $s \leftarrow s'$                // Replace $s$ with the better neighbor $s'$
6.          $ni \leftarrow 0$                // Reset $ni$
7.      Else
8.          $ni \leftarrow ni + 1$
9.      End
10. End
11. $s^* \leftarrow s'$                       // Record $s^*$ as the local optimum
```

Fig 11. The Applied Local Search Procedure Executed In Each Iteration Of The Main HILS Algorithm.

B. Adjusting the parameters of the HILS algorithm

Since the output of metaheuristic algorithms is strongly dependent on their input values, in this paper the Taguchi method is used to adjust the parameters of the HILS algorithm. Five adjustable parameters in this algorithm are the number of iterations of the HILS algorithm (*Maxiter*), the number of non-improving iterations with no improvement of the hill-climbing (E_{max}), the probability of selecting the Exchange operator (P_E), the Probability of selecting the Cyclical Shift operator (P_{CS}) and the probability of selecting the Insertion neighborhood generation operator (P_{Is}). It should be noted that since the total probability of selecting each of the three Exchange, Cyclical Shift, or Insertion operators for neighborhood generation is equal to 1, so by determining the optimal value of two of these parameters, the optimal value for the other parameter is determined automatically. Therefore, among these three parameters, only two parameters P_E and P_{CS} are considered and as a result, the total number of adjustable parameters is reduced from five to four. For each of these four parameters, 3 levels of different values were defined based on preliminary experiments and the proposed ILS in (González-Almagro et al., 2020; Penna et al., 2016) (Table VI).

Table VI. The Adjustable Parameters Of The Proposed Hils Algorithm And Their Levels.

Parameter	*Description*	*Levels*
Maxiter	Number of iterations of the HILS algorithm	5 × n
		8 × n
		10 × n
Emax	Number of iterations with no improvement of the hill climbing	20
		25
		30
PE	Probability of selecting the Exchange neighborhood generation operator	0.22
		0.33
		0.44
PCS	Probability of selecting the Cyclical Shift neighborhood generation operator	0.22
		0.33
		0.44

Therefore, the number of created states is 3^4 or 81 states. For finding the best values of these parameters, these states have been reduced to 9 states as shown in Table VII by the Taguchi method and after executing these proposed states, the objective function values (relation (7)) were given to the Minitab software.

Table VII. The Best-Suggested States By The Taguchi Method Among 81 States.

Parameter	***State***								
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
Maxiter	5 × n	5 × n	5 × n	8 × n	8 × n	8 × n	10 × n	10 × n	10 × n
Emax	20	25	30	20	25	30	20	25	30
PE	0.22	0.33	0.44	0.33	0.44	0.22	0.44	0.22	0.33
PCS	0.22	0.33	0.44	0.44	0.22	0.33	0.33	0.44	0.22

Finally, after analyzing this data based on the signal-to-noise ratio, the best state among 81 states was announced by this software as shown in Fig. (12). In the analysis of the signal-to-noise ratio, the best and strongest conditions are determined using the change in results. This ratio shows the scatter around a defined value. The higher amount of this ratio reveals the lower amount of the dispersion and, therefore the more importance of the effect of the relevant parameter. Based on the main effects plot in Fig. (12), the best parameters values of the proposed HILS algorithm are *Maxiter* = 10 × *n*, E_{max} = 20, P_E = 0.33 and P_{CS} = 0.33 respectively. Also, since $P_E + P_{CS} + P_{Is} = 1$, according to the best values of P_E and P_{CS}, the best value of P_{Is} is equal to 0.33.

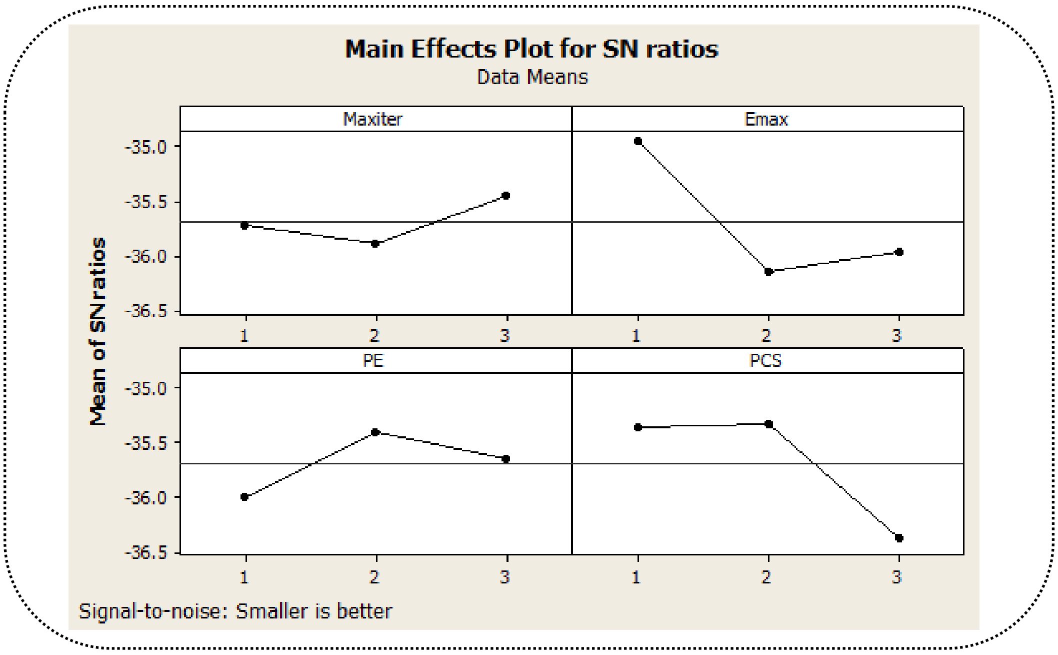


Fig 12. The Main Effects Plot For The Signal To Noise Ratio Provided By The Taguchi Method.

In order to perform the presented HILS algorithm, in the worst case, it must be run for *maxiter* iterations. Also, in each iteration of this algorithm, the local search method is applied and the worst-case number of attempts within a local search iteration equals to E_{max}. Therefore, in the worst case, the time complexity of the presented HILS algorithm is O (*maxiter* * E_{max}). Of course, whenever the optimal solution to the test problem is found, the algorithm is stopped and the optimal solution is reported.

IV. EXPERIMENTAL RESULTS

The implementing results of the HILS algorithm for SALBP2 are presented in this section. The program was coded in Matlab™ and run on an Intel™ Core-i7 1.8 GHz CPU with 8 GB of RAM. The performance of the HILS algorithm was compared to the performance of several existing population-based algorithms in the SALBP2 literature which have the same situation as the considered problem in this paper. The compared population-based algorithms are:

(1) MODE1: A Multi-Objective Differential Evolution (DE) algorithm with fixed weights in the objective function proposed in (Nearchou, 2008),

(2) MODE2: A Multi-Objective DE algorithm with random weights in the objective function (Nearchou, 2008),

(3) MODE3: A Multi-Objective DE algorithm with adaptive weights in the objective function (Nearchou, 2008).

(4) MOGA1: A Pareto-niched Genetic Algorithm (GA) used in (Bautista and Pereira, 2007) for solving Multi-Objective SALBP2.

(5) MOGA2: A Pareto weight-sum GA developed to address Multi-Objective Flow Shop Scheduling Problem (FSSP) and is extended and applied on SALBP2 in (Bautista and Pereira, 2007).

(6) SPT-SWAP: A variable neighborhood strategy adaptive search (VaNSAS) method with the shortest processing time-swap (SPT-SWAP) neighborhood strategy (Pitakaso et al., 2021). This method is extended and applied to SALBP2 in this article.

The following performance metrics are typically used in the SALBP2 literature and were considered for comparing the abovementioned algorithms:

1- Minimizing the cycle time of the best-found solution attained by an algorithm; ***c*** in Eq. (4).

2- Minimizing the mean relative deviation in percent (mrd%) from the best-known cycle times; ***mrd%*** = $100 \times ((c - c^*) / c^*)$, in witch c^* is the existing best-known cycle time, and c the cycle time of the best solution generated by a specific algorithm.

3- Minimizing the average Smoothness Index of the found optimal solution by an algorithm; ***SI*** in Eq. (3).

4- Minimizing the average Balance Delay time of the founded optimal solution by a considered algorithm; $BD = \sum_{K=1}^{m}(c - t(S_k))/m.$

For comparisons, we solved 5 difficult test problems of the SALBP2 based on the precedence graph by the HILS method and compared the results with the obtained results by the MODE1, MODE2, MODE3 (Nearchou, 2008), MOGA1, MOGA2 (Bautista and Pereira, 2007) and SPT-SWAP (Pitakaso et al., 2021) population based metaheuristics, as showed in Tables VIII and IX. Because of the stochastic behavior of the methods, every test instance was solved by each method 10 times and the average solution quality was reported. On the other hand, each method was run over $(8 + 8 + 10 + 9 + 23) \times 10 = 580$ test instances. The results in Table VIII present the runs with the objective given by $F_1 = w_1 \times c + w_2 \times SI$, and the presented results in Table IX show the runs with the objective given by $F_2 = w_1 \times c + w_2 \times BD$. The proposed self-adapted method in (Nearchou, 2008) is used for estimation of the weights $w_1 = \lambda \times e^{-d} + \mu$ and $w_2 = 1 - w_1$ in which $\lambda = \mu = 0.5$ and $d = c - c^*$. The second and third columns of these Tables represent the number of the included tasks in the precedence graphs (i.e., n) and the number of test instances of the considered test problem (i.e., in#). The fourth column show two considered comparison metrics (i.e., mrd% and *SI* in Table VIII and mrd% and *BD* in Table IX). Also, the last seven columns contain the amount of these metrics for the best solution attained in 10 runs of every method for all instances of a specific test problem. Note that the best values of these performance metrics are shown in bold in Tables

VIII and IX.

Based on the results of Table VIII, the HILS method has the best *SI* metric in all test problems thanks to its neighborhood generation operators. Also, it reaches the best amount of the mrd% metric in all except the Sawyer test problem for which the SPT-SWAP approach has achieved a better value. It should be noted that the difference between the obtained values for this metric by these two methods for the Sawyer test problem is very small. Therefore, the HILS method performs better than all other methods based on both comparison metrics, either mrd% or *SI*.

Table VIII. Comparison Of The Mrd% And Si Metrics Of Mode1, Mode2, Mode3, Moga1, Moga2, And Hils Methods For 5 Difficult Test Problems Of The Salbp2.

Test problem	*n*	*in#*	*Metric*	*MODE1*	*MODE2*	*MODE3*	*MOGA1*	*MOGA2*	*SPT-SWAP*	*HILS*
Buxey	29	8	mrd%	0.266	3.969	0.266	0.266	2.831	0.266	0.266
			SI	6.211	9.580	7.252	6.276	8.513	6.012	5.987
Sawyer	30	8	mrd%	1.712	4.372	0.669	2.562	2.910	0.579	0.745
			SI	5.997	9.704	6.911	7.002	8.161	5.365	4.962
Gunther	35	10	mrd%	1.139	2.610	0.750	1.139	1.139	0.543	0.210
			SI	14.717	19.301	18.454	14.902	15.424	13.298	12.123
Kilbridge	45	9	mrd%	0.000	0.881	0.000	0.302	3.572	0.000	0.000
			SI	2.796	5.027	3.786	0.820	4.756	2.796	1.865
Tonge	70	23	mrd%	1.093	3.726	0.977	1.755	1.989	0.977	0.977
			SI	22.800	52.931	23.274	27.858	30.690	22.639	22.639

Table IX. Comparison Of The Mrd% And Bd Metrics Of Mode1, Mode2, Mode3, Moga1, Moga2, And Hils Methods For 5 Difficult Test Problems Of The Salbp2.

Test problem	*n*	*in#*	*Metric*	*MODE1*	*MODE2*	*MODE3*	*MOGA1*	*MOGA2*	*SPT-SWAP*	*HILS*
Buxey	29	8	mrd%	0.712	3.936	0.266	2.226	3.579	0.266	0.266
			BD	18.250	25.625	15.252	22.500	26.750	14.219	13.453
Sawyer	30	8	mrd%	1.922	4.372	1.562	4.272	3.313	1.43	1.54
			BD	15.750	27.625	18.125	20.125	24.000	14.97	13.212
Gunther	35	10	mrd%	0.750	1.778	0.250	7.659	2.082	0.237	0.220
			BD	40.600	52.700	44.400	83.500	49.400	40.600	40.123
Kilbridge	45	9	mrd%	0.340	0.981	0.161	6.851	0.860	0.128	0.128
			BD	9.778	13.333	8.778	47.000	12.667	7.876	7.876
Tonge	70	23	mrd%	1.213	3.919	1.109	2.050	3.217	0.987	0.987
			BD	87.609	183.930	87.700	117.261	137.435	87.493	87.001

Similar high performance for the HILS method is reported in Table IX. As specified in the Table, HILS obtained the best results based on the mrd% performance metric. The mean offset of the best generated solutions by this algorithm from optimum solutions is equal to 0.22% (BD≈40) for Gunther's problems, 0.27% (BD≈13) for Buxey's problems, 1.54% (BD≈13) for Sawyer's problems, etc. Again, the proposed HILS performs better than all the other methods by generating high quality solutions. For a better illustration of the generated results, Fig. (13) plots the mrd% and Smoothness Index *SI* of six algorithms for five test problems when F_1 is used as the objective function. Also, the mrd% and Balance Delay *BD* of six algorithms for five test problems when F_2 is used as the objective function are plotted in Fig. (14). These Figures show the outperformance of the HILS method over the others for all test problems and performance metrics. An important observation from these figures and tables is that considering *SI* as the second term in the objective function leads to generating better solutions than those obtained using *BD* as the second term.

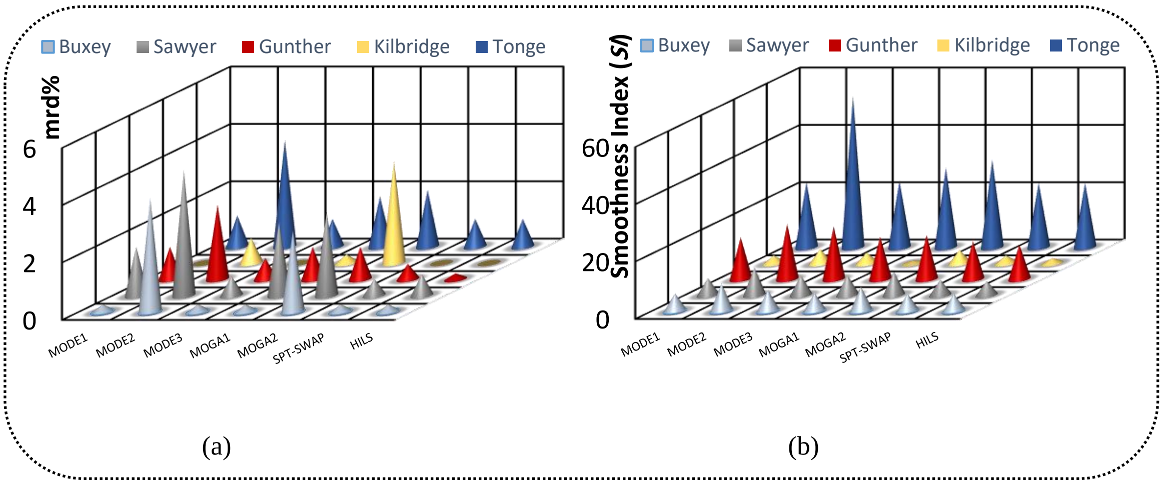


Fig 13. (A) Average Mrd% (In Percent), And (B) Average Smoothness Index (Si) Of Six Algorithms For Five Test Problems When F1 Is Used As The Objective Function.

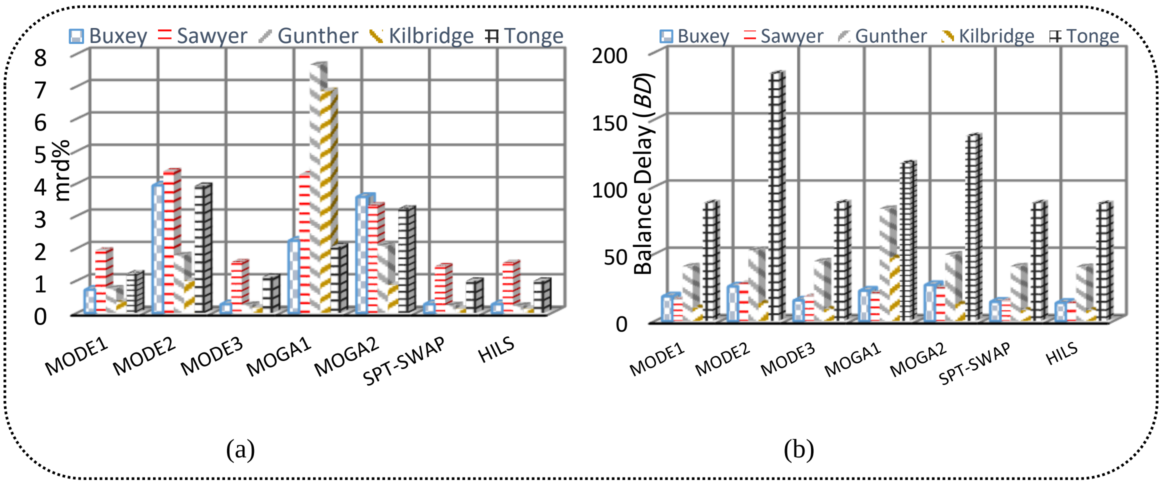


Fig 14. (A) Average Mrd% (In Percent), And (B) Average Balance Delay (BD) Of Six Algorithms For Five Test Problems When F2 Is Used As The Objective Function.

Then again, in order to evaluate the performance of the HILS with other single-solution based metaheuristics, we solved eight SALB2 test problems with the various number of workstations by our designed HILS, as well as the PNA-for (A forward version of the Petri Network Algorithm (PNA)), PNA-back (A backward version of the PNA algorithm) and PNA-bid (A bidirectional version of the PNA algorithm) methods all of them were developed in (Kilincci, 2010). The comparison results are presented in Table X. Altogether 21 test problems were solved. Here the objective function is minimizing the cycle time (c) and the stopping criterion of the algorithms is set to 100 iterations.

The number of tasks (i.e., n), the number of considered stations in the corresponding test problem (i.e., m), and the optimal cycle time for the specific test problem (i.e., c^*) are given in the second, third and fourth columns of these Tables. Columns 5 to 10 show the cycle time and percent excess cycle time by three versions of the PNA, and the last two columns display the results by the proposed HILS method. Note that in this Table, the best optimal values of the performance metrics are shown in bold and with green background. Also, the last row of this table includes the number of issues among the 22 considered test problems for which the corresponding algorithm has succeeded to reach the optimal assembly cycle time (i.e., $c = c^*$) for them. The following observations are made from Table X:

- Although 9, 12, and 12 of 22 problems are optimally solved by PNA-for, PNA-back, and PNA-bid, the proposed HILS algorithm finds optimal solutions for 18 out of 22 problems respectively.

- The average of percent excess cycle time for the proposed approach (i.e., 0.59%) is better than the three versions of the PNA (i.e., 3.07%.1.91%, and 2.38%). This is due to the heuristic procedures used to create the initial solution for the HILS algorithm.
- HILS outperforms or equally fits other algorithms in all test problems.
- The proposed HILS algorithm has not achieved the optimal cycle time for only 4 out of 22 test problems. Also, in these four problems, it has the same or better performance than other algorithms according to the generated cycle time and mrd% metrics.

Table X. Comparison Results Of Hils Method And Three Versions Of Petri Network Algorithm For Salbp2.

Test problem	*N*	*M*	*c**	*PNA-for*		*PNA-back*		*PNA-bid*		*HILS*	
				c	*mrd%*	*c*	*mrd%*	*c*	*mrd%*	*c*	*mrd%*
Merten	7	5	7	7	0.00	7.0	0.00	7	0.00	7	0.00
		3	10	11	10.00	10	0.00	10	0.00	10	0.00
		2	15	15	0.00	15	0.00	15	0.00	15	0.00
Jaeschke	9	7	7	7	0.00	7	0.00	7	0.00	7	0.00
		4	10	10	0.00	10	0.00	10	0.00	10	0.00
		3	13	14	7.69	13	0.00	13	0.00	13	0.00
Jackson	11	5	10	11	10.00	11	10.0	11	10.0	11	10.0
		4	12	13	8.33	12	0.00	13	8.33	12	0.00
		3	16	16	0.00	16	0.00	16	0.00	16	0.00
Mitchell	21	8	14	15	7.14	14	0.00	14	0.00	14	0.00
		5	21	21	0.00	23	9.52	23	9.52	21	0.00
		3	35	35	0.00	36	2.86	36	2.86	35	0.00
Heskiaoff	28	5	205	208	1.46	205	0.00	205	0.00	205	0.00
		4	256	257	0.39	256	0.00	256	0.00	256	0.00
Sawyer	30	13	26	26	0.00	28	7.69	28	7.69	26	0.00
		8	41	45	9.76	41	0.00	41	0.00	41	0.00
		5	65	67	3.08	67	3.08	69	6.15	66	1.53
Kilbridge	45	10	56	57	1.79	57	1.79	57	1.79	56	0.00
		6	92	93	1.09	94	2.17	94	2.17	92	0.00
		3	184	184	0.00	186	1.09	184	0	184	0.00
Tonge	70	11	320	333	4.06	327	2.19	327	2.19	322	0.63
		8	439	451	2.73	446	1.59	446	1.59	443	0.91
Number of optimums				9		12		12		18	
Average mrd%				3.07		1.91		2.38		0.59	

For more comparisons, we used and solved the usual set of 302 benchmark instances from Armin Scholl's website, http://www.assembly-line-balancing.de. The benchmark set is composed of two subsets: Dataset1 is a set by 128 instances with the number of tasks between 29 to 111 based on 9 different precedence graphs and Dataset2 is a set by 174 instances with the number of tasks between 53 to 297 based on 8 different precedence graphs. The averages of instances of the same precedence graph and the averages of all instances of Dataset1 and Dataset2 for the HILS algorithm are compared with those obtained by the Iterative Beam Search (IBS) (Blum, 2011) and two versions of the Integer-coded Differential Evolution Algorithm (IDEA) (Zhang et al., 2016). Each version of this algorithm corresponds to one of the following mutation schemes:

1- DE/rand/2/bin in which the best vector is considered as the base vector. This version was referred to with the abbreviation IDEA_rd2.

2- self-adaptive double mutation in which mutation procedures were developed in order to increase the chance of finding better solutions. This version was referred to with the abbreviation IDEA_apt, respectively.

The comparison metric is the mrd%. Here the objective function is to minimize the cycle time (c) and the proposed HILS algorithm is terminated, if the existing optimum solution is generated or the maximum number of iterations (*Maxiter*) is reached. The comparison results are shown in Tables XI and XII.

Table XI. The Mrd% Of Hils And Ibs Algorithms For The 128 Instances Of Dataset1.

Graph		*Buxey*	*Sawyer*	*Lutz1*	*Gunther*	*Kilbridge*	*Tonge*	*Arcus1*	*Lutz2*	*Arcus2*	*Average*
Algorithm	IBS	0.0	0.0	0.0	0.0	0.0	0.0	0.0278	0.0	0.0066	0.0058
	IDEA_rd2	2.21	3.34	0.35	1.02	0.60	2.07	0.98	2.64	4.98	2.02
	IDEA_apt	3.07	3.52	0.71	1.27	0.60	1.78	0.78	1.99	4.64	2.04
	HILS	0.0	0.0	0.0	0.0	0.0	0.0067	0.0198	0.0	0.0034	0.0049

Table XII. The Mrd% Of Hils And Ibs Algorithms For The 174 Instances Of Dataset2.

Graph		*Hahn*	*Warnecke*	*Wee-Mag*	*Lutz3*	*Mukherje*	*Barthold*	*Barthol2*	*Scholl*	*Average*
Algorithm	IBS	0.0	0.0579	0.0	0.0	0.0	0.0	0.0	0.0028	0.0094
	IDEA_rd2	0.0	3.98	1.63	2.88	_	0.46	4.79	1.02	3.41
	IDEA_apt	0.0	3.51	1.39	1.57	_	0.26	4.79	9.24	2.97
	HILS	0.0	0.0215	0.0143	0.0	0.0	0.0	0.0	0.0129	0.0077

The following facts are revealed from the results in these Tables:

1- The HILS outperformed or works the same as the other methods in 8 out of 9 different precedence graphs of Dataset1 and 6 out of 8 different precedence graphs of Dataset2. In the other 3 graphs, the value obtained by HILS is slightly different from the best-obtained results by the other algorithms. The reliability of this algorithm is shown by this fact.

2- As shown in the last column of Tables XI and XII, the performance of the HILS algorithm is competitive with the compared algorithms in the mrd% for the 128 instances of Dataset1and the 174 instances of Dataset2.

The above results show that considering the cycle time alone as the objective function for the HILS algorithm leads to the achievement of the minimal cycle time for Buxey, Sawyer, Gunther, and Kilbridge test problems. While for the same test problem, considering $F_1 = w_1 \times c + w_2 \times SI$ or $F_2 = w_1 \times c + w_2 \times BD$ as the objective function does not lead to the optimal cycle time (as shown in Tables VIII and IX). This confirms that the intended objective function influences the shape of the search space of a considered test problem and thus the effectiveness and efficiency of the metaheuristic.

Also single-solution-based metaheuristics, i.e. the HILS and IBS algorithms, significantly work better than the population-based metaheuristics. Another important result from Tables VIII to X is that the single-solution-based PNA-for, PNA-back, PNA-bid, and HILS algorithms outperform the population-based MODE and MOGA algorithms based on the mrd% metric for the 'Sawyer', 'Kilbridge' and 'Tonge' test problems. These are in line with the results of the analysis of the landscape of the SALBP2.

Based on these results, the performance of HILS outperforms other considered algorithms for the studied problem. Especially the proposed HILS algorithm is able to find the optimal or near-optimal assembly line balances based on the considered objective functions and performance metrics for a large number of considered and investigated test problems.

The reason for this is the used effective methods to generate the neighboring solutions, initial solutions, and perturbed solutions for this algorithm.

Of course, it is necessary to mention that despite the good performance of the proposed HILS algorithm for SALBP2 problem, the main restriction of this algorithm is that the perturbation strength has to be sufficient to lead the trajectory to a different local optimum. A too weak perturbation makes a small change in the current local optimal solution and therefore is unable to take the algorithm out of the current local optimum. On the other hand, in the case of a too strong perturbation strategy, the algorithm becomes the random restart local search. A good way is to use an adaptive perturbation strategy in which the perturbation strength is determined based on the search information.

V. CONCLUSIONS AND FUTURE WORKS

The Assembly Line Balancing (ALB) problem is used for work assignments in order to let operate with the same average production time for each workstation. This balancing leads to the flexibility of the process flow system, eliminating delays or bottlenecks, or mistakes during production and producing products correctly. The benefits of Assembly Line Balancing in the real-world application are improving the process efficiency, increasing the production rate, reducing the total processing time, minimizing the ideal time, increasing the profits and decreasing the costs. The SALBP problem is one of the categories of the ALB problem that is appropriate for modeling single–sided assembly lines. SALBP2 is one of the classes of the SALBP that partition the assembly operations into a set of tasks and assign them to assembly workstations such that the assembly time of all workstations is approximately equal. This problem's aim is to minimize cycle time (c) given a defined number of stations (m). This problem has been widely used in all single–sided assembly lines including automotive and related industries (bus, truck, aircraft, heavy machinery, etc.). SALBP2 is appropriate for single–sided assembly lines in which a unique model of a single product is created. This problem considers the minimization of cycle time with a given number of workstations. As SALBP2 problem is proved to be NP-hard, several works in the literature have been dedicated to its solution. Nevertheless, the fitness landscape of this problem has not been analyzed previously. In this paper, several distributions and correlation measures have been used for analyzing the landscape of the SALBP2. This analysis helps to find the appropriate metaheuristic to address SALBP2 problem. Based on the results of the fitness landscape analysis, the SALBP2 problem's local optima are uniformly scattered in the search space and the landscape of this problem is rugged plain. As a result, it was concluded that for solving the SALBP2 a local search method with effective exploration and exploitation capabilities is suitable. Therefore the Iterated Local Search method that is an efficient single-solution-based metaheuristic was selected for this purpose and combined with a number of constructive procedures to construct a Hybrid Iterated Local Search (HILS) algorithm and efficiently solve the SALBP2.

The precedence graph and station determination process are used for generating feasible assembly line balances. Starting from initially generated sequences by a set of 15 'priority rule-based constructive procedures' with the same probability of selection for constructing the initial solution for the HILS algorithm, we purposely use the Inversion operator to perturb a local optimum solution. In order to strengthen the neighborhood searching ability, efficient Exchange, Insertion, and Cyclical shift neighborhood generation operators of local search are incorporated. The distinctive characteristic of the HILS algorithm is its ability for perturbing local optimal solutions. By studying 58 test problems (5 test problems with the various number of stations) from the literature, our proposed method outperforms the former proposed population-based MODE1, MODE2, MODE3, MOGA1, MOGA2 and SPT-SWAP algorithms. Also, the proposed approach outperforms the considered single-solution-based algorithms including the PNA-for, PNA- back, and PNA- bid based on the generated cycle time and mrd% metrics for 22 studied test problems (8 test problems with the various number of stations). For more comparisons, we used and solved the usual set of 302 benchmark instances and compared the averages over instances and Dataset1 and Dataset2 results of the HILS algorithm with those obtained by the IBS, IDEA_rd2, and IDEA_apt. The results show the performance of the HILS algorithm over these three compared algorithms in the mrd% from the best-known upper bounds. Computational experiments showed the outperformance of HILS over nearly all of the competing methods, thanks to the careful selection of the initial solution, neighborhood generation operators, Inversion operator and algorithm parameters. Therefore, the enhanced HILS algorithm leads to finding assembly lines balanced with higher quality. This can help to reduce the product's overall cost and the leading time.

Although a large number of metaheuristics are applied for solving the Simple Assembly Line Balancing Problems (including SALBP1, SALBP-E and SALBP-F), and Generalized Assembly Line Balancing Problems (including 2S–ALB, MALB and UALB), the landscape of these problems has not been investigated in any research yet. Therefore, future research should include the landscape analysis of these problems. Also the evaluation of other efficient single-solution-based metaheuristic algorithms against the proposed HILS algorithm for solving the SALBP2 is another potential future study that can be investigated. The algorithm could be extended to solve two-sided, U-type and mixed-model assembly lines in order to prove the performance of the proposed approach for other problems. In the future, we plan to extend our FLA and algorithm for the application to assembly line balancing problems that are closer to real industrial settings. Also, the quality of the proposed HILS algorithm could be improved by using an adaptive perturbation strategy in which the perturbation strength is determined based on the search information. Applying the introduced concept to real-world applications such as SALBP-2 problem with a limit on the number of types of machines, SALBP2 with setup times and resource-constrained SALBP2 is another suggestion for future research.

REFERENCES

Alakaş, H. M. (2021), "General resource-constrained assembly line balancing problem: conjunction normal form based constraint programming models," Soft Computing, 25 (8), 6101-6111.

Alavidoost, M., Babazadeh, H., and Sayyari, S. (2016), "An interactive fuzzy programming approach for bi-objective straight and U-shaped assembly line balancing problem," Applied Soft Computing, 40, 221-235.

Anderson, E. J., and Ferris, M. C. (1994), "Genetic algorithms for combinatorial optimization: the assemble line balancing problem," ORSA Journal on Computing, 6 (2), 161-173.

Battaïa, O., and Dolgui, A. (2013), "A taxonomy of line balancing problems and their solutionapproaches," International Journal of Production Economics, 142 (2), 259-277.

Bautista, J., and Pereira, J. (2007), "Ant algorithms for a time and space constrained assembly line balancing problem," European Journal of Operational Research, 177 (3), 2016-2032.

Beham, A., Pitzer, E., Wagner, S., and Affenzeller, M. 2017. Integrating Exploratory Landscape Analysis into Metaheuristic Algorithms. In International Conference on Computer Aided Systems Theory: Springer.

Belver, M., Gomez, A., Lopez, J., De la Fuente, D., and Ponte, B. 2017. Solving Vehicle Routing Problem with Multiple Trips using Iterative Local Search with Variable Neighborhood Search. In Proceedings on the International Conference on Artificial Intelligence (ICAI): The Steering Committee of The World Congress in Computer Science.

Blum, C. (2011), "Iterative beam search for simple assembly line balancing with a fixed number of work stations," SORT, 35 (2), 145-164.

Buyukozkan, K., Kucukkoc, I., Satoglu, S. I., and Zhang, D. Z. (2016), "Lexicographic bottleneck mixed-model assembly line balancing problem: artificial bee colony and tabu search approaches with optimised parameters," Expert Systems with Applications, 50, 151-166.

Capacho Betancourt, L. (2007), "ASALBP: the alternative subgraphs assembly line balancing problem. Formalization and resolution procedures," PHD thesis, Technical University of Catalonia.

Chantarasamai, K., and Lasunon, O.-U. (2021), "Modified Differential Evolution Algorithm for U-Shaped Assembly Line Balancing Type 2".

Chiang, W.-C. (1998), "The application of a tabu search metaheuristic to the assembly line balancing problem," Annals of Operations Research, 77, 209-227.

Eghtesadifard, M., Khalifeh, M., and Khorram, M. (2020), "A systematic review of research themes and hot topics in assembly line balancing through the web of science within 1990–2017," Computers & Industrial Engineering, 139, 106182.

Esmaeilbeigi, R., Naderi, B., and Charkhgard, P. (2015), "The type E simple assembly line balancing problem: A mixed integer linear programming formulation," Computers & Operations Research, 64, 168-177.

Ghandi, S., and Masehian, E. (2015), "A breakout local search (BLS) method for solving the assembly sequence planning problem," Engineering applications of artificial intelligence, 39, 245-266.

González-Almagro, G., Luengo, J., Cano, J.-R., and García, S. (2020), "DILS: constrained clustering through Dual Iterative Local Search," Computers & Operations Research, 104979.

Hackman, S. T., Magazine, M. J., and Wee, T. (1989), "Fast, effective algorithms for simple assembly line balancing problems," Operations Research, 37 (6), 916-924.

Heinrici, A. (1994), "A comparison between simulated annealing and tabu search with an example from the production planning," in Operations research proceedings 1993: Springer, pp. 498-503.

Jana, N. D., Sil, J., and Das, S. (2017), "Selection of appropriate metaheuristic algorithms for protein structure prediction in AB off-lattice model: a perspective from fitness landscape analysis," Information Sciences, 391, 28-64.

Jirasirilerd, G., Pitakaso, R., Sethanan, K., Kaewman, S., Sirirak, W., and Kosacka–Olejnik, M. (2020), "Simple Assembly Line Balancing Problem Type 2 By Variable Neighborhood Strategy Adaptive Search: A Case Study Garment Industry," Journal of Open Innovation: Technology, Market, and Complexity, 6 (1), 21.

Kilincci, O. (2010), "A Petri net-based heuristic for simple assembly line balancing problem of type 2," The International Journal of Advanced Manufacturing Technology, 46 (1-4), 329-338.

Klein, R., and Scholl, A. (1996), "Maximizing the production rate in simple assembly line balancing—a branch and bound procedure," European Journal of Operational Research, 91 (2), 367-385.

Kriengkorakot, N., and Pianthong, N. (2012), "The assembly line balancing problem: review articles," Engineering and Applied Science Research, 34 (2), 133-140.

Li, Z., Kucukkoc, I., and Tang, Q. (2021), "Enhanced branch-bound-remember and iterative beam search algorithms for type II assembly line balancing problem," Computers & Operations Research, 131, 105235.

Liu, X., Yang, X., and Lei, M. (2021), "Optimisation of mixed-model assembly line balancing problem under uncertain demand," Journal of Manufacturing Systems, 59, 214-227

Malan, K. M., and Engelbrecht, A. P. (2014), "Fitness landscape analysis for metaheuristic performance prediction," in Recent advances in the theory and application of fitness landscapes: Springer, pp. 103-132.

McNaughton, R. (1959), "Scheduling with deadlines and loss functions," Management Science, 6 (1), 1-12.

Merengo, C., Nava, F., and Pozzetti, A. (1999), "Balancing and sequencing manual mixed-model assembly lines," International Journal of Production Research, 37 (12), 2835-2860.

Nagata, Y., and Ono, I. (2018), "A guided local search with iterative ejections of bottleneck operations for the job shop scheduling problem," Computers & Operations Research, 90, 60-71.

Nearchou, A. C. (2008), "Multi-objective balancing of assembly lines by population heuristics," International Journal of Production Research, 46 (8), 2275-2297.

Nourmohammadi, A., Fathi, M., and Ng, A. H. 2019. Choosing efficient meta-heuristics to solve the assembly line balancing problem: A landscape analysis approach. In Procedia CIRP: Elsevier.

Penna, P. H. V., Afsar, H. M., Prins, C., and Prodhon, C. (2016), "A hybrid iterative local search algorithm for the electric fleet size and mix vehicle routing problem with time windows and recharging stations," IFAC-PapersOnLine, 49 (12), 955-960.

Pinarbasi, M., Alakas, H. M., and Yuzukirmizi, M. (2019), "A constraint programming approach to type-2 assembly line balancing problem with assignment restrictions," Assembly Automation.

Pitakaso, R., Sethanan, K., Jirasirilerd, G., and Golinska-Dawson, P. (2021), "A novel variable neighborhood strategy adaptive search for SALBP-2 problem with a limit on the number of machine's types," Annals of Operations Research, 1-25.

Rashid, M. F. F., Hutabarat, W., and Tiwari, A. (2012), "A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches," The International Journal of Advanced Manufacturing Technology, 59 (1-4), 335-349.

Razali, M. M., Kamarudin, N. H., Rashid, M. F. F. A., and Rose, A. N. M. (2019), "Recent trend in mixed-model assembly line balancing optimization using soft computing approaches," Engineering Computations.

Scholl, A., and Becker, C. (2006), "State-of-the-art exact and heuristic solution procedures for simple assembly line balancing," European Journal of Operational Research, 168 (3), 666-693.

Sepahi, A., and Naini, S. G. J. (2016), "Two-sided assembly line balancing problem with parallel performance capacity," Applied Mathematical Modelling, 40 (13-14), 6280-6292.

Sotskov, Y. N., Dolgui, A., Lai, T.-C., and Zatsiupa, A. (2015), "Enumerations and stability analysis of feasible and optimal line balances for simple assembly lines," Computers & Industrial Engineering, 90, 241-258.

Sungur, B., and Yavuz, Y. (2015), "Assembly line balancing with hierarchical worker assignment," Journal of Manufacturing Systems, 37, 290-298.

Talbi, E.-G. (2009), Metaheuristics: from design to implementation (Vol. 74): John Wiley & Sons.

Wang, M., Li, B., Zhang, G., and Yao, X. (2017), "Population evolvability: Dynamic fitness landscape analysis for population-based metaheuristic algorithms," IEEE Transactions on Evolutionary Computation, 22 (4), 550-563.

Wang, X., Shao, S., and Tang, J. (2020), "Iterative Local-Search Heuristic for Weighted Vehicle Routing Problem," IEEE Transactions on Intelligent Transportation Systems.

Watanabe, T., Hashimoto, Y., Nishikawa, I., and Tokumaru, H. (1995), "Line balancing using a genetic evolution model," Control Engineering Practice, 3 (1), 69-76.

Yadav, A., Kulhary, R., Nishad, R., and Agrawal, S. (2019), "Parallel two-sided assembly line balancing with tools and tasks sharing," Assembly Automation.

Zhang, H., Yan, Q., Liu, Y., and Jiang, Z. (2016), "An integer-coded differential evolution algorithm for simple assembly line balancing problem of type 2," Assembly Automation.

Zheng, Q., Li, M., Li, Y., and Tang, Q. (2013), "Station ant colony optimization for the type 2 assembly line balancing problem," The International Journal of Advanced Manufacturing Technology, 66 (9-12), 1859-1870.

Zohali, H., Naderi, B., and Roshanaei, V. (2021), "Solving the Type-2 Assembly Line Balancing with Setups Using Logic-Based Benders Decomposition," INFORMS Journal on Computing.