International Journal of Industrial Engineering Computations 14 (2023) 589–608

Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Fitness landscape analysis of the simple assembly line balancing problem type 1

Somayé Ghandi[a] and Ellips Masehian[b*]

[a]*Department of Industrial Engineering, Faculty of Engineering, University of Kashan, Kashan, Iran*
[b]*Industrial and Manufacturing Engineering Dept., California State Polytechnic University, Pomona, United States*

C H R O N I C L E

Article history:
Received March 24 2023
Received in Revised Format July 12 2023
Accepted September 12 2023
Available online September 12 2023

Keywords:
Simple Assembly Line Balancing Problem Type 1
Fitness Landscape Analysis
Distribution and Correlation Measures
Local Search

A B S T R A C T

As the simple assembly line balancing problem type 1 (SALBP1) has been proven to be NP-hard, heuristic and metaheuristic approaches are widely used for solving middle to large instances. Nevertheless, the characteristics (fitness landscape) of the problem's search space have not been studied so far and no rigorous justification for implementing various metaheuristic methods has been presented. Aiming to fill this gap in the literature, this study presents the first comprehensive and in-depth Fitness Landscape Analysis (FLA) study for SALBP1. The FLA was performed by generating a population of 1000 random solutions and improving them to local optimal solution, and then measuring various statistical indices such as average distance, gap, entropy, amplitude, length of the walk, autocorrelation, and fitness-distance among all solutions, to understand the complexity, structure, and topology of the solution space. We solved 83 benchmark problems with various cycle times taken from Scholl's dataset which required 83000 local searches from initial to optimal solutions. The analysis showed that locally optimal assembly line balances in SALBP1 are distributed nearly uniformly in the landscape of the problem, and the small average difference between the amplitudes of the initial and optimal solutions implies that the landscape was almost plain. In addition, the large average gap between local and global solutions showed that global optimum solutions in SALBP1 are difficult to find, but the problem can be effectively solved using a single-solution-based metaheuristic to near-optimality. In addition to the FLA, a new mathematical formulation for the entropy (diversity) of solutions in the search space for SALBP1 is also presented in this paper.

1. Introduction

The broader *assembly planning* (AP) problem has three main subproblems: *assembly sequence planning* (ASP), *assembly path planning* (APP), and *assembly line balancing* (ALB) (Somaye Ghandi & Ellips Masehian, 2015). This study deals with a special type of ALB with wide applications in the manufacturing industry. Suppose that n assembly operations (such as welding, riveting, and screwing) in a facility are partitioned into j elementary tasks, each taking t_j time to complete. The challenge in ALB is to assign these tasks to m assembly workstations such that all workstations have equal assembly times, and the precedence relations between the tasks are satisfied. The ALB problem is further classified into two NP-hard subproblems: Simple Assembly Line Balancing (SALB) and Generalized Assembly Line Balancing (GALB) (Scholl & Becker, 2006), as described below:

SALB: This category is suitable for modeling (single-sided) assembly lines that produce a unique model of a single product with deterministically-known input parameters (Capacho Betancourt, 2007). A single-sided assembly line consists of a

* Corresponding author Phone: +1 909-8692653
E-mail: masehian@cpp.edu (E. Masehian)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)

doi: 10.5267/j.ijiec.2023.9.005

sequence of *m* workstations usually connected by a conveyor belt, through which the product units flow. Each workstation performs a subset of the *n* operations necessary for manufacturing the products. Each product unit remains at each station for a fixed time called the 'cycle time', *c*. In these assembly lines, workstations are consecutively arranged in a straight line. Each product unit proceeds along this line and visits each workstation once (Gonçalves & De Almeida, 2002).

SALB problems have been categorized into some types:

i. *Type 1* (SALBP1) minimizes the number of workstations (*m*) for a given fixed cycle time (*c*).
ii. *Type 2* (SALBP2) minimizes the cycle time for a given number of workstations.
iii. *Type E* (Efficiency) (SALBP-E) maximizes the line efficiency (*E*) by minimizing *c* and *m* simultaneously and considering their interrelationships. Most research on SALBP focuses on SALBP1 and SALBP2, and few studies deal with the optimization of assembly line balancing efficiency or SALBP-E (Wei & Chao, 2011).
iv. *Type F* (feasibility) (SALBP-F) determines if a feasible line balance exists for a given combination of *m* and *c*.

GALB: This category of ALB problems is appropriate for balancing more complex assembly lines and has the following classifications (Capacho Betancourt, 2007):

i. *General two-sided assembly line balancing* (2S–ALB) deals with lines with pairs of operating workstations positioned opposite to each other and each workstation performing a different task (Yadav *et al.*, 2019).
ii. *Mixed-model assembly line balancing* (MALB) involves lines that assemble several models of a basic product in an intermixed sequence. Considering that tasks have different times for different models, the problem is to optimize the balance (load) of the stations or any cost-oriented objective function by determining the best cycle time and assignment of tasks to the workstations.
iii. *U–line assembly line balancing* (UALBP) deals with single-product assembly lines in which workstations are arranged on both sides of a U-shaped path. Thus, operators can work on either side of the path while simultaneously performing both early and late tasks of the assembly.
iv. The robotic assembly line balancing problem (RALBP) involves flexible assembly lines comprising human workers, robots, and equipment (Chutima, 2022; Li et al., 2018).

On the other hand, solution approaches to the SALBP1 are categorized into two classes:

1- *Exact solution approaches*, which can find the optimal solution accurately but are not efficient enough for NP-hard optimization problems and their execution time increases exponentially with the problem size. In fact, the first methods that were developed to solve the SALBP1 belonged to exact approaches. Exact approaches include lower bounds, dominance rules, reduction rules, dynamic programming procedures, branch-and-bound procedures (Dolgui & Gafarov, 2019; Vilà & Pereira, 2013), and mathematical programming (Pastor & Ferrer, 2009; Scholl & Becker, 2006). In branch-and-bound procedures, bounding strategies and preprocessing rules are usually applied to increase the efficiency (Vilà & Pereira, 2013). Also, Intelligent techniques are elaborated to avoid complete enumeration (Dolgui & Gafarov, 2019). In the existing mathematical programs, an initial pre-process usually is carried out to calculate the range of workstations to which a task *i* may be assigned, aiming to reduce the number of variables of task–workstation assignment (Pastor & Ferrer, 2009).
2- *Approximate solution approaches*, that are able to find good (near-optimal) solutions to NP-hard problems within short runtimes. Approximate approaches are divided into three categories: problem-specific heuristics, metaheuristics, and hyperheuristics.

Although problem-specific heuristic algorithms are developed based on the features and properties of the problem at hand, they have the shortcomings of getting stuck in local optima or converging prematurely to such points. The following works have utilized heuristic methods to solve the SALBP1: (Fathi et al., 2018; Hackman et al., 1989; Helgeson & Birnie, 1961; Kilincci, 2011; Pape, 2015; Scholl & Voß, 1997; Talbot, 1985). The success of heuristic algorithms strongly depends on the problem characteristics (Pape, 2015). Also, the effectiveness and efficiency of heuristic algorithms depend, among other factors, on the fitness function used. The fitness function has a major role since, in addition to being used to identify the best solution found, it is used to guide the search algorithm toward an area of the feasible region with promising, high-quality solutions. Therefore, the fitness function must be closely correlated with the objective of the original problem in order to avoid losing the best-found solution, due to miscalculation of the solution quality, and to avoid late convergence and consequently higher computational times. In addition, the computation of the fitness function must be easy and fast due to the iterative nature of heuristic methods (Fathi et al., 2018).

Metaheuristic algorithms are general search methods applicable to a wide range of problems and can find local optimal solutions through exploration and exploitation in the search space. These algorithms allow the generation of several solutions due to the incorporation of randomness in the procedure. On each iteration of these algorithms for solving the SALBP1, a task is chosen from a subset of the candidates using a probabilistic rule which may take into account a priority rule and information obtained by previous iterations (Bautista & Pereira, 2002). In the last two decades, numerous metaheuristic methods have been developed for solving the SALBP1, including constructive procedures (Ponnambalam *et al.*, 2000), genetic algorithms (GA) (Álvarez-Miranda et al., 2021), tabu search (TS) (Abdeljaouad & Klement, 2021; Pape, 2015), simulated annealing (SA) (Nagy *et al.*, 2020), ant colony optimization (ACO) (Bautista & Pereira, 2002; Bautista & Pereira, 2007; Zhong & Ai, 2017), artificial immune systems (AIS) (Zhang, 2018), discrete particle swarm optimization (PSO) (Dou *et al.*, 2017), heuristics based on slope

indices (Baskar & Xavior, 2020), the firing sequence backward algorithm (Kilincci, 2011), and variable-depth local search (Álvarez-Miranda *et al.*, 2022).

Hyperheuristic algorithms are high-level, automatic search methods that manage a set of low-level heuristic algorithms to solve complex computational problems. In most cases, by combining machine learning techniques, the process of selecting, combining, generating, or matching several simple heuristic algorithms is used to efficiently solve computational search problems. Unlike metaheuristic algorithms that are used to solve only one problem, hyperheuristics create a system to solve classes of different problems. The following works have utilized hyperheuristic methods to solve the SALBP1: (Gonçalves & De Almeida, 2002; Hu et al., 2023; Meng et al., 2021; Özbakır & Seçme, 2022; Seçme & Özbakır, 2019).

A survey of ASP- and ALB-related researches conducted between 2001 and 2011 that utilized soft computing approaches was presented by (Rashid *et al.*, 2012). Other comprehensive surveys of SALBP1 can be found in (Scholl & Becker, 2006), (Mohammed *et al.*, 2021), (Ravelo, 2022), and (Boysen *et al.*, 2021).

While reviewing the literature on SALBP1 methods, we noticed that although most approximate solution approaches implement either single-solution-based (S-) metaheuristics (such as SA) or population-based (P-) metaheuristics (such as ACO), they do not provide justifications and reasons for using their selected algorithm. However, the effectiveness and efficiency of a metaheuristic algorithm for an optimization problem strongly depend on the landscape of the problem, and depending on the shape of the landscape, specific types of search methods with certain intensification and diversification capabilities will be more effective. Therefore, before selecting and implementing a metaheuristic algorithm for a particular problem, it is important (and sometimes necessary) to analyze the search space of the problem and identify the distribution and magnitude of its peaks and valleys. This analysis, known as *Fitness Landscape Analysis* (FLA), examines the shape of the problem's search space using the distribution of local optima and their relationships and distances to each other in the search space and provides a good understanding of the structure of the solution spaces of the problem at hand.

To the best of our knowledge, so far there are only two studies that have performed FLA in the field of assembly planning: (Somayé Ghandi & Ellips Masehian, 2015) and (Nourmohammadi *et al.*, 2019). Both studies, however, have not addressed SALBP1 exactly or properly in sufficient breadth or depth; thus, it was necessary to fill this gap and present a comprehensive FLA for SALBP1, as we did in this paper, to provide a reference for researchers in the field.

Below, we highlight the differences between the present work and each of the works.

(1) The FLA in (Somayé Ghandi & Ellips Masehian, 2015) was performed for the ASP problem (sequencing the assembly of parts in a product) and not for the ALB problem.
(2) Although an FLA was done for the SALBP1 in (Nourmohammadi *et al.*, 2019), it had serious drawbacks and inaccuracies listed as follows: (*i*) the FLA was done only for one small problem instance with 7 tasks, which is neither a benchmark problem nor a practical case; (*ii*) only one operator type (i.e., swapping) was implemented for generating neighboring solutions throughout the local search; (*iii*) no solution representation was presented and the way the local search was applied to a given solution is unclear; (*iv*) the size of the studied population of solutions (only 500) was not sufficient to draw inclusive conclusions; (*v*) some important FLA measures like Gap, Length of walks, and Autocorrelation were not computed or analyzed; (*vi*) the function used for calculating the entropy was specific to the quadratic assignment problem (QAP) and not to the SALBP1, where in addition to the sequence of tasks, their assignment to workstations is also critical; and most importantly, (*vii*) their final conclusion about the shape of the landscape made based on the FLA results was incorrect (we will talk about those later in Sec. 5). In this paper, we avoided the abovementioned drawbacks in our FLA by conducting it for numerous benchmark problems (83 in total) with large solution populations of size 1000, and by implementing four different neighborhood operators. In addition, we proposed a novel entropy function designed exclusively for SALBP1 and reached a comprehensive conclusion regarding the structure of the landscape of SALBP1 based on a larger number of statistical indices.

The results of the present study can be used to determine the class of heuristic optimization algorithms that will be more effective in searching the solution space and finding near-optimal (if not optimal) solutions.

2. Problem definition and notations

To manufacture a product on an assembly line, the total amount of work must be partitioned into a set of n elementary operations called *tasks*, which should be completed on a number of loaded stations (m) given a fixed cycle time (c). Performing task j requires certain equipment of machines and/or skills of workers and takes time t_j. If the set of tasks T_k is allocated to workstation k, then the assembly time of workstation k is $tt_k = \sum t_j$, $\forall j \in T_k$. The goal of solving SALBP1 is to determine the minimum number of stations, considering the following assumptions:

- One homogeneous product is mass-produced,
- The production process is predetermined,

- The line is paced with a fixed cycle time c,
- Tasks are performed without preemption and their operation times t_j are deterministic,
- The only assignment restriction is the precedence constraints, which are either already known or are the output of an ASP subproblem solved prior to the ALB problem,
- The product of the assembly line was processed sequentially and consecutively from station 1 to *station m*.
- Assembly machines and workers in all stations are similar.

A solution s to SALBP1 can be represented by a row vector of ordered pairs (π_i, w_i), where $\pi_i \in \{1, 2, \ldots, n\}$ is the i-th assembled task, and $w_i \in \{1, 2, \ldots, m\}$ represents the number of workstations in which task π_i is assembled. Fig. 1 shows the graphical representation and encoding of an assembly line configuration in the form of $\langle(1, 1), (2, 1), \ldots, (6, 4)\rangle$, which indicates that part π_1 (= 1) must be assembled at station 1, so $T_1 = \{1\}$, then part π_2 (= 2) must be added to the subassembly at workstation 1 ($T_1 = \{1, 2\}$), and so on.

Tasks and their relations can be visualized using a precedence graph (as depicted in Fig. 1(a)) containing a node for each task, the time of each task, and arcs for designating precedence constraints. The *assembly precedence matrix* (APM) is an $n \times n$ matrix in which each entry is denoted by a binary variable apm_{ij} that takes the value of 1 if task i is the predecessor of task j and 0 otherwise. For the sample solution presented in Fig. 1(b), the number of workstations is $m = 4$, the cycle time $c = 10$ minutes, the total assembly time $t_{total} = t_1 + t_2 + \ldots + t_7 = 1 + 5 + 4 + 3 + 5 + 6 + 5 = 29$ minutes, and the theoretical lower-bound for the number of stations is $m^* = \lceil t_{total}/c \rceil = \lceil 29/10 \rceil = \lceil 2.9 \rceil = 3$. The precedence constraint for Task 2 indicates that its processing requires all its predecessor tasks {1} to be completed. Task 2 must be completed to allow *all successors* {3, 5, 6} to start. Task 1 is the *direct predecessor*, and Tasks 3 and 5 are the *direct successors* of Task 2.

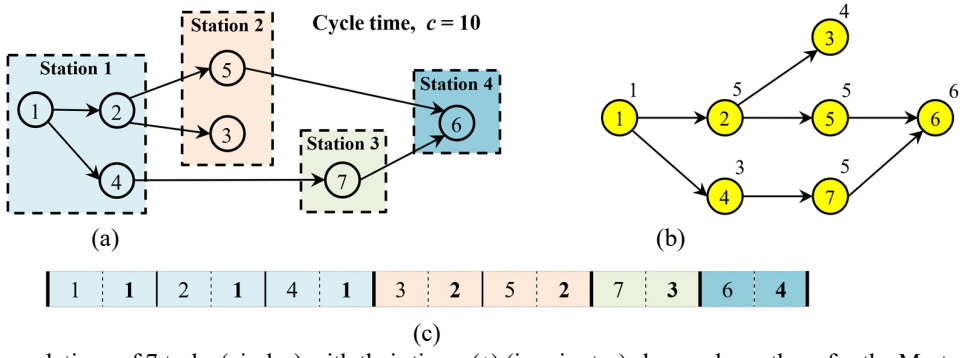


Fig. 1. (a) Precedence relations of 7 tasks (circles) with their times (t_j) (in minutes) shown above them for the Mertens (1967) benchmark problem. (b) A feasible assembly line configuration of the tasks assigned to 4 workstations. (c) Encoding of the configuration shown in (b).

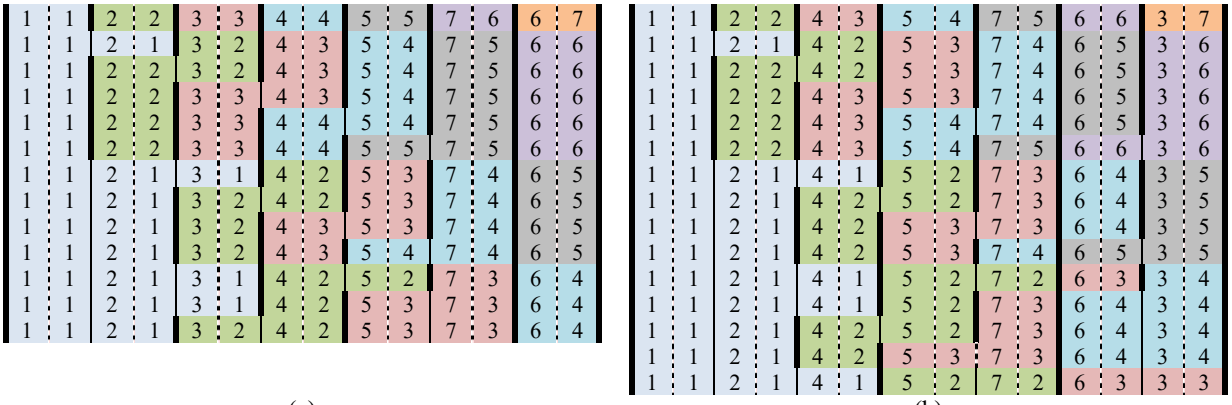
2.1 How huge is the search space of the SALBP1 problem?

The total number of possible permutations of n tasks is $n!$ and the minimum number of possible workstations for assembling a particular task is m^*. Therefore, the total number of feasible and infeasible solutions (i.e., the size of the entire search space) equals $n!(m^*)^n$, which clearly shows that SALBP1 is NP-hard because of the combinatorial explosion of its solution space. For example, for the small precedence graph shown in Fig. 1 with $n = 7$ tasks and $m^* = 3$ (the theoretical lower-bound for the number of stations), the size of the entire search space (including feasible and infeasible solutions) equals $7! \times 3^7 = 11{,}022{,}480$.

In addition, because a solution encoding (such as the one shown in Fig. 1(c)) has $2n$ cells, the diameter of the search space (i.e., the maximum distance between any two solutions) equals $2n$ as all cells in the two solutions may be different. In finding a feasible solution to SALBP1, the precedence relations between the tasks must be observed to accommodate technological and organizational conditions. This adds more complexity to finding the optimal solution that is also feasible. For the precedence graph shown in Fig. 1 with $n = 7$ tasks, $m^* = 3$ stations, and Cycle time $c = 10$ minutes, the total number of feasible solutions that respect precedence relations is only 364 solutions (out of more than 11 million!). This clearly shows how hard it is to find the optimal feasible solution in the vast search space of the SALBP1. For more clarification, all the feasible solutions to the sequences [1, 2, 3, 4, 5, 7, 6] (13 in total) and [1, 2, 4, 5, 7, 6, 3] (16 in total) are shown in Fig. 2. For the solutions in Fig. 2(a), the minimum number of workstations is $m = 4$, and for the solutions in Fig. 2(b), the minimum number of workstations is $m = 3$, which is equal to the theoretical optimal number of stations $m^* = 3$. Also, in general, the maximum number of workstations is equal to the number of all tasks (e.g., $n = 7$) which refers to the situation of each task being processed in a separate workstation. It is noted that the remaining $364 - (13+16) = 335$ feasible solutions are based on other feasible sequences (i.e., those that comply with the precedence relations) of the tasks.

3. Steps of Fitness Landscape Analysis

The landscape of a problem is completely defined by properties such as the objective function, neighborhood, and types of solution representation. A graph $G = (V, E)$ can be used to define the search space of a problem, where V is the set of vertices and E is the set of edges. Each vertex corresponds to a solution to the problem represented by some encoding, and each edge corresponds to a move operator used to generate new solutions.



(a) (b)

Fig. 2. All solutions generated based on the feasible sequences of (a) [1, 2, 3, 4, 5, 7, 6] and (b) [1, 2, 4, 5, 7, 6, 3]. Each row represents the encoding of a solution, in which thick solid lines separate workstations colored uniquely, thin solid lines separate tasks in the same workstation, and dashed lines distinguish task and workstation numbers.

Two solutions s_1 and s_2 are *neighbors* if s_1 (resp. s_2) can be reached from s_2 (resp. s_1) by implementing a move operator. Neighbor solutions are connected via an edge in the graph. Computing and analyzing statistical measures, such as the distribution and correlation of local optima in the landscape of the problem, can guide and enable us to infer the ruggedness and shape of the fitness landscape, thus proposing a suitable metaheuristic in that context (Talbi, 2009).

In this section, we present the steps of our fitness landscape analysis for the SALBP1, briefly listed as follows:

1. *Selection of Test Problems* – Our FLA is performed on 83 test problem instances based on 11 sample assembly line balancing problems with various cycle times.
2. *Generation of an Initial Population* – For each test problem, we first generate an initial uniform random population U of 1000 random starting assembly line configurations (solutions), each encoded using the method shown in Fig. 1(c).
3. *Performing Local Search* – A simple hill-climbing local search algorithm is applied to each solution until either a local optimum is reached or a preset number of iterations (e.g., 20) is completed. We then define the optimal population O as the set of all locally optimal solutions obtained from the local search.

Calculation of FLA Measures – By having populations U and O, FLA is performed for the studied test problems by calculating some statistical measures of the correlations and distribution of the local optima in the search space of the SALBP1 problem.

3.1 Step 1: Selection of Test Problems

Because the structure of the problem instance affects the fitness landscape of a problem, we investigated 83 SALBP1 instances from 11 different SALBP1 benchmark problems with various cycle times taken from the well-known Scholl database[1], as shown in Table 1, to reach a reliable and context-free conclusion regarding the fitness landscape shape of the general SALBP1 problem. Table 1 also presents the size of the solution space of each benchmark instance calculated based on the discussion in Sec. 2.1.

3.2 Step 2: Generation of an Initial Population

To generate the set U, an initial uniform random population of solutions of size N (= 1000) with sufficient diversity, we first created N random permutations of n tasks (called *priority lists*) and then built solution representations based on the lists. To do so, we employed two types of task assignment procedures, *shallow* and *deep*, each producing half of the population, as described below.

[1] https://assembly-line-balancing.de/sualbsp/data-set-of-scholl-et-al-2013/

Table 1
The specifications of the investigated benchmark test problems

Test	NT	CT	LB	TMIN	TMAX
Arcus–83	83	5048	16	3.45E+224	7.578554 E+283
		5853	14	5.31E+219	
		6842	12	1.47E+214	
		7571	11	1.08E+211	
		8412	10	3.95E+207	
		8898	9	6.28E+203	
		10816	8	3.57E+199	
Arcus–111	111	5755	27	1.341586E+339	1.892741 E+407
		8847	18	3.814949E+319	
		10027	16	8.008639E+313	
		10743	15	6.199696E+310	
		11378	14	2.93E+307	
		17067	9	1.47E+286	
Heskiaoff	28	138	8	5.90E+54	1.010568 E+70
		205	5	1.14E+49	
		216	5	1.14E+49	
		256	4	2.20E+46	
		324	4	2.20E+46	
		342	3	6.97E+42	
Jackson	11	7	8	3.43E+17	1.138873 E+19
		9	6	1.45E+16	
		10	5	1.95E+15	
		13	4	1.67E+14	
		14	4	1.67E+14	
		21	3	7.07E+12	
Jaeschke	9	6	8	4.87E+13	1/405871 E+14
		7	7	1.46E+13	
		8	6	3.66E+12	
		10	4	9.51E+10	
		18	3	7.14E+09	
Mitchell	21	14	8	4.71E+38	2.985033 E+47
		15	8	4.71E+38	
		21	5	2.44E+34	
Sawyer	30	25	14	6.42E+66	5.461321 E+76
		27	13	6.95E+65	
		30	12	6.30E+64	
		36	10	2.65E+62	
		41	8	3.28E+59	
		54	7	5.98E+57	
		75	5	2.47E+53	
Scholl–297	297	1394	50	4.496614E+1111	2.933998 E+1341
		1422	50	4.496614E+1111	
		1452	48	2.440292E+1106	
		1483	47	4.697406E+1103	
		1515	46	7.904426E+1100	
		1548	46	7.904426E+1100	
		1584	44	1.459618E+1095	
		1620	44	1.459618E+1095	
		1659	42	1.458255E+1089	
		1699	42	1.458255E+1089	
		1742	40	7.423552E+1082	
		1787	39	4.027009E+1079	
		1834	38	1.796894E+1076	
		1883	37	6.526891E+1072	
		1935	36	1.908258E+1069	
		1991	35	4.436122E+1065	
		2049	34	8.091568E+1061	
		2111	33	1.141391E+1058	
		2177	32	1.225569E+1054	
		2247	31	9.845038E+1049	
		2322	30	5.805064E+1045	
		2402	29	2.460366E+1041	
		2488	28	7.323685E+1036	
		2580	27	1.492218E+1032	
		2680	26	2.022454E+1026	
		2787	25	1.765944E+1022	
Tonge	70	176	21	4.30E+192	1.718968 E+229
		364	10	1.20E+170	
		410	9	7.51E+166	
		468	8	1.97E+163	
		527	7	1.72E+159	
Mertens	7	6	6	1.41E+09	4.150657 E+9
		7	5	3.94E+08	
		8	5	3.94E+08	
		10	3	1.10E+07	
		15	2	6.45E+05	
		18	2	6.45E+05	
Kilbridge & Wester	45	57	10	1.20E+101	2.967392 E+130
		79	7	1.28E+94	
		92	6	1.24E+91	
		110	6	1.24E+91	
		138	4	1.48E+83	
		184	3	3.53E+77	

NT = No. of tasks (n), CT = Cycle time (c), LB = Lower bound of the number of stations (m^*),TMIN = Theoretical minimum size of the entire search space, $n!(m^*)^n$, *TMAX* = Theoretical maximum size of the entire search space, $n!(n)^n$

Shallow task assignment: In this procedure, a random priority list (permutation) of tasks (named L) is generated, and then the first station is created with a total time $tt_1 = 0$ and idle time $I_1 = c$. Next, through an iterative process and starting from the left, the first unassigned task in L (say task j), which has a completion time less than or equal to the idle time of the first station, is assigned to that station, and the station's total and idle times are updated as $tt_1 = tt_1 + t_j$ and $I_1 = I_1 - t_j$. After each assignment, list L is scanned again to find other tasks to be assigned to the current station. If such a task cannot be found, the number of stations is incremented by one (i.e., a new station is created with total and idle times equal to zero and c, respectively) and the list is re-scanned from the left to assign another task to the newly created station. This procedure is repeated until all the tasks are assigned to some station.

We name this method 'shallow' because the resulting solutions may be infeasible due to the precedence relations of the tasks being not considered. However, the generation of such solutions contributes to the diversity of the initial pool, which is essential for the success of metaheuristics. For example, applying this procedure to the random priority list L = [3, 1, 4, 5, 7, 6, 2] for the tasks presented in Fig. 1(a) generates the infeasible solution ⟨(3, 1), (1, 1), (4, 1), (5, 2), (7, 2), (6, 3), (2, 4)⟩ after sorting based on the station number. This procedure is repeated for $N/2$ random priority lists to create half of the initial population U.

Deep task assignment: In this procedure, a random priority list (permutation) of tasks (L) is generated. Then, in each iteration i of the procedure, the first unassigned task in L (e.g., task $\pi_i = j$) is selected and assigned to the current station k only if $I_k \geq t_j$ and all direct predecessors of task j (DP_j) have already been assigned. For example, applying this procedure to the random priority list L = [3, 1, 4, 5, 7, 6, 2] for the tasks presented in Fig. 1(a) generates the solution ⟨(1, 1), (4, 1), (7, 1), (2, 2), (3, 2), (5, 3), (6,

4)⟩ after sorting based on the station number. Basically, deep task assignment is based on the shallow task assignment, but additionally considers precedence relations.

We name this method 'deep' because the resulting solutions are merely feasible, as the precedence relations of the tasks are considered. This procedure is repeated for $N/2$ random priority lists to create the other half of the initial population U. The generation of random priority lists, followed by the above two procedures, constitutes the algorithm used to create a uniform random population U in the landscape analysis of SALBP1.

3.3 Step 3: Performing Local Search

After generating the initial population U of 1000 solutions using the shallow and deep task assignment methods, the steepest descent (hill-climbing) local search is performed on each solution to reach a solution with a local minimum fitness value. In each iteration of the local search, a neighboring feasible solution to an initial solution is generated using a randomly selected neighborhood generation operator (introduced in this subsection) and is kept only if it improves (decreases) the fitness function value. Iterations repeat until either a local optimum is reached, or a certain number of non-improving iterations are repeated. Fig. 3. Presents the pseudocode of the used local search.

Algorithm Local_Search

1. Set s = current solution
2. **While** termination criterion is not met
3. Generate s' // Generate a neighbor by randomly applying an operator on s
4. **If** s' is feasible AND $f(s') < f(s)$ // As calculated in Sec. 3.3.2
5. **Then** $s \leftarrow s'$ // Replace s with the better neighbor s'
6. **End**
7. **End**
8. $s^* \leftarrow s'$ // Record s^* as the local optimum

Fig. 3. Procedure of the hill-climbing local search performed on each initial solution of the population U.

3.3.1 Neighborhood-Generation Operators

We use four different neighborhood-generation operators–*exchange*, *insertion*, *inversion*, and *float shift*–to generate neighboring solutions, as explained below. The fourth operator, float shift, is originally designed by us.

Exchange operator: This operator randomly selects two tasks from a task priority list and exchanges (swaps) their positions. As an example, in Fig. 4, consider the priority list L_1 and its corresponding solution s_1 obtained by the deep task assignment. Two tasks, 3 and 2, were randomly selected and switched to yield a new list L_2. Then, the new neighboring solution s_2 is generated by implementing the deep task assignment on L_2 and sorting based on the workstations. By applying this operator, the total size of the neighborhood for any solution equals the number of ways two tasks can be selected in a list of size n, which can be expressed as $C(n, 2) = n(n-1)/2$, where n is the total number of tasks and $C(\ )$ is the combination function.

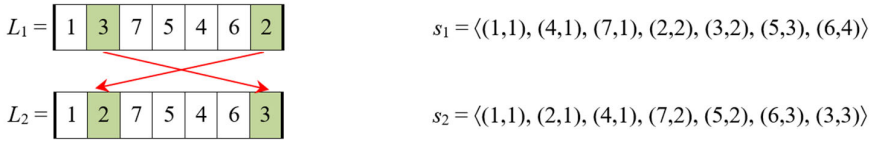


Fig. 4. The exchange neighborhood generation operator.

Insertion operator: In this operator, a task in the priority list is randomly selected and relocated to another position on the list. For example, in Fig. 5, assume that for the initial solution s_1 and its corresponding task list L_1, task 2 is randomly selected and moved to the new random position 2, resulting in task list L_2 and neighboring solution s_2 (after the deep assignment of stations). By applying this operator, the size of the neighborhood for any solution is $n(n-1)$, because n tasks can be selected as the beginning, and $n-1$ different positions (all but the current position of the task) can be selected as the destination of the relocation.

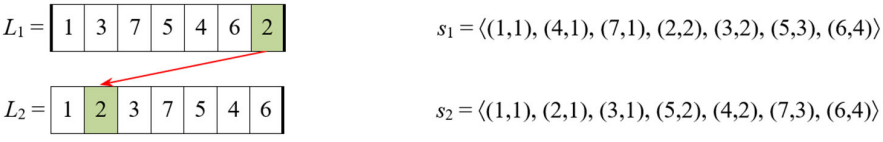


Fig. 5. The insertion neighborhood generation operator.

Inversion operator: This operator randomly selects two positions in the list and reverses the sequence of all the tasks between these two positions. For example, in Fig. 6, for the initial solution s_1 and its corresponding task list L_1, two positions (1 and 7) are randomly selected, and the new list after applying the inversion operator is L_2, which yields the neighboring solution s_2 after deep task assignment. Through enumeration of all possible pairs of tasks that are two or more positions apart, it can be shown that for a list of n tasks, by applying this operator, the total size of the neighborhood of any solution is $n(n-1)/2 - (n-1) - (n-2) = (n-2)(n-3)/2$.

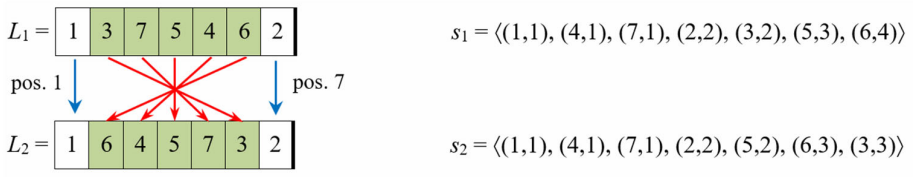


Fig. 6. The inversion neighborhood generation operator.

Float shift operator: Before introducing this neighborhood operator, we define a new concept called the *float* of a task as follows. For any task j in a feasible priority list L, Float F_j is the set of positions π_f (locations) in the array L between (and not including) the latest direct predecessor and the earliest direct successor of task j. Here, a *feasible priority list* is a list of tasks in a feasible solution with satisfied assembly precedence relations; that is, no task appears in the list before any of its predecessors. For example, considering the precedence graph in Fig. 1(a), the floats of tasks in $L = [1, 2, 4, 3, 5, 7, 6]$ will be as follows: $F_1 = \{\pi_1\}$, $F_2 = \{\pi_2, \pi_3\}$, $F_3 = \{\pi_3, \pi_4, \pi_5, \pi_6, \pi_7\}$, $F_4 = \{\pi_2, \pi_3, \pi_4, \pi_5\}$, $F_5 = \{\pi_3, \pi_4, \pi_5, \pi_6\}$, $F_6 = \{\pi_7\}$, and $F_7 = \{\pi_4, \pi_5, \pi_6\}$.

The float shift operator randomly selects a task in a feasible priority list and relocates it to another position in its float set, thereby maintaining the tasks between all its predecessors and successors. Fig. 7 shows the result of implementing this operator on a given feasible priority list for Task 3, which moves from its current 4th position to the 7th position, which is within the float of Task 3, $F_3 = \{\pi_3, \pi_4, \pi_5, \pi_6, \pi_7\}$. The neighborhood size of this operator for a list with length n has an upper bound of $n(n-2)$ because n tasks can be selected and relocated to $n-2$ positions (reserving the first and last positions of the list for the predecessor and successor tasks). More precisely, the neighborhood size is equal to $\sum_{i=1}^{n}\left(|F_i|-1\right)$, where $|F_i|$ is the cardinality (size) of the set F_i. An advantage of the Float shift operator is that it maintains the feasibility of the solutions, while there is no guarantee that solutions obtained from the exchange, insertion, and inversion operators will remain feasible.

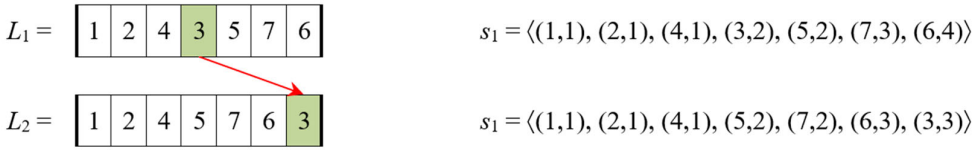


Fig. 7. The Float Shift neighborhood generation operator.

An important property of any neighborhood generating operator is its *connectivity* property, which guarantees that any solution in the search space can be reached from any starting solution through successive applications of merely that operator. Fortunately, all the abovementioned neighborhood generation operators possess connectivity properties. Additionally, it is worth noting that after applying the exchange, inversion, or float-shift operators, the number of stations changed from four to three, thereby minimizing the number of required workstations.

3.3.2 Calculating the Fitness of a Solution

Two important properties of any solution to SALBP1 are its feasibility and quality, which are used to guide the search to converge to the final solution by selecting the best neighbor of a solution in each iteration. A solution $s = \langle(\pi_1, w_1), (\pi_2, w_2), \ldots, (\pi_n, w_n)\rangle$ is feasible if any task π_i ($\forall i = 2, \ldots, n$) is not a predecessor of any already assembled task π_j ($\forall j < i$). More formally, solution s is feasible if the number of its unsatisfied precedence constraints (UPC) equals zero. In fact, $UPC(s_l)$ is the number of times a predecessor of a particular task is assembled after that task has been assembled in solution s_l, calculated in Eq. (1), where the variable *apm* is defined as in Sec. 2:

$$UPC(s_l) = \sum_{i=2}^{n}\sum_{j=1}^{i-1} apm_{\pi_i,\pi_j} = 0\,. \tag{1}$$

In addition, to measure the feasibility of any solution s_l generated and evaluated during the search, we define the *Percentage of Unsatisfied Precedence Constraints*, $PUPC(s_l)$, as follows:

$$0 \le PUPC(s_l) = \frac{\text{total number of unsatisfied precedence constraints}}{\text{total number of possible pairs of tasks}} = \frac{UPC(s_l)}{C(n,2)} = \frac{2\sum_{i=2}^{n}\sum_{j=1}^{i-1} apm_{\pi_i,\pi_j}}{n(n-1)} \le 1. \quad (2)$$

As part of the fitness of solution s_l, we are also interested in minimizing m, the number of workstations in the assembly line, as well as the *Smoothness Index* $SI(s_l)$ calculated in Eq. (3), as presented by (Baykasoglu, 2006; Hong & Cho, 1999), in which tt_k and $tt_{\max}$ are defined earlier. A small $SI(s_l)$ for solution s_l implies that the processing times in all workstations are almost equal, which contributes to a smooth and level distribution of the workload and enhances worker and equipment utilization.

$$0 \le SI(s_l) = \sqrt{\frac{\sum_{k=1}^{m}\left(tt_{\max} - tt_k\right)^2}{m}} < c\,. \quad (3)$$

The total fitness function of s_l can now be defined as a weighted aggregation of the above criteria, expressed as

$$f(s_l) = \left(1 + \alpha \cdot PUPC(s_l)\right) \cdot \left(\beta \cdot SI(s_l) + \gamma \cdot m\right). \quad (4)$$

The first term in Eq. (4) counts the total number of unsatisfied precedence constraints among all checked precedence constraints for solution s_l, and the second term calculates the total weighted sum of the smoothness indices and the number of stations in the assembly line. In the SALBP1 we try to minimize the function $f(s)$ so that feasible and smooth solutions with least workstations are favored, and therefore to bias the search toward feasible solutions (as feasibility is the most important feature of a solution), we set the values of the weighting factors in (4) to $\alpha = 3$ (the coefficient of *PUPC*), $\beta = 1$ (the coefficient of *SI*), and $\gamma = 2$ (the coefficient of m), thus prioritizing feasibility over smoothness, and smoothness over the number of workstations.

3.4 Step 4: Calculation of FLA Measures

Two groups of statistical measures are calculated to describe the properties of the fitness landscape of the SALBP1 problem: Distribution measures and Correlation measures, as defined below:

(1) *Distribution measures* study the topology of locally optimal solutions in the objective and search spaces. For landscape analysis, several measures are calculated for both the *U* and *O* populations, together with their relative variations (Δ). Our investigated distribution measures are *Dmm*(*P*) (the average distance between each pair of solutions) and its variation Δ_{Dmm}, *ent*(*P*) (entropy in the search space) and its variation Δ_{ent}, *Amp*(*P*) (amplitudes of solutions) and its variation Δ_{Amp}, and *Gap*(*P*). It is noted that we designed a new mathematical formula for calculating the entropy in the search space of the SALBP1, as presented in Sec. 3.4.1. Also, whenever required, we use the *Hamming* distance $d_H(s_1, s_2)$ to measure the distance between any two solutions, defined as the number of bitwise differences in their encodings. For example, $d_H(s_1, s_2) = 7$ for solutions s_1 and s_2 in Fig. 4.

(2) *Correlation measures* are used to analyze the correlation between the relative distance of solutions and their quality, as well as the correlation between the distance of solutions to the best-known solution and their quality gap. These measures include *Lmm* (average length of the walks), $\rho(d_H)$ (autocorrelation function), and *FDC* (fitness–distance correlation). FDC analysis can be visualized using the FDC scatterplot, presented in the Sec. 6.

Table 2 lists and defines the major distribution and correlation measures typically used for FLA, expressed for a general population *P* of size $|P|$. When needed, *P* is replaced with populations *U* or *O*. More details about these measures can be found in (Talbi, 2009).

3.4.1 A New Entropy Measure for SALBP1

Entropy, or lack of predictability, is a concept first used in the 19th century for describing the degree of disorder or randomness in a thermodynamic system, which then was statistically formulated by Paul Shannon in 1948 as a measure of random losses of information in telecommunication signals. The concept was later evolved as a measure of irregularity or diversity in a population in Information Theory. Entropy has found wide applications in many scientific fields such as thermodynamics, physics, information sciences, biology, cosmology, and economics. In the context of optimization problems, the concentration or dispersion of solutions can be assessed through the concept of entropy. When the entropy is weak or close to zero, it indicates a concentration of solutions, whereas high entropy implies a significant dispersion of solutions within the search space. Although Shannon proposed the general formulation for the entropy of population *X* in terms of a discrete set of probabilities p_i as follows:

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log p(x_i) \quad (5)$$

Table 2
Distribution and Correlation measures and their average numeric values for FLA

	Formula	Description
Distribution Measures	$\frac{2}{\lvert P\rvert} \le Dmm(P) = \frac{\sum_{i=1}^{\lvert P\rvert}\sum_{j=1,\, j\neq i}^{\lvert P\rvert} d_H(s_i, s_j)}{\lvert P\rvert(\lvert P\rvert-1)\cdot \max_{\forall i,j}\{d_H(s_i,s_j)\}} \le 1$	The normalized average Hamming distance between any two solutions of the population set *P*. It is an indicator of the concentration of a population *P* of solutions in the whole search space *S*.
	$\Delta_{Dmm} = \frac{Dmm(U) - Dmm(O)}{Dmm(U)} \le 1$	The relative difference between normalized average Hamming distances of initial uniform random solutions (*Dmm*(*U*)) and the normalized average Hamming distances of local optimum solutions (*Dmm*(*O*)).
	$0 \le ent(P) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{m} q_{ijk}\left(\lvert P\rvert - q_{ijk}\right)}{n\left(\lvert P\rvert-1\right)} \le 1$	*Entropy* ∈ [0, 1]: The diversity of solutions in the search space, where q_{ijk} is the number of all solutions in *P* that have task *i* in the *j*-th position of the assembly sequence assembled in workstation *k*, or mathematically, $\pi_j = i$ and $w_j = k$. It is an indicator of the diversity of solutions in a population.
	$\Delta_{ent} = \frac{ent(U) - ent(O)}{ent(U)} \le 1$	*Entropy variation*: The relative distance between the entropy of initial uniform random solutions (*ent*(*U*)) and the entropy of local optimum solutions (*ent*(*O*)).
	$0 \le Amp(P) = \frac{\lvert P\rvert \cdot \left(f(s^w) - f(s^b)\right)}{\sum_{\forall s\in P} f(s)} \le \lvert P\rvert$	*Amplitude*: the relative difference between the objective values of the worst (i.e., maximum) (s^w) and the best (i.e., minimum) (s^b) solutions in the population set *P*.
	$\Delta_{Amp} = \frac{Amp(U) - Amp(O)}{Amp(U)} \le 1$	The relative differences between the amplitudes of initial (*U*) and locally optimal (*O*) solutions. It indicates the distribution of optimum solutions in the search space and reveals whether the search space is rugged or not.
	$Gap(O) = \frac{\sum_{\forall s\in O}\left(f(s) - f(s^*)\right)}{\lvert O\rvert \cdot f(s^*)}$	*Gap*: the relative difference between the objective value of local optimum solutions and the global optimum solution s^*.
Correlation Measures	$Lmm = \frac{\sum_{i=1}^{\lvert P\rvert} l(s_i)}{\lvert P\rvert}$	*Average length of walk*: required number of steps needed for finding a local optimum solution from a random solution $s_i \in U$.
	$-1 \le \rho(d) = \frac{\sum_{\forall i,j\in P,\; d = d_H(s_i,s_j)} \left(f(s_i) - \bar{f}\right)\cdot\left(f(s_j) - \bar{f}\right)}{\lvert P\rvert \cdot \sigma_f^2} \le 1$	*Autocorrelation function*: computes the correlation of all solution pairs having a Hamming distance of d_H in the search space, where $\bar{f}$ is the average fitness value of the whole population and σ_f^2 is its variance.
	$FDC = \frac{\mathrm{cov}(F,D)}{\sigma_f \sigma_d}$	*Fitness-Distance Correlation*: the correlation between the quality of local optimum solutions and their distance from the global optimum solution. The set $F = \{f_1, f_2, \ldots, f_n\}$ contains the fitness values of solutions and the set $D = \{d_{H1}, d_{H2}, \ldots, d_{Hn}\}$ contains the Hamming distances of the solutions to the best-known solution.

There is no universally applicable formulation for entropy as it varies depending on the specific optimization problem being addressed. To the best of our knowledge, there has been no entropy formulation customized for Assembly Line Balancing problems in general, and the SALBP1 in particular. To fill this gap, in this paper, we are introducing an original formulation for entropy as follows:

$$0 \le ent(P) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{m} q_{ijk}\left(\lvert P\rvert - q_{ijk}\right)}{n\left(\lvert P\rvert-1\right)} \le 1 \tag{6}$$

in which q_{ijk} is named the *locality variable* and counts the number of all solutions in population *P* that have task *i* in the *j*-th position of the assembly sequence assembled in workstation *k*. Mathematically expressed, $\pi_j = i$ and $w_j = k$.

For a problem instance with n tasks and m workstations, the total number of locality variables will be n^2m, and in a population P, the minimum and maximum values of any locality variable q_{ijk} is 0 and $|P|$, respectively. For example, for the 13-member population of solutions shown in Fig. 2(a), some locality variables are as follows: $q_{1,1,1} = 13$, $q_{2,2,1} = 8$, $q_{2,2,2} = 5$, $q_{3,3,1} = 3$, $q_{7,6,6} = 1$, $q_{7,6,5} = 5$, $q_{6,7,4} = 3$, $q_{2,1,k\ (\forall k)} = 0$, and $q_{2,2,3} = 0$.

In a population P of solutions for SALBP1, the maximum entropy $ent(P) = 1$ indicates the highest possible diversity of solutions, which occurs in the following cases: (1) All solutions have distinct task sequences; or (2) If two or more solutions have completely or partially equal task sequences, same tasks with equal positions among those solutions are allocated to workstations in a different way in each solution. In either of these cases, for any task i, there will be only one unique combination of j and k (among all $n \times m$ possible combinations) for which $q_{ijk} = 1$, and for the remaining $n \cdot m - 1$ combinations of j and k, we have $q_{ijk} = 0$. Also, in a population P of solutions for SALBP1, the minimum entropy $ent(P) = 0$ indicates the lowest possible diversity of solutions, which occurs when all solutions in the population are completely similar and therefore only for one unique combination of i, j, and k, $q_{ijk} = |P|$, and for all remaining $n^2m - 1$ combinations of i, j, and k, $q_{ijk} = 0$.

4. Fitness Landscape Analysis Computational Results

For conducting the fitness landscape analysis for the SALBP1, the hill-climbing local search algorithm was applied to a population U of 1000 solutions for each of the 83 instances introduced in Sec. 3.1, to obtain 83,000 local optimal solutions, and then the statistical measures introduced in Sec. 3.4 were calculated for each instance. In addition, the average values of all measures for each test problem set, as well as the total averages of all measures over all the problem sets were calculated. The computational results are reported in Table 3. Here we expand on some noticeable facts:

1. While most Δ_{Dmm} values are positive, there are few negative values, mostly in the Tonge dataset, which occur when the average distance between all local optima pairs were greater than the average distance between all pairs of initial random solutions. This implies that in those instances, local optimal solutions cover a larger part of the search space compared to the initial random solutions.
2. While most Δ_{ent} values are positive, there are few negative values, mostly in the Sawyer, Scholl–297, and Tonge datasets, which imply that the average diversity of locally optimal solutions in the search space was greater than those of initial random solutions.
3. While most *FDC* values are positive, there are few negative values, mostly in the Heskiaoff dataset, which show that the landscape is 'misleading', meaning that solutions closer to the global optimum are not necessarily high quality or feasible, leading the search away from the optimum. This fact will be further discussed in Sec. 6.

Table 3
Results of computing statistical measures for 83 SALBP1 instances from 11 benchmark problem sets.

Instance №	Cycle Time, c	Distribution measures							Correlation measures				
		$Dmm(O)$	Δ_{Dmm}	$ent(O)$	Δ_{ent}	$Amp(O)$	Δ_{Amp}	$Gap(O)$	Lmm	$\rho(2)$	$\rho(4)$	$\rho(6)$	FDC
Arcus–83 test problems													
1	5048	0.307	0.2133	0.013	−0.0974	3.59	−0.909	0.641	43.50	0.5000	0.4686	0.4978	0.66
2	5853	0.365	0.0056	0.069	0.0023	2.16	−0.020	2.417	1.92	0.5000	0.5000	0.4461	0.79
3	6842	0.304	0.1575	0.065	0.0637	2.42	−0.404	1.281	26.02	0.4995	0.4050	0.4383	0.14
4	7571	0.308	0.1246	0.065	0.0580	2.61	−0.248	0.705	17.98	0.5000	0.5000	0.4957	0.50
5	8412	0.294	0.1193	0.064	0.0754	2.20	−0.012	0.551	22.84	0.5000	0.4927	0.5000	−0.04
6	8898	0.293	0.1236	0.069	−0.0784	4.58	−0.741	3.139	25.01	0.4998	0.5000	0.5000	0.67
7	10816	0.319	0.0044	0.069	−0.0027	1.56	−0.007	0.947	1.52	0.5000	0.4998	0.4912	0.12
Average		0.313	0.1069	0.059	0.0029	2.73	−0.334	1.383	19.83	0.4999	0.4808	0.4813	0.41
Arcus–111 test problems													
8	5755	0.422	0.0293	0.039	−0.0109	1.97	−0.145	1.612	8.09	0.5000	0.4998	0.4912	0.87
9	8847	0.381	0.0899	0.038	0.0304	2.55	−0.405	1.957	22.53	0.5000	0.5000	0.4963	0.82
10	10027	0.349	0.1319	0.038	0.0432	2.58	−0.444	1.431	30.34	0.5000	0.5000	0.5000	0.72
11	10743	0.345	0.1183	0.038	0.0375	2.34	−0.262	1.072	27.47	0.5000	0.5000	0.5000	0.65
12	11378	0.285	0.2679	0.036	0.0961	3.97	−1.091	0.412	39.30	0.5000	0.5000	0.5000	0.73
13	17067	0.358	0.0126	0.039	0.0068	2.31	−0.178	10.427	8.64	0.5000	0.5000	0.5000	0.24
Average		0.357	0.1083	0.038	0.0339	2.62	−0.421	2.818	22.73	0.5000	0.5000	0.4979	0.67

Table 3
Results of computing statistical measures for 83 SALBP1 instances from 11 benchmark problem sets (Continued)

Instance №	Cycle Time, c	Distribution measures							Correlation measures				
		$Dmm(O)$	Δ_{Dmm}	$ent(O)$	Δ_{ent}	$Amp(O)$	Δ_{Amp}	$Gap(O)$	Lmm	$\rho(2)$	$\rho(4)$	$\rho(6)$	FDC
Heskiaoff test problems													
14	138	0.368	0.0866	0.592	0.0406	1.60	−0.181	0.896	19.01	0.5000	0.5000	0.5000	0.61
15	205	0.337	0.0803	0.588	0.0340	1.62	−0.035	4.419	21.37	0.5000	0.4998	0.4672	−0.04
16	216	0.355	−0.0160	0.606	−0.0108	1.96	−0.171	2.302	9.27	0.5000	0.4983	0.4443	0.42
17	256	0.343	0.0054	0.605	0.0058	1.50	−0.051	9.092	4.58	0.4800	0.5000	0.4800	−0.20
18	324	0.334	0.0048	0.603	0.0066	1.44	−0.056	2.205	6.08	0.5000	0.5000	0.5000	0.44
19	342	0.315	0.0219	0.602	0.0116	1.89	−0.362	13.491	11.59	0.5000	0.4997	0.4996	−0.04
Average		0.342	0.0304	0.599	0.0146	1.67	−0.143	5.401	11.98	0.4967	0.4996	0.4819	0.20
Jackson test problems													
20	7	0.139	0.5734	0.189	0.4584	0.87	−0.109	0.017	10.31	0.5000	0.4984	0.4996	0.72
21	9	0.327	0.0001	0.353	0.0002	0.82	0.000	0.301	0.01	0.4995	0.4995	0.4958	0.84
22	10	0.214	0.3386	0.256	0.2937	1.20	−0.274	0.215	14.77	0.4763	0.4892	0.5000	0.90
23	13	0.192	0.3459	0.246	0.3049	1.42	−0.114	0.078	14.43	0.5000	0.4984	0.4976	0.63
24	14	0.281	0.0000	0.348	0.0000	1.07	0.000	0.428	0.02	0.5000	0.4540	0.4996	0.71
25	21	0.293	0.0005	0.355	0.0006	1.09	−0.001	0.723	0.04	0.5000	0.4067	0.4637	0.18
Average		0.241	0.2097	0.291	0.1763	1.08	−0.083	0.294	6.60	0.4960	0.4744	0.4927	0.66
Jaeschke test problems													
26	6	0.246	0.0123	0.931	0.0093	0.66	−0.003	0.199	0.01	0.5000	0.4984	0.4996	0.85
27	7	0.296	0.0000	0.955	0.0000	0.77	0.000	0.241	0.00	0.5000	0.4995	0.4958	0.93
28	8	0.271	0.0110	0.885	0.0092	0.83	−0.002	0.257	0.01	0.4443	0.4984	0.4976	0.90
29	10	0.240	0.0137	0.914	0.0106	1.10	−0.004	0.409	0.34	0.5000	0.4984	0.4976	0.88
30	18	0.288	0.0000	0.917	0.0000	1.04	0.000	0.178	0.00	0.5000	0.4067	0.4958	0.73
Average		0.268	0.0074	0.920	0.0058	0.88	−0.002	0.257	0.07	0.4889	0.4803	0.4973	0.86
Kilbridge & Wester test problems													
31	57	0.372	0.0042	0.047	0.0013	1.32	−0.035	1.047	2.69	0.5000	0.5000	0.4801	0.77
32	79	0.325	−0.0870	0.043	−0.0981	1.98	−0.303	0.333	26.84	0.5000	0.5000	0.5000	0.67
33	92	0.286	0.0426	0.217	0.0105	2.33	−0.185	1.952	24.76	0.4984	0.5000	0.5000	0.54
34	110	0.281	−0.0530	0.415	−0.0449	1.68	−0.001	0.419	18.60	0.5000	0.5000	0.4984	0.36
35	138	0.300	−0.0890	0.436	−0.0843	1.82	−0.165	6.293	27.10	0.5000	0.5000	0.5000	0.28
36	184	0.273	0.0000	0.403	0.0001	1.91	−0.002	13.700	0.04	0.5000	0.5000	0.5000	0.44
Average		0.306	−0.0300	0.260	−0.0359	1.84	−0.115	3.957	16.67	0.4997	0.5000	0.4964	0.51
Mitchell test problems													
37	14	0.243	0.8890	0.152	0.0357	2.11	−0.163	0.306	10.47	0.5000	0.5000	−0.1667	0.70
38	15	0.266	0.0015	0.162	0.0009	1.76	−0.001	0.411	0.09	0.5000	0.5000	0.5000	0.79
39	21	0.209	0.0503	0.142	0.0397	2.44	−0.240	0.624	8.65	0.5000	0.5000	−0.2637	0.85
Average		0.240	0.3136	0.152	0.0254	2.10	−0.135	0.447	6.40	0.5000	0.5000	0.0232	0.78
Mertens test problems													
40	6	0.204	−0.0002	0.133	−0.0003	1.62	0.0000	0.181	0.02	0.5000	0.5000	0.5000	0.79
41	7	0.229	0.0000	0.125	0.0000	1.91	0.0000	0.195	0.00	0.5000	0.5000	0.5000	0.71
42	8	0.221	0.0004	0.128	0.0005	2.12	−0.0004	0.268	0.00	−0.0714	0.3606	0.5000	0.80
43	10	0.194	0.0004	0.116	0.0004	2.57	−0.0004	0.687	0.03	0.5000	0.5000	−0.2767	0.71
44	15	0.219	0.0000	0.128	0.0000	4.21	0.0000	0.433	0.00	0.5000	−0.0714	−0.2996	0.28
45	18	0.243	0.0000	0.119	0.0000	2.05	0.0000	0.232	0.00	0.5000	−0.0714	−0.1667	0.54
Average		0.218	0.0001	0.125	0.0001	2.41	−0.0001	0.333	0.01	0.4048	0.2863	0.1262	0.64
Sawyer test problems													
46	25	0.337	−0.0170	0.947	−0.0255	1.63	−0.015	0.311	2.22	0.5000	0.4998	0.5000	0.76
47	27	0.328	0.0033	0.918	−0.0071	1.56	−0.003	0.407	0.25	0.4999	0.5000	0.5000	0.82
48	30	0.304	0.0672	0.979	−0.0254	0.51	−0.112	0.074	19.67	0.5000	0.4997	0.4912	0.54
49	36	0.281	0.0898	0.953	−0.0287	1.77	−0.204	0.190	32.33	0.5000	0.5000	0.4846	0.77
50	41	0.310	−0.0620	0.990	−0.1015	1.77	−0.187	0.608	23.54	0.5000	0.5000	0.4998	0.88
51	54	0.280	−0.0190	0.925	−0.0246	1.72	−0.040	0.864	6.05	0.4224	0.5000	0.5000	0.76
52	75	0.259	0.0112	0.879	0.0050	2.05	−0.054	0.967	5.15	0.5000	0.5000	0.5000	0.50
Average		0.300	0.0105	0.941	−0.0297	1.57	−0.088	0.489	12.75	0.4889	0.4999	0.4965	0.72

Table 3
Results of computing statistical measures for 83 SALBP1 instances from 11 benchmark problem sets (Continued)

Instance №	Cycle Time, *c*	Distribution measures							Correlation measures				
		***Dmm*(*O*)**	**Δ_{Dmm}**	***ent*(*O*)**	**Δ_{ent}**	***Amp*(*O*)**	**Δ_{Amp}**	***Gap*(*O*)**	***Lmm***	**ρ(2)**	**ρ(4)**	**ρ(6)**	***FDC***
Scholl–297 test problems													
53	1394	0.388	−0.0510	0.005	−0.0559	2.59	−0.024	1.370	7.90	0.5000	0.5000	0.5000	0.56
54	1422	0.358	−0.0320	0.005	−0.0628	1.02	−0.011	0.469	6.72	0.5000	0.5000	0.5000	0.91
55	1452	0.304	0.0672	0.979	−0.0254	0.51	−0.112	0.074	19.67	0.5000	0.4997	0.4912	0.54
56	1483	0.281	0.0898	0.953	−0.0287	1.77	−0.204	0.190	32.33	0.5000	0.5000	0.4846	0.77
57	1515	0.434	−0.0030	0.001	−0.0027	1.86	−0.007	2.069	1.40	0.5000	0.4995	0.4886	0.67
58	1548	0.447	−0.0020	0.000	−0.0060	1.34	−0.013	1.865	6.70	0.4123	0.4995	0.5000	0.74
59	1584	0.404	0.0700	0.001	0.0143	2.28	−0.229	1.444	13.63	0.5000	0.5000	0.5000	0.47
60	1620	0.409	0.0249	0.001	0.0007	2.12	−0.148	1.548	4.84	0.5000	0.4532	0.5000	0.60
61	1659	0.421	0.0031	0.001	−0.0005	1.90	−0.029	1.985	1.12	0.4999	0.5000	0.4999	0.66
62	1699	0.402	0.0356	0.001	0.0047	2.13	−0.078	1.477	7.00	0.5000	0.5000	0.5000	0.58
63	1742	0.402	0.0433	0.001	0.0061	2.39	−0.206	1.307	8.53	0.5000	0.5000	0.4875	0.58
64	1787	0.381	0.1082	0.001	0.0220	1.18	−0.033	0.717	12.38	0.5000	0.5000	0.5000	0.51
65	1834	0.381	0.1089	0.001	0.0333	0.10	−0.087	0.352	13.92	0.5000	0.5000	0.5000	0.65
66	1883	0.400	0.0359	0.001	0.0219	0.13	−0.085	0.435	13.12	0.5000	0.5000	0.5000	0.57
67	1935	0.389	0.0622	0.001	0.0129	0.13	−0.085	1.443	18.10	0.5000	0.5000	0.5000	0.53
68	1991	0.397	0.0342	0.001	0.0205	2.11	−0.078	1.420	14.02	0.5000	0.3719	0.5000	0.54
69	2049	0.382	0.0858	0.001	0.0254	1.33	−0.227	1.032	14.02	0.5000	0.5000	0.4998	−0.15
70	2111	0.382	0.0289	0.001	0.0135	1.30	0.044	0.868	17.69	0.5000	0.5000	0.5000	0.59
71	2177	0.372	0.1004	0.001	0.0276	1.34	−0.035	1.671	14.28	0.5000	0.5000	0.5000	0.19
72	2247	0.374	0.0840	0.001	0.0300	1.35	0.409	1.590	15.38	0.5000	0.5000	0.5000	0.18
73	2322	0.369	0.0973	0.001	0.0284	1.14	−0.426	0.779	18.88	0.5000	0.5000	0.5000	0.15
74	2402	0.377	0.0402	0.001	0.0267	1.60	0.289	0.413	16.30	0.5000	0.5000	0.5000	0.44
75	2488	0.389	0.0097	0.001	0.0233	1.39	0.424	0.378	18.83	0.5000	0.4999	0.5000	0.51
76	2580	0.364	0.0705	0.001	0.0255	1.44	0.411	0.503	14.34	0.5000	0.5000	0.5000	0.29
77	2680	0.395	0.0002	0.001	0.0281	1.22	0.403	0.474	17.90	0.5000	0.4996	0.5000	0.35
78	2787	0.393	0.0053	0.001	0.0294	1.31	−0.421	1.475	20.30	0.5000	0.5000	0.5000	0.24
Average		0.393	0.0373	0.001	0.0090	1.52	−0.014	0.902	11.72	0.4966	0.4932	0.4991	0.48
Tonge test problems													
79	176	0.369	−0.057	0.188	−0.0829	2.24	−0.033	0.538	14.65	0.5000	0.5000	0.5000	0.59
80	364	0.320	−0.013	0.180	−0.0250	3.59	−0.014	1.602	6.01	0.5000	0.5000	0.5000	0.67
81	410	0.301	−0.004	0.181	−0.0325	4.29	−0.331	1.516	24.91	0.5000	0.5000	0.5000	0.75
82	468	0.291	−0.001	0.177	−0.0066	3.65	−0.076	1.934	6.81	0.5000	0.5000	0.5000	0.50
83	527	0.304	−0.130	0.185	−0.0708	6.47	−0.804	1.273	42.72	0.5000	0.5000	0.5000	0.50
Average		0.317	−0.041	0.182	−0.0436	4.05	−0.252	1.373	19.02	0.5000	0.5000	0.5000	0.60
Total Average		**0.299**	**0.068**	**0.275**	**0.014**	**2.043**	**−0.144**	**1.605**	**11.616**	**0.4880**	**0.4740**	**0.4170**	**0.594**

5. Analysis of Distribution Measures

The entropy variation (Δ_{ent}) and the average normalized pairwise distance variation (Δ_{Dmm}) in the search space play key roles in evaluating the distribution of the obtained local optimal solutions in the landscape. There are three combinations of the values of these two measures (Talbi, 2009):

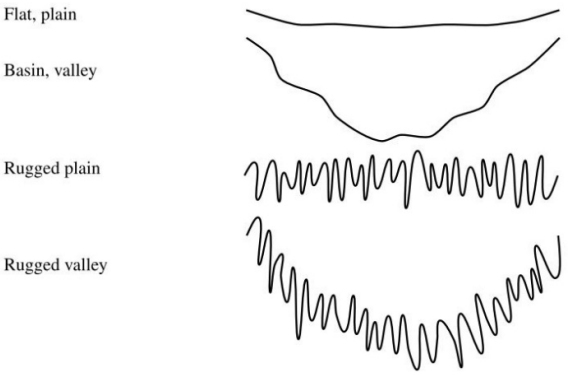
Case 1: A search space with high Δ_{ent} and high Δ_{Dmm} is called *One-Massif* or *Massif Central*, meaning that most local optimal solutions are concentrated in a dense and small region, as shown in Fig. 8(a), first row. Therefore, using a single-solution metaheuristic to search the space is more suitable than using a population-based metaheuristic because the latter would waste more time searching for relatively sparse areas of the space.

Case 2: A search space with high Δ_{ent} and low Δ_{Dmm} is called *Multi-massif*, meaning that local optima are localized in several attraction areas, as shown in Fig. 8(a), second row. Therefore, population-based metaheuristics are better choices than single-solution-based metaheuristics because the latter cannot sufficiently explore various solution clusters scattered across the search space.

Case 3: A search space with low Δ_{ent} and low Δ_{Dmm} is called *Uniform*, meaning that local optima are scattered rather evenly in the search space, as shown in Fig. 8(a), third row. Therefore, starting from a random initial solution, the local search converges rapidly to a nearby local optimum.

Δ_{ent}	Δ_{Dmm}	Local Optima Distribution
High	High	 One-massif
High	Low	 Multi-massif
Low	Low	 Uniform

(a)



(b)

Fig. 8. (a) Distribution of local optima in the landscape affected by Δ_{ent} and Δ_{Dmm}; (b) Four types of search space (Talbi, 2009).

Based on the computational results in Table 3, we plotted the scatterplot of average Δ_{ent} vs. average Δ_{Dmm} for all 11 benchmark problem sets in Fig. 9, in which in addition to the benchmark sets, in which the total average with $\bar{\Delta}_{Dmm} = 0.068$ and $\bar{\Delta}_{ent} = 0.014$ is shown as a larger red blob. Since both $\bar{\Delta}_{Dmm}$ and $\bar{\Delta}_{ent}$ are small, 'Case 3' above is established, suggesting that overall, the local optima in SALBP1 are generally ***uniformly*** scattered over the search space.

Further analyses specific to certain problem sets or instances therein are provided below:

i. The relatively higher average value of Δ_{Dmm} for the Mitchel test problem (0.3136), especially for $c = 14$ (instance no. 37 in Table 3, with $\Delta_{Dmm} = 0.8890$) indicates that its local optima are located far from each other over the search space. On the other hand, since the average value of Δ_{ent} for this test problem is almost close to the $\bar{\Delta}_{ent} = 0.014$, it can be concluded that the local optimal solutions of this problem are uniformly distributed in its search space. But their distance of solutions in this space is on average greater than the distance of local optimal solutions of other sample problems.

ii. The relatively high average values of Δ_{Dmm} and Δ_{ent} for the Jackson test problem (0.2097 and 0.1763, respectively) indicate that, unlike the other 10 test problems, 'Case 1' above is true for this problem, meaning that the distribution of the local optima over the search space of this problem is *One-massif*.

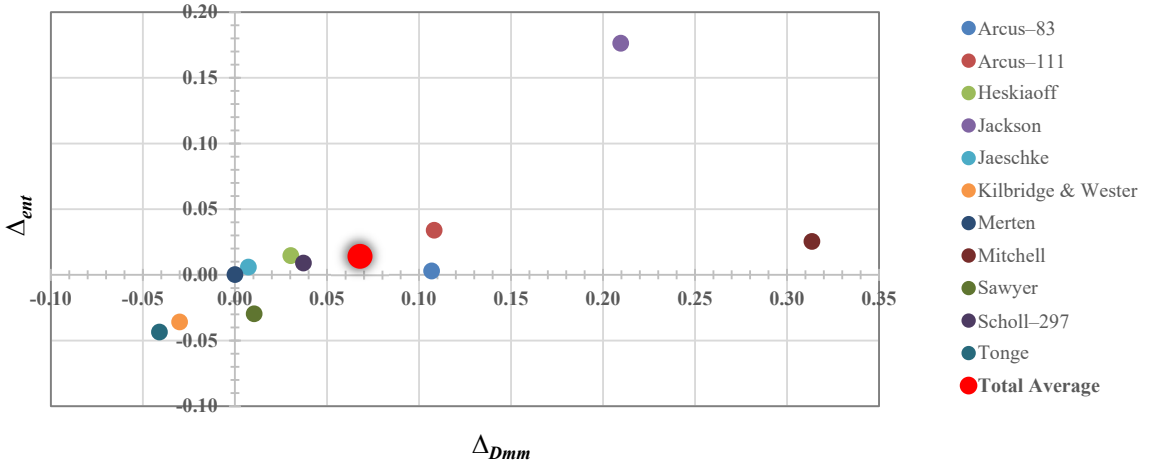


Fig. 9. Average values of Δ_{ent} and Δ_{Dmm} for the studied 11 SALBP1 benchmark problem sets (blue dots), with their total average shown as a large red blob (shadowed).

iii. The Δ_{Amp} measure is another important criterion that describes the *objective space*. The total average value of $\bar{\Delta}_{Amp} = -0.144$ for all problem sets implies that the landscape is almost *plain* or *rugged plain* (as shown in Fig. 8(b)), and there is no clear difference between the qualities of the local optimal solutions in different areas of the search space.

iv. As shown in Fig. 10, the Arcus-83, Arcus-111, and Tonge problem sets have unusually larger Δ_{Amp} values (−0.334, −0.421, and −0.252, respectively) compared to the other benchmark sets, suggesting that their landscapes are somewhat like a *valley*, and the qualities of obtained local optimal solutions were clearly better than the qualities of their starting random solutions.

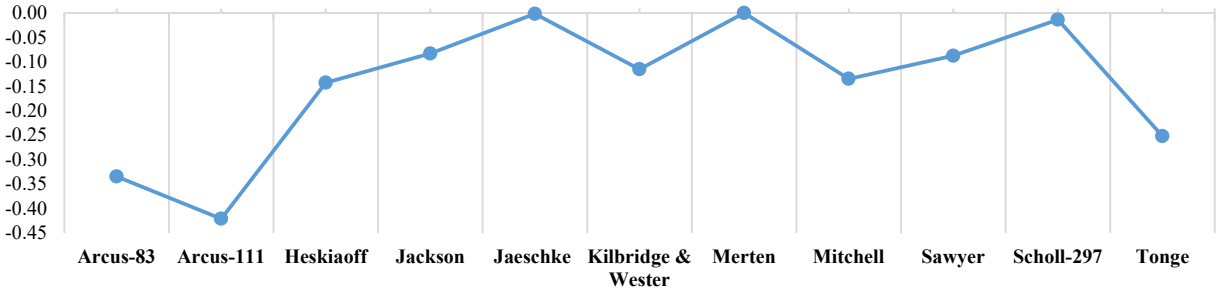


Fig. 10. Average amplitude variations (Δ_{Amp}) between the populations of initial (U) and locally optimal (O) solutions.

v. While most Δ_{amp} values are negative in Table 3, there are few positive values in the Scholl–297 dataset, indicating that the solution quality gap, i.e., $f(s^{best}) - f(s^{worst})$, and the diversity of the initial population were larger than those of the optimal population. This means that the local search was able to effectively converge the starting solutions toward relatively comparable local optimal solutions.

vi. Across all benchmark problems, the relatively large total average gap between the global (or best-known) solutions and the obtained local optimal solutions ($\overline{Gap}(O) = 1.605 = 160.5\%$), as shown by the red blob in Fig. 11, indicates that global optimum solutions in the SALBP1 problem are hard to find, and basic local search methods are unlikely to discover even near-optimal solutions efficiently. In particular, the Arcus-111, Kilbridge & Wester, and Heskiaoff problem sets have unusually large gaps (281.8%, 395.7%, and 540.1%, respectively), and hence, are more difficult to be solved to optimality compared to other benchmark sets.

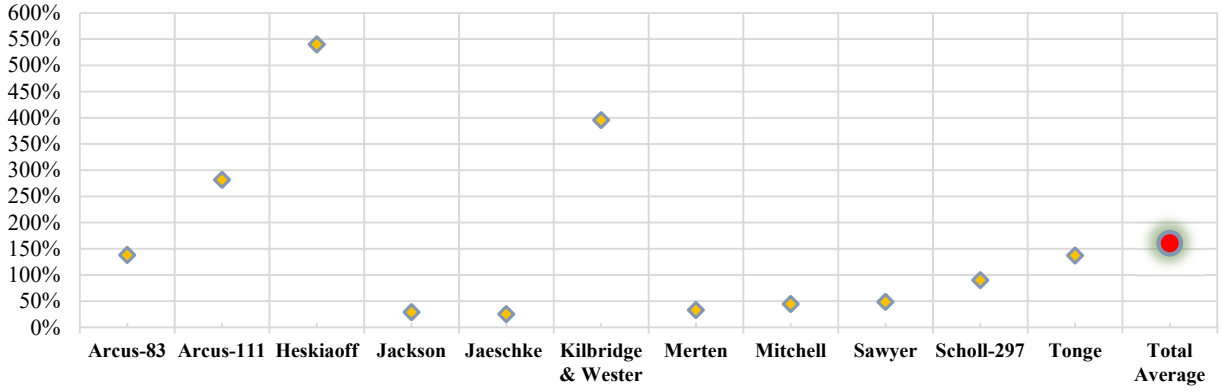


Fig. 11. Average gaps ($Gap(O)$) between the obtained local optimal solutions and global or best-known solutions of the solved 11 SALBP1 benchmark problem sets (blue asterisks), together with the total average gap over all the problems (the red asterisk).

vii. The two datasets Mertens and Jaeschke, which have the smallest number of tasks (7 and 9, respectively), stand out among other problem sets due to their near-zero average Δ_{Dmm}, Δ_{ent}, and Δ_{ent}. The average gaps ($\overline{Gap}(O)$) of these two datasets are also the smallest among all problem sets. The mentioned measures are so low because their populations of initial and optimal solutions (U and O) are not substantially different; meaning that the initial solutions generated by the shallow and deep task assignment procedures were already good enough to be close to local optimal solutions. This may bring up the question that how it is possible to produce such high-quality initial solutions even before improving them through a local search. This is because of the small number of tasks in those datasets that lead to smaller solution spaces compared to the huge search spaces of other datasets. In addition, the deep task assignment procedure guarantees the generation of feasible solutions with their $PUPC = 0$ in the fitness function (4), which are naturally of good quality (i.e., have small fitness values). As a result, we do not recommend these two datasets be included in future studies of SALBP1 due to their simplicity and lack of challenge in solving.

6. Analysis of Correlation Measures

The average *length of the walk* measure (*Lmm*) provides an estimate of the landscape's ruggedness and the correlation between the distances of solutions to the best-known or global optimum solution and their quality gap. A small value of *Lmm* implies that a short walk (or number of steps) is required to reach a nearby local optimum from a random solution; hence, the landscape

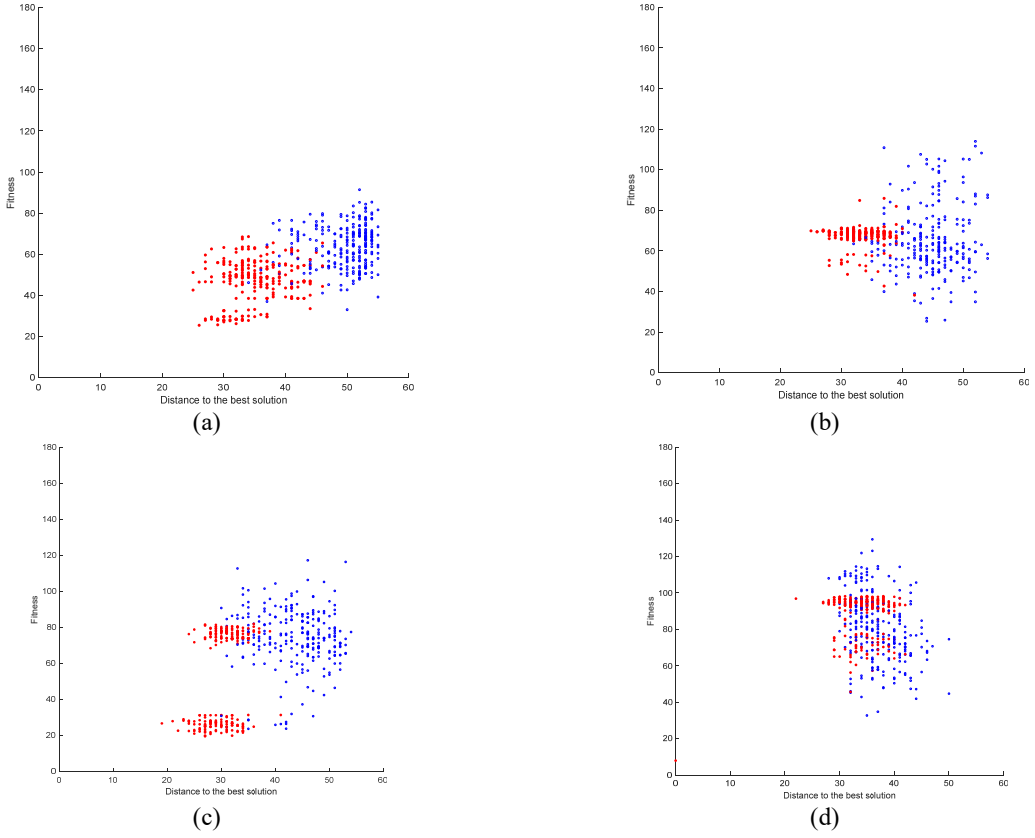
is rugged. The value of *Lmm* depends on the test problem and its cycle time. For example, analyzing the computational results in the Table 2 reveals that the landscape of the Mertens test problem (with *Lmm* = 0.01) seems to be more rugged than other problems, and hence converges faster to a local optimum solution.

In addition, we calculated the *autocorrelation* measure for three Hamming distances of $d_H = \{2, 4, 6\}$, which yielded almost equal average values for $\rho(2)$, $\rho(4)$, and $\rho(6)$, as shown in the last row and the rightmost columns of Table 3. This implies that variations in the objective values of any pair of solutions are not sensitive to their distance from each other; hence, the landscape is *rugged*. Of course, it should be noted that the Mitchel test problem is an exception to this rule, and due to the clear difference between the values of $\rho(2)$, $\rho(4)$, and $\rho(6)$, the search space for this problem is flat.

Finally, we calculated the total average FDC value of all benchmark problems (= 0.594). This value is not sufficiently large to conclude that the problem is of the 'misleading' type, implying that the move operator will not guide the search near the global optimum. Among the studied problems, the Heskiaoff problem set had the smallest average FDC value (=0.20), implying that finding its global optimal solution is more difficult than for other problems. To analyze the Heskiaoff problem in more depth, we plotted the FDC diagrams of 1000 local optimal solutions for six different cycle times: c = 138, 205, 216, 256, 324, and 342. Fig. 12 presents the obtained FDC plots, in which the hollow (blue) circles indicate infeasible solutions and solid (red) dots show feasible solutions. As stated earlier, the maximum Hamming distance between any two solution encodings of an n-task assembly line is $2n$, and so for the Heskiaoff test problem with 28 assembly tasks, the maximum value on the x-axis in FDC plots will be 56. For this test problem, the FDC value is very small (even negative for scenarios with cycle times of 205, 256, and 342), which shows that there is no correlation between the distance of the solutions to the best-known solution and their fitness. Therefore, achieving a global optimal solution for this test problem is difficult and requires an effective search algorithm.

Analyzing the FDC plots in Fig. 12 shows that:

- Almost all the distances of the local optimum solutions to the best-known solution are within the interval [18, 55], indicating that the FDC is weak; therefore, the problem is difficult to solve. Moreover, in general, feasible solutions have shorter Hamming distances to the best-known solution than to infeasible solutions.
- Feasible (red) and infeasible (blue) solutions overlap at much fitness–distance points. This indicates that discriminating infeasible solutions from feasible solutions is difficult or impossible. It was noticed that in some cases (especially in Fig. 12(c) and (f)), feasible solutions had two concentration areas, implying that those with the same distance to the best-known solution had different fitness values, adding more to the complexity.



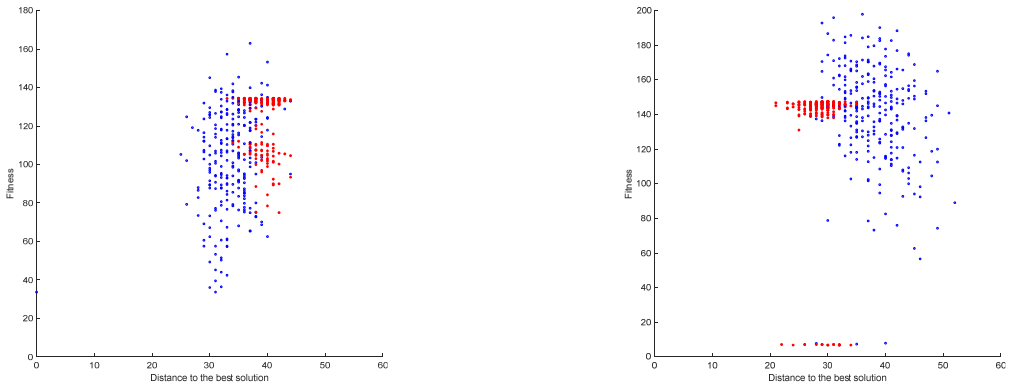


Fig. 12. FDC plots for 1000 local optimal solutions of the Heskiaoff test problem classified according to different cycle times: (a) c = 137, (b) c = 205, (c) c = 216, (d) c = 256, (e) c = 324, and (f) c = 342. Blue circles indicate infeasible, and red dots show feasible solutions.

As a conclusion to the FLA, it can be stated that the fitness landscape of the SALBP1 is a rugged plain (as shown in Fig. 8(b)) with uniformly distributed local optima and no apparent correlation between the quality and distance of solutions to the global optimum. Therefore, a single solution-based metaheuristic algorithm is more suitable for solving this problem.

7. Conclusion

Unlike its name, the Simple Assembly Line Balancing Problem Type 1 is not a simple problem. An optimal solution must satisfy the precedence relations among all tasks and assign as many tasks as possible in as few assembly workstations as possible, such that the total task time of each workstation does not exceed the given cycle time and the workloads of the stations are as even (balanced) as possible. The problem is proven to be NP-hard, and thus, finding globally optimal solutions for large instances is practically prohibitive.

Despite the considerable amount of work on solving SALBP1, there is no methodical study of the structure and landscape of the problem; hence, choices of using (meta)heuristic solution methods have mostly been arbitrary and lacking strong justification. In this paper, for the first time, we present a comprehensive and in-depth Fitness Landscape Analysis to understand the structure and topology of the solution space of SALBP1. The analysis was carried out using seven distribution and three correlation measures for 83 instances taken from 11 benchmark problem sets known in the literature, and revealed that the problem's landscape is more like a rugged plain with local optimums scattered all over the search space uniformly, suggesting that a single-solution-based metaheuristic method would be more successful in searching the space and finding an optimal or near-optimal solution than a population-based metaheuristic.

It is noted that since various benchmark problem sets with different number of tasks, precedence relations, and cycle times were used, the outcome of the presented FLA adequately covers the SALBP1 in its totality, and therefore can be safely used to describe the fitness landscape of the problem. Also, the new formula developed for measuring the entropy of solutions in the search space of the SALBP1 is another novelty of the present paper.

FLA has seldom been performed for assembly line balancing problems; thus, as a future research direction, we propose conducting such analyses for other types and sizes of SALB problems such as SALBP2, SALBP−E, SALBP−F, as well as for GALB problems including 2S–ALB, MALB, and UALB.

References

Abdeljaouad, M. A., & Klement, N. (2021). Tabu search algorithm for single and multi-model line Balancing problems. *In Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems: IFIP WG 5.7 International Conference*, APMS 2021, Nantes, France, September 5–9, 2021, Proceedings, Part I (pp. 409-415). Springer International Publishing.

Álvarez-Miranda, E., Pereira, J., Torrez-Meruvia, H., & Vilà, M. (2021). A Hybrid Genetic Algorithm for the Simple Assembly Line Balancing Problem with a Fixed Number of Workstations. *Mathematics, 9*(17), 2157.

Álvarez-Miranda, E., Pereira, J., Vargas, C., & Vilà, M. (2022). Variable-depth local search heuristic for assembly line balancing problems. *International Journal of Production Research*, *61*(9), 3103-3121.

Baskar, A., & Xavior, M. A. (2020). Heuristics based on slope indices for simple type I assembly line balancing problems and analyzing for a few performance measures. *Materials Today: Proceedings, 22*, 3171-3180.

Bautista, J., & Pereira, J. (2002, August). Ant algorithms for assembly line balancing. In *International Workshop on Ant Algorithms* (pp. 65-75). Berlin, Heidelberg: Springer Berlin Heidelberg.

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

Baykasoglu, A. (2006). Multi-rule multi-objective simulated annealing algorithm for straight and U type assembly line balancing problems. *Journal of Intelligent Manufacturing, 17*(2), 217-232.

Boysen, N., Schulze, P., & Scholl, A. (2022). Assembly line balancing: What happened in the last fifteen years?. *European Journal of Operational Research*, *301*(3), 797-814.

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

Chutima, P. (2022). A comprehensive review of robotic assembly line balancing problem. *Journal of Intelligent Manufacturing, 33*(1), 1-34.

Dolgui, A., & Gafarov, E. (2019). Can a Branch and Bound algorithm solve all instances of SALBP-1 efficiently? *IFAC-PapersOnLine, 52*(13), 2788-2791.

Dou, J., Li, J., & Zhao, X. (2017). A novel discrete particle swarm algorithm for assembly line balancing problems. *Assembly Automation*, *37*(4), 452-463.

Fathi, M., Fontes, D. B. M. M., Urenda Moris, M., & Ghobakhloo, M. (2018). Assembly line balancing problem: A comparative evaluation of heuristics and a computational assessment of objectives. *Journal of Modelling in Management, 13*(2), 455-474.

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

Ghandi, S., & Masehian, E. (2015). Review and taxonomies of assembly and disassembly path planning problems and approaches. *Computer-Aided Design*, 67, 58-86.

Gonçalves, J. F., & De Almeida, J. R. (2002). A hybrid genetic algorithm for assembly line balancing. *Journal of heuristics, 8*, 629-642.

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

Helgeson, W., & Birnie, D. P. (1961). Assembly line balancing using the ranked positional weight technique. *Journal of industrial engineering, 12*(6), 394-398.

Hong, D. S., & Cho, H. S. (1999, October). Generation of robotic assembly sequences using a simulated annealing. In *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients (Cat. No. 99CH36289)* (Vol. 2, pp. 1247-1252). IEEE.

Hu, Y., Liu, C., Zhang, M., Jia, Y., & Xu, Y. (2023). A novel simulated annealing-based hyper-heuristic algorithm for stochastic parallel disassembly line balancing in smart remanufacturing. *Sensors, 23*(3), 1652.

Kilincci, O. (2011). Firing sequences backward algorithm for simple assembly line balancing problem of type 1. *Computers & Industrial Engineering,* 60(4), 830-839.

Li, Z., Janardhanan, M. N., Nielsen, P., & Tang, Q. (2018). Mathematical models and simulated annealing algorithms for the robotic assembly line balancing problem. *Assembly Automation*, *38*(4), 420-436.

Meng, K., Tang, Q., Zhang, Z., & Yu, C. (2021). Solving multi-objective model of assembly line balancing considering preventive maintenance scenarios using heuristic and grey wolf optimizer algorithm. *Engineering applications of artificial intelligence, 100*, 104183.

Mohammed, F. D., Zakaria, M. Z., Ramli, M. F., Jusoh, M., Azizan, M., & Fadzli, N. (2021, May). Metaheuristic optimization in solving assembly line balancing problems: A short review. In *AIP Conference Proceedings* (Vol. 2339, No. 1). AIP Publishing.

Nagy, L., Ruppert, T., & Abonyi, J. (2020). Analytic hierarchy process and multilayer network-based method for assembly line balancing. *Applied Sciences, 10*(11), 3932.

Nourmohammadi, A., Fathi, M., & Ng, A. H. (2019). Choosing efficient meta-heuristics to solve the assembly line balancing problem: A landscape analysis approach. *Procedia CIRP*, *81*, 1248-1253.

Özbakır, L., & Seçme, G. (2022). A hyper-heuristic approach for stochastic parallel assembly line balancing problems with equipment costs. *Operational Research*, 1-38.

Pape, T. (2015). Heuristics and lower bounds for the simple assembly line balancing problem type 1: Overview, computational tests and improvements. *European Journal of Operational Research, 240*(1), 32-42.

Pastor, R., & Ferrer, L. (2009). An improved mathematical program to solve the simple assembly line balancing problem. *International Journal of Production Research, 47*(11), 2943-2959.

Ponnambalam, S., Aravindan, P., & Naidu, G. M. (2000). A multi-objective genetic algorithm for solving assembly line balancing problem. *The International Journal of Advanced Manufacturing Technology, 16*(5), 341-352.

Rashid, M. F. F., Hutabarat, W., & 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.

Ravelo, S. V. (2022). Approximation algorithms for simple assembly line balancing problems. *Journal of Combinatorial Optimization, 43*(2), 432-443.

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

Scholl, A., & Voß, S. (1997). Simple assembly line balancing—Heuristic approaches. *Journal of Heuristics, 2*, 217-244.

Seçme, G., & Özbakır, L. (2019). An assembly line balancing application on oven production line with hyper-heuristics. *International Journal of Operations Research and Information Systems (IJORIS), 10*(3), 44-58.

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

Talbot, F. B., Patterson, J. H., & Gehrlein, W. V. (1986). A comparative evaluation of heuristic line balancing techniques. *Management science*, *32*(4), 430-454.

Vilà, M., & Pereira, J. (2013). An enumeration procedure for the assembly line balancing problem based on branching by non-decreasing idle time. *European Journal of Operational Research, 229*(1), 106-113.

Wei, N.-C., & Chao, I.-M. (2011). A solution procedure for type E simple assembly line balancing problem. *Computers & Industrial Engineering, 61*(3), 824-830.

Yadav, A., Kulhary, R., Nishad, R., & Agrawal, S. (2020). Parallel two-sided assembly line balancing with tools and tasks sharing. *Assembly Automation*, *40*(6), 833-846.

Zhang, H. Y. (2019). An immune genetic algorithm for simple assembly line balancing problem of type 1. *Assembly Automation*, 39(1), 113-123.

Zhong, Y.-g., & Ai, B. (2017). A modified ant colony optimization algorithm for multi-objective assembly line balancing. *Soft Computing, 21*(22), 6881-6894.