<http://www.diva-portal.org>

Postprint

This is the accepted version of a paper published in *Engineering computations*. This paper has been peer-reviewed but does not include the final publisher proof-corrections or journal pagination.

Citation for the original published paper (version of record):

Fathi, M., Nourmohammadi, A., Ng, A H., Syberfeldt, A., Eskandari, H. (2020)
An improved genetic algorithm with variable neighborhood search to solve the
assembly line balancing problem
Engineering computations, 37(2): 501-521
<https://doi.org/10.1108/EC-02-2019-0053>

Access to the published version may require subscription.

N.B. When citing this work, cite the original published paper.

<https://creativecommons.org/licenses/by-nc/4.0/>

Permanent link to this version:

<http://urn.kb.se/resolve?urn=urn:nbn:se:his:diva-17157>

An improved genetic algorithm with variable neighborhood search to solve the assembly line balancing problem

Journal:	<i>Engineering Computations</i>
Manuscript ID	EC-02-2019-0053.R2
Manuscript Type:	Research Article
Keywords:	Assembly line balancing, Genetic algorithm, Variable neighborhood search, Generation transfer

SCHOLARONE™
Manuscripts

An improved genetic algorithm with variable neighborhood search to solve the assembly line balancing problem

Abstract

- **Purpose** – This study aims to propose an efficient optimization algorithm to solve the assembly line balancing problem (ALBP). The ALBP arises in high-volume, lean production systems when decision makers aim to design an efficient assembly line while satisfying a set of constraints.
- **Design/methodology/approach** – An improved genetic algorithm (IGA) is proposed in this study to deal with ALBP in order to optimize the number of stations and the workload smoothness.
- **Findings** – To evaluate the performance of the IGA, it is used to solve a set of well-known benchmark problems and a real-life problem faced by an automobile manufacturer. The solutions obtained are compared against two existing algorithms in the literature and the basic genetic algorithm. The comparisons show the high efficiency and effectiveness of the IGA in dealing with ALBPs.
- **Originality/value** – The proposed IGA benefits from a novel generation transfer mechanism that improves the diversification capability of the algorithm by allowing population transfer between different generations. In addition, an effective variable neighborhood search is employed in the IGA to enhance its local search capability.

Keywords: assembly line balancing, genetic algorithm, variable neighborhood search, generation transfer.

1. Introduction

The task of balancing assembly lines is a key issue for manufacturing companies committed to high-volume mass production in a highly competitive business environment. Identifying the most efficient balance of assembly lines has therefore attracted considerable attention in recent decades. Attempts to solve this problem have resulted in the introduction of what is known as the assembly line balancing problem (ALBP).

The ALBP deals with the assignment of assembly tasks to assembly stations in the presence of certain constraints, with the goal of improving some performance measures. According to the literature (e.g., Nourmohammadi and Eskandari, 2017; Pereira and Álvarez-Miranda, 2018), ALBPs are generally classified into two groups on the basis of their assumptions, constraints and objectives. The two groups are (1) the simple assembly line balancing problem (SALBP) where a single model of a product is produced, task times are deterministic and the assembly line is straight and (2) the general assembly line balancing problem (GALBP) which covers all problems that do not fall within SALBP. GALBPs have more restrictions or factors such as parallel stations, stochastic task times and zoning constraint (Fathi et al., 2019).

The SALBP is the most commonly studied problem in the assembly line balancing context and the majority of research on ALBPs has concentrated on solving it (Fathi et al., 2018). SALB problems are divided into four groups based on their objectives (Zhang, 2017). The groups are

(1) SALBP-1 to minimize the number of stations (NS) while cycle time (CT) is fixed, (2) SALBP-2 to minimize the CT while NS is fixed, (3) SALBP-E to minimize the CT and the NS simultaneously and (4) SALBP-F, which is a feasibility problem where all parameters are fixed.

SALBP-1 has received the most attention from researchers and industrial practitioners (Fathi, et al., 2016a; Nourmohammadi et al., 2019). Over the past decade, researchers have proposed several exact solution methods to tackle SALBP-1. However, for real-world/large-scale problems, these exact methods are not efficient enough in terms of computational time and computational cost due to the complex nature of the problem, which has been proven to be NP-hard (Zacharia and Nearchou, 2016). Thus, there has been a growing trend toward using metaheuristic algorithms to overcome the computational complexity while still providing quality solutions.

The literature agrees that metaheuristic algorithms provide good results by intelligent exploration of the fitness landscape that is defined by the objective function (Aledo et al., 2017). Therefore, choosing the right objective function can guide the algorithm toward promising search space that may lead to high-quality solutions. According to Chutima and Chimklai (2012), two groups of objectives can be defined for SALBs, namely workload smoothness-related objectives and efficiency-related objectives. Although the conventional SALBP-1 is a single objective problem and aims to minimize the NS , the literature has shown that this efficiency-related objective cannot be an operational measure for guiding a metaheuristic to find a good solution. The reason is that the objective function contains large plateaus and it is very common to find solutions with different levels of stations' workload smoothness but the same NS (Azizoğlu and İmat, 2018). In such circumstances, it is desirable to use a secondary objective to evaluate different solutions with the same NS but different workloads.

In line with research trends and to treat the shortcomings of the classic SALBP-1 in terms of objectives, this study proposes an improved genetic algorithm (IGA) where a secondary objective, namely workload smoothness, is to be optimized beside the prime objective of SALBP-1 (i.e., minimizing the NS). The distinctive features of the proposed algorithm are the incorporation of an effective variable neighborhood search (VNS) to improve the intensification of the algorithm and a novel generation transfer (GT) strategy that allows the transfer of the best individuals between different generations to improve the diversification capability of the algorithm as the evolution progresses.

The remainder of the paper is organized as follows. Section 2 provides a literature review of the multi-objective SALBP. A description of the problem is provided in Section 3. The proposed IGA is presented in Section 4. The computational study is presented in Section 5. Concluding remarks are made in Section 6.

2. Literature review

This section reviews the studies most relevant to this research, that is, studies on the multi-objective SALBP in which heuristic/metaheuristic algorithm has been applied. The focus of the review is on the algorithms employed and the optimization objectives considered. Readers interested in knowing more about the history of other types of ALBPs are referred to the

comprehensive reviews of Battaia and Dolgui (2013), Sivasankaran and Shahabudeen (2014), Li et al. (2017) and Razali et al. (2019).

Ponnambalam et al. (2000) suggested a genetic algorithm (GA) and incorporated a trade and transfer mechanism to improve the quality of solutions. The performance of the algorithm was compared against some heuristics such as ranked positional weight, Kilbridge and Wester, Moodie and Young and etc. The optimization criteria considered were the *NS* and the smoothness index (*SI*). Liu et al. (2003) proposed two heuristic algorithms based on a bi-directional assignment procedure. The objectives considered were to minimize the deviation from the optimal or lower bound of *CT* and the mean absolute deviation (*MAD*) of the stations' workload (i.e., a measure of the stations' workload smoothness). Baykasoğlu (2006) developed a simulated annealing (SA) method combined with several priority heuristic rules to balance straight and U-shaped lines. Following the goal programming approach, this study aimed to minimize the *NS* and the *SI* as a measure of workload smoothness between the stations. Nearchou (2008) suggested a population-based metaheuristic, adapting the differential evolution (DE) algorithm to cope with the problem. Two variants of objective functions were considered in the study: (1) minimizing the *CT* and the balance delay (*BD*) (which amounts to the total idle time of the stations) and (2) minimizing *CT* and *SI*. Özcan and Toklu (2009) proposed a hybrid improvement heuristic (HIH) based on adaptive learning and SA to balance straight and U-shaped lines. They considered minimization of the *NS*, the variations of workload (*VW*) among stations and *SI* as the optimization criteria.

Yu and Yin (2010) tackled the problem by developing an adaptive genetic algorithm (AGA) in which the crossover and mutation rates were dynamically adjusted based on the fitness value of each individual. The objectives considered in this study were to minimize the *NS* and *SI*. Petropoulos and Nearchou (2011) proposed a particle swarm optimization (PSO) algorithm in which two variants of the objectives, (1) *CT* and *MAD* and (2) *CT*, *MAD* and *BD*, were to be optimized. The authors proposed a new representation scheme tailored to the algorithm to generate feasible solutions for the problem. Nearchou (2011) proposed a PSO algorithm by considering the minimization of *CT* and *MAD* of workloads as the optimization objectives. The author improved the algorithm by introducing a new elitism strategy and applying a novel dynamic weighting method. Nourmohammadi and Zandieh (2011) suggested a DE algorithm where the technique for order preference by similarity to the ideal solution was applied to rank the solutions in each population in terms of the *CT* and *SI*. The authors also took advantage of a new procedure for solution acceptance to build a good pool of candidate solutions. De Albuquerque et al. (2016) proposed a fish school search (FSS) algorithm and incorporated some improvement guidelines to ameliorate the search process and avoid trapping in local optima. The authors considered a combination of *NS* and smoothness of line (*SOL*) as the objective function. The performance of the algorithm was tested against an exact solution method and PSO algorithm by solving a set of test problems taken from the literature. Dou et al. (2017) proposed a modified PSO algorithm featuring a new mechanism for particles to enhance the algorithm search ability. The optimization objectives considered in this study were the *NS* and *SI*. The authors benchmarked their PSO algorithm against other existing variants of the same algorithm. Zhang (2017) addressed the problem by proposing an improved immune algorithm (IA) based on a new procedure that enables the algorithm to escape from local

optima. The objectives considered in this study were to minimize the *NS* and *SI*. The author proposed a new formula to simplify the calculation of similarity between the antibodies. Zhong and Ai (2017) proposed a modified ant colony optimization (ACO) algorithm to minimize *NS*, *CT* and variation of workload (*VW*). They embedded a new task assignment procedure guided by heuristic information to improve the efficiency of the ant colony search for the optimal solution. Lalaoui and Afia (2018) proposed an SA algorithm embedded with the fuzzy inference system. The objectives considered were to minimize the *NS* and *SOL*. The performance of the algorithm was tested by solving a set of test problems and comparing the results with an exact solution method. Zhang (2019) applied a hybrid GA (HGA) by combining the IA and GA to avoid the stagnation of the search in local optima. The *NS* and *SI* were used as the optimization objectives.

Table 1 summarizes the above reviewed literature and shows the optimization algorithm used and the objectives considered in each study.

Table 1. Summary of the reviewed studies

Study	Objective function							Optimization algorithm									
	<i>NS</i>	<i>CT</i>	<i>SI</i>	<i>VW</i>	<i>SOL</i>	<i>BD</i>	<i>MAD</i>	GA	ACO	PSO	HH	SA	DE	IA	HGA	OHA	FSS
Ponnambalam et al. (2000)	•		•**						•								
Liu et al. (2003)		•					•										•
Baykasoğlu (2006)	•		•									•					
Nearchou (2008)		•	•				•						•				
Özcan and Toklu (2009)	•		•	•							•						
Yu and Yin (2010)	•		•*						•								
Petropoulos and Nearchou (2011)		•				•	•				•						
Nearchou (2011)		•					•				•						
Nourmohammadi and Zandieh (2011)		•	•										•				
De Albuquerque et al., (2016)	•				•												•
Dou et al. (2017)	•		•*								•						
Zhang (2017)	•		•*												•		
Zhong and Ai (2017)	•	•		•					•								
Lalaoui and Afia, (2018)	•				•							•					
Zhang (2019)	•		•**												•		

*Only a case study or an example is considered; **The detailed results for *SI* are not reported.

Both the literature review and the information in Table 1 indicate that either *NS* or *CT* has always been the primary objective in SALBP studies. It is safe to conclude that maximizing the workload smoothness of the stations has been the secondary optimization criterion in all the reviewed studies, although this objective was achieved in different ways and through different objective functions. Table 1 also suggests that *SI* is the most popular and frequently used objective to improve workload smoothness in SALBP. Thus the current study also aims to optimize the *NS* and *SI*, as they are the most popular SALBPs objectives.

The literature review shows that the main research focus has been on improving and tailoring metaheuristic algorithms to solve SALBP. The algorithms have been improved to cope with problem complexity more efficiently and provide quality solutions while shortening the computational time. Table 1 also shows that, although some algorithms already exist for solving the SALBP with the objective of minimizing the *NS* and *SI*, their performance and efficiency are not properly tested through solving standard test problems and benchmarking against other existing algorithms. Moreover, the detailed results including the obtained value for each objective, the used computational time for each problem and the statistical details of

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60

the proposed algorithms are not reported in many of the previous studies. All of these, hindered industrial practitioners and researchers to wisely choose the most competent algorithm among the existing ones in terms of quality of solutions and computational time. In these circumstances, the main aim of this study is to propose a trustworthy algorithm that can be used to efficiently solve the SALBPs. Given the promising performance of GAs in dealing with a variety of combinatorial optimization problems (Baniamerian et al., 2018), an improved GA (IGA) is proposed in this study. The IGA benefits from a GT mechanism improving its diversification and a VNS enhancing its intensification.

The performance of the IGA is evaluated by solving existing standard benchmark problems and comparing the solutions to the basic GA (without VNS and GT) as well as to the two most relevant and comparable algorithms proposed by Özcan and Toklu (2009) and Baykasoğlu (2006). Although more studies, shown in Table 1, have been performed with the same overall objectives as the current study, it is difficult to perform a direct comparison due to these other studies not having solved the standard benchmark problems or reporting detailed results.

3. Problem description

The SALBP-1 considered in this study can be described as follows. A single-model of a specific product is assembled in a set of stations sequentially ordered on a straight line represented as $k = \{1, 2, \dots, NS\}$. The total work to be performed to assemble a product is divided into smaller operations known as tasks, which are represented as $i = \{1, 2, \dots, N\}$, where N is the number of tasks. Performing each task requires a certain time known as the task-processing time, t_i . Due to the existing organizational, operational and technological requirements, some restricting relationships between tasks, known as precedence relationships, are defined. Based on the precedence relationships, each task i can only be assigned to station k if and only if all of its predecessors are assigned to the same or prior stations.

A task cannot be assigned to more than one station, but each station can take up more than one task, depending on its available time capacity. The sum of tasks' times assigned to each station is known as the station time. The station time must not exceed a fixed time known as CT (i.e., the time required to deliver one complete product). At the end of the CT , each assembly object is transferred to the next station. At that time a new object enters the first station and a completed object leaves the last station. Considering the problem description above, the considered SALBP-1 aims to find a feasible combination of tasks to be performed at each station with the primary goal of minimizing the total number of required stations and secondarily of maximizing the workload smoothness at the stations. To better understand the problem, including its constraints and objectives, the mathematical model of the considered SALBP-1 is presented in the [supplementary document](#).

4. The proposed improved genetic algorithm

GAs are versatile search algorithms inspired by the natural selection mechanism and are used to address combinatorial optimization problems. A GA relies on a set of individuals or solutions known as chromosomes, which undergo a set of operations, namely selection, mutation and crossover, to generate the next population members (Rezaie et al., 2019). Throughout the

generations, a GA searches different areas of the search space to find high-quality solutions while avoiding local optima. To improve the search capability of a GA, it has to be appropriately tailored to the characteristics of the considered problem. This can be done by embedding proper search mechanisms to enhance the efficiency of the GA search process.

This study proposes an improved GA (IGA) that incorporates novel search mechanisms tailored to the characteristics of the SALBP-1. It has three main features (1) specific encoding and decoding procedures that prevent the generation of infeasible solutions, (2) hybridization using VNS that improves the GA's intensification (local search) capability within each generation and (3) introduction and application of a GT mechanism that allows the transfer of a portion of the best individuals from future generations back to previous generations so as to improve the diversification of the GA throughout the evolution progress. The general framework for the IGA is presented in Figure 1. The following sections provide a detailed description of the proposed IGA.

Figure 1. The flow diagram of IGA for SALBP-1

4.1 Chromosome representation

A vector of integer numbers is assumed for representation, in which positions $1, 2, \dots, N$ (N is the number of tasks) are associated with a permutation of numbers between $[1, N]$ as the tasks' relative priority (ψ_i). This vector known as task priority vector is randomly generated at the initial generation. According to this representation shown in Figure 2, each task is given a priority and the tasks are assigned to the stations based on their priority. A task with a higher priority number must be assigned to the station before the tasks with lower priority number. However, the precedence and CT constraints also have to be considered for the assignment of tasks. To satisfy these constraints, an encoding and decoding scheme is used as presented in the next section.

Figure 2. The representation scheme

4.2 Encoding and decoding procedures

To satisfy the SALBP-1 precedence relationships, an encoding procedure is needed to map each task priority vector (presented in Figure 2) to a feasible solution. Following the general encoding procedure used in the literature (e.g., Hwang and Katayama, 2010) and assuming an N -task problem with known precedence relationships between tasks, a graph $G = (V, E)$ can be defined in which V denotes the set of vertices representing the tasks (i.e., $V = \{i \mid i = 1, \dots, N\}$) and E shows the set of edges denoting the precedence relationships between tasks. The encoding procedure aims to generate a topological sort of G shown by a vector of tasks' sequence (TS) with length N from each vector of task priorities ψ . In other words, TS shows a ranking of all tasks according to the priorities given by ψ so that the precedence constraints among tasks are satisfied. The encoding procedure is presented in pseudo-code 1.

Pseudo-code 1. Encoding procedure

```

1 input: task priority vector ( $\psi$ )
2 set  $V' = \emptyset$  ( $V' \subseteq V$ )
3 set  $V'' = V$ 
4 set  $TS = \emptyset$ 
5 repeat
6   for task  $i = 1$  to  $N$  ( $N =$  number of tasks)
7     if  $i \in V''$  and all its predecessors are included in  $TS$ 
8        $V' = V' \cup \{i\}$ , i.e., insert task  $i$  into set  $V'$ 
9     end
10  end
11  select the task in  $V'$  with the maximum priority according to  $\psi$ 
12  insert the task obtained in line 11 into next empty position in  $TS$  and remove it from  $V'$  and  $V''$ 
13 until  $TS$  is complete
14 report  $TS$ .
```

Additionally, a decoding procedure is required to assign the available tasks in the TS vector to the stations so that the resulting SALBP-1 solutions are feasible considering the maximum allowed station time (CT). The decoding procedure ensures that tasks are assigned to stations, if and only if for the set of tasks assigned to station k , shown by S_k , the cumulative sum of task times assigned to station k , shown by $t(S_k) = \sum_{i \in S_k} t_i$, does not exceed the CT . When the station's time limit is reached, a new station will be opened. The assignment procedure is continued until all tasks in the TS vector are assigned to stations. The decoding procedure is outlined in pseudo-code 2.

Pseudo-code 2. Decoding procedure	
1	Input: CT, TS , task times
2	$k = 1$; (the first station is opened)
3	repeat
4	for $\vartheta = 1$ to N
5	pick the task positioned in the ϑ th element of TS vector (TS_{ϑ})
6	if $TS_{\vartheta} \neq 0$ and $t(S_k) + t(TS_{\vartheta}) \leq CT$ and all the predecessors of task positioned in TS_{ϑ} are already assigned; ($t(TS_{\vartheta})$ = time of task positioned in ϑ th element of TS vector)
7	assign the relating task to station k and remove it from TS by putting $TS_{\vartheta} = 0$
8	$t(S_k) = t(S_k) + t(TS_{\vartheta})$
9	end
10	end
11	$k = k + 1$
12	until all the elements of TS are zero.

4.3 Fitness function evaluation

To guide the algorithm toward the optimal solution, a mechanism is needed to evaluate the fitness of each individual in the current population. Considering the targeted problem in this study (i.e., SALBP-1), the fitness of each individual in the current population has to be evaluated as regards (1) the number of station (NS) as the primary objective and (2) the smoothness index (SI) as the secondary objective. The SI is calculated as $\sqrt{\sum_{k=1}^{NS} (t(S_k) - \max(t(S_k)))^2 / (NS)}$. As presented in Equation (1), the minimum deviation method is used to combine the two mentioned objectives (Fathi et al., 2016b).

$$f = \alpha \left(\frac{NS - \min(NS)}{\max(NS) - \min(NS)} \right) + \beta \left(\frac{SI - \min(SI)}{\max(SI) - \min(SI)} \right) \quad (1)$$

where $\min(NS)$ and $\min(SI)$ are the minimum of NS and SI found so far, respectively and $\max(NS)$ and $\max(SI)$ are the maximum of NS and SI obtained to this point, respectively. The coefficients α and β show the relative importance of NS and SI , respectively. As NS is considered more important than SI , $\alpha \gg \beta$. As it is explained in the introduction section, the primary objective of SALBP-1 is to minimize the number of stations. The secondary objective is mainly used in order to distinguish the difference between solutions that are the same in terms of the number of stations but different in terms of workloads. Therefore, the secondary objective should be optimized in a hierarchy order and after the first (top priority) objective is

achieved. Thus, the coefficients α and β are chosen in a meaningful manner to give a very strong priority to the first objective. In such circumstances, the relative importance of NS and SI were chosen as $\alpha=0.9$ and $\beta=0.1$, respectively.

As it is known, GAs are traditionally used for solving maximization problems where high fitness value indicates a better solution. Therefore the ALBP, which is a minimization problem, needs to be transformed into maximization problem. To do so, the transformation proposed by Deb (1995) is used as $F = 1/(1 + f)$.

4.4 Elitism mechanism

To preserve the individuals with the best fitness function over the population evolution, an elitism mechanism is applied in the IGA. Using this mechanism, a portion of the best individuals in each generation is reproduced in the next population as determined by reproduction rate (Pr_{re}).

4.5 Crossover

Based on the chromosome representation in section 4.1, a two-point based weight mapping crossover is used in the proposed IGA. This type of crossover has proven its efficiency when the representation type of the algorithm is a permutation vector of integer numbers (Hwang and Katayama, 2010). To make sure that parents with the best fitness function will be likely to get more copies, the roulette wheel selection is applied. For further information about roulette wheel, interested readers are referred to Man et al. (1999).

The applied crossover has four main steps: (1) selecting a random sub-vector on two randomly chosen parents, (2) ranking the tasks based on their given priority in an ascending order with a lower number denoting a task with higher priority, (3) exchanging the ranks between the chosen sub-vectors and rearranging the priorities based on the new ranks, (4) generating the offspring based on the newly mapped task priorities in step (3). The crossover is performed using the crossover rate Pr_{cr} . To better understand the crossover operator applied here, an illustrative example is presented in Figure 3.

Figure 3. Crossover operator for IGA

4.6 Mutation

A swap operator is used in the proposed IGA for direct exchange of genes between two different points on a randomly chosen individual. The swap operation is performed in two steps: (1) a parent and two mutation points are selected randomly, (2) the task priority of the chosen genes are exchanged. The mutation operator is limited by a specific rate known as the mutation rate (Pr_{mu}). An illustrative example of the applied mutation operator is presented in Figure 4.

Figure 4. Mutation operator for IGA

4.7 Variable neighborhood search

VNS is usually employed in evolutionary algorithms to improve the possibility of finding better solutions and avoid getting trapped in local optima by systematically using different search structures (Zhang et al., 2018). In this study, two VNS operators, namely 3-opt and 4-opt, are adopted from the literature (e.g., Zhao et al., 2017) and used to enhance the local search (intensification) capability of the proposed algorithm. The chosen VNS operators are applied in the IGA in three main steps: (1) using the VNS rate (Pr_{vns}), a limited number of the best solutions in each population are chosen, (2) the 3-opt operation is performed by selecting three random genes of each chosen chromosome and swapping their task priorities, (3) the 4-opt operator is applied in the same way, but by selecting four random genes only if the 3-opt was not effective and has not resulted in a better solution. The VNS procedure is shown in pseudo-code 3.

Pseudo-code 3. VNS procedure	
1	counter=1
2	select a set of the best solutions (i.e., set A) from the current population according to Pr_{vns} ; $A = \{x_\delta \delta = 1, \dots, \lceil \text{population size} \times Pr_{vns} \rceil\}$
3	for $\delta = 1$ to $\lceil \text{population size} \times Pr_{vns} \rceil$
4	find a new solution (x') using 3-opt operator
5	if the new solution is better than the current one
6	$x_\delta \leftarrow x'$
7	else
8	find a new solution using 4-opt operator (x'')
9	if the new solution is better than the current one
10	$x_\delta \leftarrow x''$
11	else
12	counter=counter+1
13	if counter $\leq$ maximum number of local search iterations
14	go to line 4
15	end
16	end
17	end
18	end.

4.8 Generation transfer

The GT mechanism is inspired by the possibility of the human time travel theorem in the universe. Using the GT mechanism, some of the best individuals in future generations are allowed to transfer to previous generations. This way, the characteristics of the best individuals in later generations can be inherited by former generations. The proposed GT is a recursive procedure in which starting from the first generation (as the first origin), using a fixed interval, a future generation is chosen. Then, a portion of the best individuals of the chosen future generation are transferred to the origin generation to substitute for the worst individuals. Following the fixed interval in the proposed GT procedure, the chosen future generation for the current origin will be the next origin for the upcoming transfer.

In this study, based on some careful pilot experiments, the fixed interval (FI) for GT is set to 10 ($FI=10$) and the portion of worst individuals in the origin to be replaced by the best individuals from the future generation is set by a generation transfer rate ($Pr_{gen\ tr}$). Like any

other GA parameters, $Pr_{gen\ tr}$ is determined through parameter tuning (see section 5). The GT procedure is presented in pseudo-code 4 for better illustration.

Pseudo-code 4. GT procedure	
1	Initialize generation transfer counter (GTC); $GTC = 0$
2	$GTC = GTC + 1$
3	if $GTC = 1$
4	save the current generation as the origin generation
5	else if $GTC = FI$
6	perform GT considering $Pr_{gen\ tr}$ by transferring the best individuals in the current
7	generation as a substitute for the worst individuals in the origin generation
8	set $GTC = 0$
9	else
10	go to line 2
11	end.

To better understand the proposed algorithm and all of it features, the complete pseudo-code of the IGA is provided in [supplementary document](#).

5. Computational study

The performance of the proposed IGA in dealing with the SALBP-1 was assessed by solving some well-known SALBP-1 test problems from the literature (Scholl et al., 1995). These problems were chosen from different scales (i.e., small, medium and large problem sizes) with a wide range of CT s. A case study was also performed based on one of the assembly lines of a major Swedish automotive manufacturing company. In the studied assembly line, a specific part of a certain model of car engine is assembled. The decision makers at the company are interested in finding the optimum NS and, secondly, in maximizing the SI of the stations. Allowing for some degrees of freedom in defining the CT , four different CT s were proposed (i.e., 60, 65, 70 and 75 seconds) as the decision makers were interested in knowing the effect of CT on NS and SI . The data for the case study are given in [Table A.1](#) in the appendix.

The performance of the proposed IGA for SALBP-1 was compared against two existing algorithms proposed by Baykasoğlu (2006) and Özcan and Toklu (2009), respectively. To be able to show the effect of VNS and the GT mechanism on IGA, the results are also compared with the pure GA. For this reason, both the pure GA and the proposed IGA were coded in MATLAB and run on a PC with a Core i7 2.4 GHz processor and 8 GB of RAM.

Parameter setting is an important part of using a GA and has a significant impact on algorithm performance. Therefore, Taguchi method was applied in this study to determine the best levels of the parameters for IGA. The Taguchi method attempts to find the best combination of IGA parameters so that the algorithm will perform efficiently while being insensitive to variations in the noise factors. Thus, by applying the Taguchi experiments over all the test problems while considering their related CT as the noise factor, the best parameter levels can be chosen. To learn more about the Taguchi method, interested readers may refer to Taguchi et al. (2005). Based on the Taguchi experiments, the IGA parameter levels were set to 0.8, 0.1, 0.2 and 0.1 for Pr_{cr} , Pr_{mu} , Pr_{vns} and $Pr_{gen\ tr}$, respectively. For Pr_{re} , it is assumed that $Pr_{re} = 1 - Pr_{cr}$. The population size of both GA and IGA were chosen as 100 based on some pilot studies. The

stopping condition was set to 100 repetitions of the best found solution without any improvement in the fitness function, or when IGA had reached 1000 generations. To be able to show the influence of the VNS and GT mechanisms, the parameters of the GA and its stopping condition were chosen to be equal to the IGA settings.

To take into account the stochastic nature of the algorithms and test their performance, all the benchmark problems and the case study were solved ten times by both IGA and GA. As the optimal NS for the solved test problems exist in the literature, only the best found NS (NS_{Best}) is reported for each problem. However, since the optimal value of SI is not known for any of the problems, the best, average and standard deviation of values obtained for SI (i.e., SI_{Best} , SI_{Avg} and SI_{Std}) are reported in Table 2 where $SI_{Std} = \sqrt{\sum_{Rep=1}^{10} (SI_{Rep} - SI_{Avg})^2 / 10}$ and SI_{Rep} is the SI value obtained at Rep^{th} replication of IGA ($Rep=1, \dots, 10$). In this table, the first four columns show the problems' characteristics including the size, problem name, number of tasks (N) and CT . Column NS_{Opt} represents the optimal NS for each problem available in the literature. Table 2 shows the performance of the IGA compared to basic GA (without VNS and GT) as well as two efficient algorithms available in the literature, namely the "multi-objective multi-rule simulated annealing algorithm" (MRSA) and, "multi-objective hybrid improvement heuristic" (HIH) proposed by Baykasoğlu (2006) and Özcan and Toklu (2009), respectively. These two studies are the only ones dealing with exactly the same problem and objectives and solving the standard test problems. However, only the best results (SI_{Best}) are reported in these two previous studies and not the average (SI_{Avg}) and the standard deviation (SI_{Std}).

The CPU time is not reported in the two previous similar studies. However, the CPU time for each solved problem by GA and IGA are reported in Table 2. According to Table 2, the difference on run time between two algorithms is not constant for different problems. However it is evident that, there is no significant difference between GA and IGA in terms of CPU time for small- and medium- sized problems. As for large size problems, IGA consumed more time than the GA, ranging from about 30 to 150 seconds depending on the problem. For the case study, IGA also showed a slightly higher CPU time compared to GA.

This higher CPU time can be explained by the additional improvement mechanisms implemented in IGA, namely VNS and GT. Considering that the highest CPU time for the large-sized problems is only 210 seconds, a solution with higher quality is undoubtedly preferable and the time difference is negligible.

Table 2. Comparison of IGA with other metaheuristics in terms of NS_{Best} , SI_{Best} , SI_{Avg} and SI_{Std}

Size	Problem	N	CT	NS _{opt}	MRSA		HHH		GA				CPU time ^s	IGA				CPU time
					NS _{Best}	SI _{Best}	NS _{Best}	SI _{Best}	NS _{Best}	SI _{Best}	SI _{Avg}	SI _{Std}		NS _{Best}	SI _{Best}	SI _{Avg}	SI _{Std}	
Small	Case study	41	60	NA*	NA	NA	NA	NA	6	5.43	6.29	0.68	3.24	6	2.02	2.56	0.38	32.39
			65	NA	NA	NA	NA	NA	5	0.14	0.26	0.09	5.28	5	0.04	0.12	0.05	23.90
			70	NA	NA	NA	NA	NA	5	3.43	4.44	1.02	2.75	5	1.14	2.42	0.48	28.41
			75	NA	NA	NA	NA	NA	5	13.42	15.06	0.98	1.90	5	7.98	9.83	1.06	56.74
	Mertens	7	6	6	6	1.35	6	1.35	6	1.35	1.35	0.00	<1	6	1.35	1.35	0.00	<1
			7	5	5	1.41	5	1.41	5	1.41	1.41	0.00	<1	5	1.41	1.41	0.00	<1
			8	5	5	1.41	5	1.41	5	1.41	1.41	0.00	<1	5	1.41	1.41	0.00	<1
			10	3	3	0.58	3	0.58	3	0.58	0.58	0.00	<1	3	0.58	0.58	0.00	<1
			15	2	2	0.71	2	0.71	2	0.71	0.71	0.00	<1	2	0.71	0.71	0.00	<1
			18	2	2	3.54	2	3.54	2	3.54	3.54	0.00	<1	2	3.54	3.54	0.00	<1
		9	6	8	8	1.70	8	1.70	8	1.70	1.70	0.00	<1	8	1.70	1.70	0.00	<1
			7	7	7	2.00	7	2.00	7	2.00	2.00	0.00	<1	7	2.00	2.00	0.00	<1
			8	6	6	2.35	6	2.35	6	2.35	2.35	0.00	<1	6	2.35	2.35	0.00	<1
			10	4	4	0.87	4	0.87	4	0.87	0.87	0.00	<1	4	0.87	0.87	0.00	<1
			18	3	3	7.51	3	7.51	3	7.51	7.51	0.00	<1	3	7.51	7.51	0.00	<1
		11	7	8	8	1.66	8	1.66	8	1.66	1.66	0.00	<1	8	1.66	1.66	0.00	<1
			9	6	6	1.73	6	1.73	6	1.73	1.73	0.00	<1	6	1.73	1.73	0.00	<1
			10	5	5	1.10	5	1.10	5	1.10	1.10	0.00	<1	5	1.10	1.10	0.00	<1
			13	4	4	0.71	4	0.71	4	0.71	0.71	0.00	<1	4	0.71	0.71	0.00	<1
			14	4	4	0.71	4	0.71	4	0.71	0.71	0.00	<1	4	0.71	0.71	0.00	<1
			21	3	3	5.80	3	5.80	3	5.80	5.80	0.00	<1	3	5.80	5.80	0.00	<1
Medium	Mitchell	21	14	8	8	1.06	8	1.06	8	1.06	1.06	0.00	<1	8	1.06	1.06	0.00	<1
			15	8	8	2.32	8	2.32	8	2.32	2.32	0.00	<1	8	2.32	2.32	0.00	<1
			21	5	5	0.00	5	0.00	5	0.00	0.00	0.00	<1	5	0.00	0.00	0.00	<1
	Heskia	28	138	8	8	9.01	8	5.83	8	5.72	6.60	0.73	1.71	8	5.68	5.75	0.07	4.75
			205	5	5	0.45	5	0.45	5	0.45	13.08	11.55	1.68	5	0.45	2.37	6.08	2.54
			216	5	5	4.10	5	2.49	5	2.79	3.84	0.58	1.34	5	1.41	2.27	0.46	4.11
			256	4	4	0.00	4	0.00	4	0.00	0.00	0.00	<1	4	0.00	0.00	0.00	1.06
			324	4	4	73.88	4	62.95	4	64.49	65.37	0.80	<1	4	62.95	63.57	0.80	3.16
			342	3	3	0.82	3	0.82	3	0.82	0.82	0.00	<1	3	0.82	0.82	0.00	<1
	Sawyer	30	25	14	14	2.36	14	2.20	14	2.20	2.36	0.15	1.99	14	2.20	2.23	0.03	4.71
			27	13	13	1.71	13	1.36	13	1.80	2.21	0.34	2.03	13	1.36	1.41	0.05	8.83
			30	12	12	2.42	12	2.42	12	2.61	3.01	0.48	2.90	12	2.35	2.40	0.04	13.14
			36	10	10	2.05	10	1.84	10	2.93	3.00	0.05	2.79	10	1.84	1.93	0.07	9.63
			41	8	8	0.71	8	0.71	8	0.71	2.45	2.01	<1	8	0.71	0.71	0.00	<1

Table 2. Comparison of IGA with other metaheuristics in terms of NS_{Best} , SI_{Best} , SI_{Avg} and SI_{Std}

Size	Problem	N	CT	NS _{Opt}	MRSA		HHI		GA				IGA					
					NS _{Best}	SI _{Best}	NS _{Best}	SI _{Best}	NS _{Best}	SI _{Best}	SI _{Avg}	SI _{Std}	CPU time ^s	NS _{Best}	SI _{Best}	SI _{Avg}	SI _{Std}	CPU time
Large	Killbridge	45	54	7	7	2.65	7	2.39	7	2.39	4.57	1.33	1.87	7	2.27	2.37	0.05	1.88
			75	5	5	9.32	5	3.95	5	4.05	4.41	0.50	1.02	5	3.74	3.91	0.09	1.49
			57	10	10	1.00	10	0.89	10	1.10	1.31	0.39	2.82	10	0.89	0.99	0.03	12.16
			79	7	7	0.38	7	0.38	7	0.38	5.11	2.93	7.62	7	0.38	0.38	0.00	16.10
			92	6	6	0.00	6	0.00	6	0.00	10.18	5.78	4.57	6	0.00	2.41	3.95	9.50
			110	6	6	25.05	6	18.88	6	18.88	22.84	3.28	2.53	6	9.15	15.83	3.17	23.96
			138	4	4	0.00	4	0.00	4	0.00	0.00	0.00	<1	4	0.00	0.00	0.00	1.70
			184	3	3	0.00	3	0.00	3	0.00	0.00	0.00	3.05	3	0.00	0.00	0.00	12.00
	Tonge	70	176	21	21	8.42	21	8.70	21	12.04	17.21	2.90	17.32	21	7.89	8.92	0.65	152.65
			364	10	10	4.80	10	5.75	10	6.84	8.83	2.31	8.04	10	3.69	4.41	0.68	45.04
			410	9	9	6.90	9	6.20	9	9.73	11.18	1.37	12.34	9	3.82	4.51	0.40	44.64
			468	8	8	15.05	8	9.30	8	16.03	21.73	4.03	6.46	8	8.50	10.02	1.47	92.19
			527	7	7	14.54	7	9.85	7	12.12	14.93	2.12	6.50	7	5.73	7.15	0.95	47.73
	Arcus1	83	5048	16	16	284.79	16	260.25	16	294.22	298.27	3.76	15.67	16	249.19	251.91	3.49	165.71
			5853	14	14	293.29	14	225.64	14	290.47	335.51	32.67	28.39	14	97.39	206.60	42.83	177.06
			6842	12	12	522.81	12	387.14	12	431.96	506.88	44.15	15.43	12	325.61	362.01	39.43	155.28
			7571	11	11	650.62	11	557.36	11	570.33	627.44	54.21	27.89	11	303.52	412.33	102.30	108.38
			8412	10	10	1102.57	10	610.88	10	620.61	809.14	198.94	21.76	10	491.49	567.58	64.31	93.60
			8898	9	9	149.37	9	128.59	9	145.41	177.24	19.80	16.15	9	127.61	133.83	4.19	170.66
			10816	8	8	2387.01	8	1941.41	8	1934.73	2047.64	101.11	21.22	8	1869.45	1894.33	14.13	81.48
	Arcus2	111	5755	27	27	312.59	27	300.55	27	320.62	457.98	68.11	61.00	27	298.99	304.58	4.35	210.35
			8847	18	18	650.65	18	460.05	18	444.33	524.01	46.31	44.56	18	284.15	337.05	46.65	153.64
			10027	16	16	923.23	16	538.89	16	566.00	604.21	63.30	67.98	16	381.15	444.75	35.91	196.29
			10743	15	15	1138.43	15	690.81	15	759.36	822.21	54.34	83.47	15	394.24	474.27	52.89	210.08
			11378	14	14	988.74	14	395.41	14	546.86	600.73	51.74	86.42	14	286.60	323.97	16.54	166.81
			17067	9	9	492.40	9	186.28	9	220.71	287.40	35.69	30.25	9	89.88	157.24	60.98	166.21

* Not available (NA) in the literature; § The CPU time is in seconds.

According to Table 2, all the algorithms achieved the optimal NS (NS_{opt}) for all the solved benchmark problems. Thus, in terms of NS the performance of the considered algorithms is similar and accordingly no further analysis in this regard is performed. As for the case study, both IGA and GA found the same number of stations over all the considered CT s. Therefore, none of the considered CT s provide an advantage over the others in terms of NS . Comparing the results obtained for the case study in terms of SI , it can be seen that the CT of 65 seconds has resulted in the lowest SI value ($SI = 0.04$) compared to other CT scenarios.

To better compare the performance of the IGA with the other algorithms considered in terms of SI , the outperforming percent (OP) of IGA over other algorithms is presented in Table 3 under column OP . The OP is calculated by dividing the number of times IGA outperformed the other algorithms in terms of SI_{Best} by the total number of test problems.

To further quantify the amount of improvement obtained by IGA compared to other considered algorithms in terms of SI_{Best} , a new measure called relative percent improvement (RPI) is calculated using Equation (2). The average RPI obtained by IGA compared to the other algorithms in terms of SI_{Best} for each set of the problems solved is also reported in Table 3.

$$RPI = \frac{SI_{Best}^{Alg} - SI_{Best}^{IGA}}{SI_{Best}^{Alg}} \times 100; \quad Alg \in \{MRSA, HHH, GA\} \tag{2}$$

Table 3. Comparison of IGA with other algorithms in terms of OP and average RPI for SI_{Best}

Problem	# of problems	IGA vs. MRSA		IGA vs. HHH		IGA vs. GA	
		OP	RPI	OP	RPI	OP	RPI
Case study	4	NA*	NA	NA	NA	100%	60%
Small	17	0%	0%	0%	0%	0%	0%
Medium	22	50%	14%	27%	5%	45%	9%
Large	18	100%	43%	100%	26%	100%	37%
Total	61	51%	19%	42%	10%	52%	18%

* Not available

Table 3 shows that the IGA improved on the GA with OP of 100% and average RPI of 60% for the case study. As for the small-sized problems, the results obtained by IGA and other algorithms are similar and thus the value of OP and average RPI is equal to zero. This situation may be the result of limited solution space in small-sized problems that do not contain many hills and valleys. The analysis of results for medium-sized test problems showed that the IGA provided better solutions compared to MRSA, HHH and GA by 50%, 27% and 45%, respectively, in terms of OP . It also showed that IGA outperformed those three algorithms in terms of average RPI by 14%, 5% and 9%, respectively. These values of OP and RPI over the medium-sized problems show that the performance of the existing algorithms and pure GA in finding good solutions was not efficient enough. The results for larger test problems are even more interesting. The comparison of the performance of IGA with other algorithms for large-sized test problems showed a noticeable outperforming percent (OP) of 100%. Moreover, the IGA was superior to the other three algorithms in terms of RPI by 43%, 26% and 37%, respectively. Overall, Table 3 shows that, moving from small- to large-sized problems, IGA provides significantly better solutions in terms of SI_{Best} as evidenced by considerable improvements in both OP and average RPI .

The last row of Table 3 summarizes the performance of IGA over all the solved problems compared to the other three algorithms. The results show that in terms of OP the IGA outperformed the other algorithms by a minimum of 42% and a maximum of 52%. In terms of average RPI , the IGA was shown to be superior to the other compared algorithms by a minimum of 10% and a maximum of 19%. Considering all the above comparisons, it is safe to conclude that IGA can result in a smoother workload for the stations, particularly for medium- and large-sized problems, while also being able to find the minimum NS .

The analysis revealed that all the algorithms considered in this study are similarly efficient in finding the optimal NS . The results reported in Table 3 and their analysis showed that the proposed IGA outperforms all the other algorithms considered in terms of SI_{Best} . However, the superiority of IGA cannot be judged only by analyzing SI_{Best} . Therefore, statistical analysis of the results in terms of SI_{Avg} and SI_{Std} (as reported in Table 2) was done to reach a firm conclusion. The analysis is performed only based on the obtained results by IGA and GA because the value of SI_{Avg} and SI_{Std} are not available for the other two algorithms (i.e., MRSA and HIH).

The Wilcoxon signed rank test is applied as a nonparametric test to evaluate whether there is a significant difference between the performance of the GA and IGA in terms of SI_{Avg} and SI_{Std} at the 0.05 significance level. According to the results of the Wilcoxon test, both the SI_{Avg} and the SI_{Std} obtained by IGA are significantly lower than GA. Thus, the superiority of IGA over GA is statistically proven. A detailed explanation about the performed statistical analysis can be found in the [supplementary document](#).

6. Conclusion

This study dealt with a specific type of assembly line balancing problem, where the number of stations (NS) has to be minimized for given a cycle time (CT). To maximize the workload smoothness at the stations, a secondary objective known as the smoothness index (SI) also needed to be optimized. To address the problem, an improved genetic algorithm (IGA) was proposed in which the local search ability of the algorithm has been improved by including a variable neighborhood search (VNS) in the genetic algorithm (GA). In addition, a new mechanism named generation transfer (GT) was introduced and implemented in the body of the GA to improve its diversification capability as the evolution progresses.

The computational results were tested on a real case taken from an automobile manufacturing company and against a wide range of known standard test problems from the literature. The performance of IGA was benchmarked against the basic GA (without VNS and GT) and against two existing algorithms in the literature. The results show that the proposed IGA is promisingly efficient in finding the optimum NS for all the solved problems within a reasonable computational time. The IGA results showed that the decision makers in the real case considered should choose the CT of 65 seconds, which yields the lowest NS (5 stations) and provides a very smooth workload at stations ($SI = 0.04$). The results show that IGA outperforms the other algorithms and the basic GA in optimizing the SI , particularly for medium- and large-sized problems. Finally, the superiority of IGA was further tested and

proved through statistical analysis, confirming its effectiveness in dealing with a wide range of test problems.

As for future research, the performance of the proposed IGA can be tested on other assembly line configurations such as U-shaped lines. The same algorithm can also be used for solving other types of assembly line balancing problems with the same or different optimization objectives. Given the promising effect of the proposed generation transfer mechanism, it can be employed in other population-based algorithms. Finally, the influence of VNS and GT on the quality of GA solutions can be further investigated. This can be done by separately incorporating VNS and GT within GA and comparing their results with IGA.

Appendix

Table A.1. The data for the case study

Task No.	Predecessor	Task time (Sec.)	Task No.	Predecessor	Task time (Sec.)
1	-	5.3	22	21, 26	11
2	1	3.1	23	-	4.9
3	2	6	24	23	4.5
4	3	7.1	25	19, 14, 22	7.2
5	2	6.5	26	-	10.3
6	5	7.1	27	25	2.4
7	4	11.1	28	26, 27	5.1
8	6	11.1	29	28	8.8
9	-	11.2	30	20, 26	15.7
10	-	3.3	31	30	11.9
11	10	12.6	32	31	5.3
12	11	15.5	33	31	4.9
13	-	3.6	34	32, 33	18.1
14	10	2.7	35	34	13.5
15	10	16.5	36	35	3.2
16	9	9.9	37	32, 33	2.7
17	10	8.5	38	37	4.5
18	17	4.4	39	-	3.3
19	10	2.7	40	38	4.7
20	14, 19	9.4	41	40	8.4
21	17	8.9			

References

Aledo, J. A., Gámez, J. A. and Rosete, A. (2017), "Partial evaluation in Rank Aggregation Problems", *Computers and Operations Research*, Vol. 78, pp. 299–304.

Azizoğlu, M. and İmat, S. (2018), "Workload smoothing in simple assembly line balancing", *Computers and Operations Research*, Vol. 89, pp. 51–57.

Baniamerian, A., Bashiri, M. and Zabihi, F. (2018), "A modified variable neighborhood search hybridized with genetic algorithm for vehicle routing problems with cross-docking", *Electronic Notes in Discrete Mathematics*, Vol. 66, pp. 143–150.

Battaia, O. and Dolgui, A. (2013), "A taxonomy of line balancing problems and their solution approaches", *International Journal of Production Economics*, Vol. 142 No. 2, pp. 259–277.

Baykasoğlu, A. (2006), "Multi-rule Multi-objective Simulated Annealing Algorithm for Straight and U Type Assembly Line Balancing Problems", *Journal of Intelligent Manufacturing*, Vol. 17 No. 2, pp. 217–232.

- Chutima, P. and Chimklai, P. (2012), "Multi-objective two-sided mixed-model assembly line balancing using particle swarm optimisation with negative knowledge", *Computers and Industrial Engineering*, Vol. 62 No. 1, pp. 39–55.
- Deb, K. (1995), *Optimisation for Engineering Design*, Prentice-Hall, New Delhi.
- De Albuquerque, I. M. C., Filho, J. M., de Lima Neto, F. B. and de Oliveira Silva, A. M. (2016), "Solving Assembly Line Balancing Problems with Fish School Search algorithm", In *2016 IEEE Symposium Series on Computational Intelligence (SSCI). Athens, Greece, 2007*, IEEE, pp. 1-8.
- Dou, J., Li, J. and Zhao, X. (2017), "A novel discrete particle swarm algorithm for assembly line balancing problems", *Assembly Automation*, Vol. 37 No. 4, pp. 452–463.
- Fathi, M., Álvarez, M. J. and Rodríguez, V. (2016a), "A new heuristic-based bi-objective simulated annealing method for U-shaped assembly line balancing", *European Journal of Industrial Engineering*, Vol. 10 No. 2, pp. 145–169.
- Fathi, M., Fontes, D. B. M. M., Urenda Moris, M. and Ghobakhloo, M. (2018), "Assembly line balancing problem: A comparative evaluation of heuristics and a computational assessment of objectives", *Journal of Modelling in Management*, Vol. 13 No. 2, pp. 455–474.
- Fathi, M., Rodríguez, V., Fontes, D. B. M. M. and Alvarez, M. J. (2016b), "A modified particle swarm optimisation algorithm to solve the part feeding problem at assembly lines", *International Journal of Production Research*, Vol. 54 No. 3, pp. 878–893.
- Fathi, M., Nourmohammadi, A., Ng, A. H. C. and Syberfeldt, A. (2019), "An optimization model for balancing assembly lines with stochastic task times and zoning constraints", *IEEE Access*, Vol. 7 No. 1, pp. 32537–32550.
- Hwang, R. and Katayama, H. (2010), "Uniform workload assignments for assembly line by GA-based amelioration approach", *International Journal of Production Research*, Vol. 48 No. 7, pp. 1857–1871.
- Lalaoui, M. and Afia, A. El. (2018), "A Fuzzy generalized simulated annealing for a simple assembly line balancing problem", *IFAC-PapersOnLine*, Vol. 51 No. 32, pp. 600–605.
- Li, Z., Kucukkoc, I. and Nilakantan, J. M. (2017), "Comprehensive review and evaluation of heuristics and meta-heuristics for two-sided assembly line balancing problem", *Computers and Operations Research*, Vol. 84, pp. 146–161.
- Liu, S. B., Ong, H. L. and Huang, H. C. (2003), "Two bi-directional heuristics for the assembly line type II problem", *The International Journal of Advanced Manufacturing Technology*, Vol. 22 Nos. 9–10, pp. 656–661.
- Man, K.F., TANG, K. S., Kwong, S. (1999), *Genetic Algorithms Concepts and Designs*, Springer, Heidelberg.
- Nearchou, A. C. (2008), "Multi-objective balancing of assembly lines by population heuristics", *International Journal of Production Research*, Vol. 46 No. 8, pp. 2275–2297.
- Nearchou, A. C. (2011), "Maximizing production rate and workload smoothing in assembly lines using particle swarm optimization", *International Journal of Production Economics*, Vol. 129 No.2, pp. 242–250.
- Nourmohammadi, A. and Eskandari, H. (2017), "Assembly line design considering line balancing and part feeding", *Assembly Automation*, Vol. 37 No. 1, pp. 135–143.
- Nourmohammadi, A., Eskandari, H. and Fathi, M. (2019), "Design of stochastic assembly lines considering line balancing and part feeding with supermarkets", *Engineering Optimization*, Vol. 51 No. 1, pp. 63–83.
- Nourmohammadi, A. and Zandieh, M. (2011), "Assembly line balancing by a new multi-objective differential evolution algorithm based on TOPSIS", *International Journal of Production Research*, Vol. 49 No. 10, pp. 2833–2855.
- Özcan, U. and Toklu, B. (2009), "A new hybrid improvement heuristic approach to simple straight and U-type assembly line balancing problems", *Journal of Intelligent Manufacturing*, Vol. 20 No. 1, pp. 123–136.
- Pereira, J. and Álvarez-Miranda, E. (2018), "An exact approach for the robust assembly line balancing problem", *Omega*, Vol. 78, pp. 85–98.
- Petropoulos, D. I. and Nearchou, A. C. (2011), "A particle swarm optimization algorithm for balancing assembly lines", *Assembly Automation*, Vol. 31 No. 2, pp. 118–129.
- Ponnambalam, S. G., Aravindan, P. and Mogileeswar Naidu, G. (2000), "A Multi-Objective Genetic Algorithm for Solving Assembly Line Balancing Problem", *The International Journal of Advanced Manufacturing Technology*, Vol. 16 No. 5, pp. 341–352.

Razali, M. M., Kamarudin, N. H., Ab. Rashid, M. F. F. and Mohd Rose, A. N. (2019), “Recent trend in mixed-model assembly line balancing optimization using soft computing approaches”, *Engineering Computations*, Vol. 36 No. 2, pp. 622–645.

Rezaie, A., Tsatsaronis, G. and Hellwig, U. (2019), “Thermal design and optimization of a heat recovery steam generator in a combined-cycle power plant by applying a genetic algorithm”, *Energy*, Vol. 168, pp. 346–357.

Scholl, A., Boysen, N., Flidner, M. and Klein, R. (1995), “Homepage for assembly line optimization research”, available at: <https://assembly-line-balancing.de/> (accessed 10 January 2017)

Sivasankaran, P. and Shahabudeen, P. (2014), “Literature review of assembly line balancing problems”, *The International Journal of Advanced Manufacturing Technology*, Vol. 73 Nos.9–12, pp. 1665–1694.

Taguchi, G., S. Chowdhury and Y. Wu. (2005), *Taguchi’s Quality Engineering Handbook*, Wiley, Hoboken, NJ.

Yu, J. F. and Yin, Y. H. (2010), “Assembly line balancing based on an adaptive genetic algorithm”, *The International Journal of Advanced Manufacturing Technology*, Vol. 48 Nos.1–4, pp. 347–354.

Zacharia, P. T. and Nearchou, A. C. (2016), “A population-based algorithm for the bi-objective assembly line worker assignment and balancing problem”, *Engineering Applications of Artificial Intelligence*, Vol. 49, pp. 1–9.

Zhang, G., Zhang, L., Song, X., Wang, Y. and Zhou, C. (2018), “A variable neighborhood search based genetic algorithm for flexible job shop scheduling problem”, *Cluster Computing*, In press, doi.org/10.1007/s10586-017-1420-4

Zhang, H. (2017), “An improved immune algorithm for simple assembly line balancing problem of type 1”, *Journal of Algorithms and Computational Technology*, Vol. 11 No. 4, pp. 317–326.

Zhang, H. (2019), “An immune genetic algorithm for simple assembly line balancing problem of type 1”, *Assembly Automation*, Vol. 39 No. 1, pp. 113–123.

Zhao, F., Liu, Y., Zhang, Y., Ma, W. and Zhang, C. (2017), “A hybrid harmony search algorithm with efficient job sequence scheme and variable neighborhood search for the permutation flow shop scheduling problems”, *Engineering Applications of Artificial Intelligence*, Vol. 65, pp. 178–199.

Zhong, Y. and Ai, B. (2017), “A modified ant colony optimization algorithm for multi-objective assembly line balancing”, *Soft Computing*, Vol. 21 No. 22, pp. 6881–6894.

Supplementary document

1. Mathematical formulation

In this section, the mathematical formulation of the considered SALBP-1 is presented. The main objective of SALBP-1 is to minimize the NS . The smoothness index (SI), which measures the level of workload equalization among the stations, is also to be minimized as the secondary objective. The lower the value of SI , the smoother is the distribution of workload among the stations. Table 1 presents the notations used in the mathematical model.

Table 1. List of notations

i, j :	Task index $i, j = 1, 2, \dots, N$;
k :	Station index $k = 1, 2, \dots, K$;
N :	Number of tasks;
CT :	Given cycle time;
t_i :	Processing time of task i ;
S_k :	Set of tasks assigned to station k ;
$t(S_k)$:	Time of station k ;
Pr_{ij} :	Precedence matrix: $\begin{cases} 1; & \text{if task } i \text{ is the predecessor of task } j, \\ 0; & \text{otherwise} \end{cases}$;
NS_{max} :	Maximum number of stations; $NS_{max} = \left\lceil \frac{\sum_{i=1}^N t_i}{\max_{\forall i} t_i} \right\rceil$
x_{ik} :	$\begin{cases} 1; & \text{If task } i \text{ is assigned to station } k, \\ 0; & \text{otherwise} \end{cases}$;
y_k :	$\begin{cases} 1; & \text{If station } k \text{ is established,} \\ 0; & \text{otherwise} \end{cases}$;
NS :	Number of stations;
SI :	Smoothness index;

The considered SALBP-1 is modeled as follows.

$$Min \begin{cases} (1) NS = \sum_{k=1}^{NS_{max}} y_k \\ (2) SI = \sqrt{\frac{\sum_{k=1}^{NS_{max}} ((\max(t(S_k)) - t(S_k)) \times y_k)^2}{NS}} \end{cases} \quad (1)$$

s.t:

$$\sum_{k=1}^{NS_{max}} x_{ik} = 1 \quad \forall i = 1, \dots, N \quad (2)$$

$$\sum_{i=1}^N t_i x_{ik} \leq CT \times y_k \quad \forall k = 1, \dots, NS_{\max} \quad (3)$$

$$\sum_{k=1}^{NS_{\max}} k.x_{ik} \leq \sum_{k=1}^{NS_{\max}} k.x_{jk} \quad \forall (i, j) \in Pr \quad (4)$$

$$t(S_k) = \sum_{i=1}^N t_i x_{ik} \quad \forall k = 1, \dots, NS_{\max} \quad (5)$$

$$x_{ik} \in \{0, 1\} \quad i = 1, \dots, N \quad k = 1, \dots, NS_{\max} \quad (6)$$

$$y_k \in \{0, 1\} \quad k = 1, \dots, NS_{\max} \quad (7)$$

Equation (1) represents the two considered objective functions, NS and SI , to be minimized. Constraint (2) ensures that each task is assigned to only one station. Constraint (3) ensures that the station workload in terms of time does not exceed the given CT . Constraint (4) guarantees that the precedence relations between tasks are satisfied. Constraint (5) calculates the time at each station. Finally, constraints (6) and (7) restrict the decision variables domain to binary numbers.

2. The proposed IGA

A summary of the proposed IGA is presented in pseudo code 1, below. All the notations used in the pseudo code can be found in the manuscript.

Pseudo-code 1. The proposed IGA

Pre-processing step

Read input parameters: precedence relationships, task times t_i ($i = 1, \dots, N$), CT ;

Initialization step

Set values for the IGA control parameters: including reproduction rate (Pr_{re}), crossover rate (Pr_{cr}), mutation rate (Pr_{mu}), VNS rate (Pr_{vns}), GT rate ($Pr_{gen tr}$) and population size (PS);

Initialize the generation counter ($GC = 0$);

Generate a population of integer vectors with length N , in which each vector is a permutation of integer numbers from $[1, N]$;

Repeat

For $q = 1$ to PS do;

Begin

Crossover step

Considering the Pr_{cr} , apply the crossover scheme in section 4.5 (in manuscript) on the two selected individuals;

Mutation step

Considering the Pr_{mu} , apply the mutation scheme in section 4.6 on a randomly selected individual;

Encoding and decoding schemes

Use the encoding procedure in section 4.2 to map each vector of task priorities to a feasible task sequence (TS). Then, apply the decoding procedure in section 4.2 to assign the tasks in TS to the stations;

Evaluation step

Using Equation (1) in the manuscript, calculate the fitness function for each solution in the current population;

Elitist preserving mechanism

Rank the solutions in the current population according to their fitness function and select the best found solutions as the next population members considering the Pr_{re} (if $GC = 0$ select them randomly);

End;

VNS procedure

Considering the Pr_{vns} , perform the VNS operator in section 4.7;

Generation transfer procedure

Considering the $Pr_{gen tr}$, perform the GT procedure in section 4.8;

Update the best found solution

Update the best found solution till now;

$GC = GC + 1$;

Until the stopping condition in section 5 is met;

Return the best found solution

3. Statistical analysis

The statistical analysis performed to compare the performance of GA and IGA in terms of SI are presented in this section. The two statistical hypotheses relating to SI_{Avg} and SI_{Std} are presented in Equations (8) and (9), respectively.

$$\begin{cases} H_0 : \text{There is \textbf{not} a significant difference between GA and IGA in terms of } SI_{Avg} \\ H_1 : \text{There is a significant difference between GA and IGA in terms of } SI_{Avg} \end{cases} \quad (8)$$

$$\begin{cases} H_0 : \text{There is \textbf{not} a significant difference between GA and IGA in terms of } SI_{Std} \\ H_1 : \text{There is a significant difference between GA and IGA in terms of } SI_{Std} \end{cases} \quad (9)$$

The hypotheses were tested using SPSS statistics software, and the p -value for both hypotheses was found to be 0.000. Since the obtained p -values are less than $\alpha = 0.05$, the null hypothesis of both statistical tests is rejected. Therefore, it is statistically proved that there is a significant difference between the performance of GA and IGA in terms of both SI_{Avg} and SI_{Std} . Next, it should be decided which algorithm offers better results. Remembering that the preferred algorithm should provide lower SI_{Avg} and SI_{Std} , Table 2 shows the ranks obtained by the Wilcoxon test.

Considering SI_{Avg} , Table 2 shows that in none of the solved problems did GA obtain a lower value than IGA. Setting aside the 24 problems for which SI_{Avg} for GA and IGA are identical, this table shows that for 37 of the solved problems the SI_{Avg} provided by IGA is lower than that provided by GA. On average, the mean rank of SI_{Avg} obtained by IGA is 19 units better than the mean rank of SI_{Avg} obtained by GA.

Table 2 indicates that the value provided by IGA is lower than GA for 30 of the solved problems for SI_{Std} . GA only found a lower value in six problems compared to IGA, and both algorithms found the same value in 25 problems. On average, the mean rank of SI_{Std} obtained by IGA is 18.70 units better than the mean rank of SI_{Std} obtained by GA.

According to the results of the Wilcoxon test, it can be concluded that both the SI_{Avg} and the SI_{std} obtained by IGA are significantly lower than GA. Thus, the superiority of IGA over GA is statistically proven.

Table 2. Results obtained by Wilcoxon signed rank tests for SI_{Avg} and SI_{std}

GA - IGA	SI_{Avg}			SI_{std}		
	Sample size	Mean Rank	Sum of Ranks	Sample size	Mean Rank	Sum of Ranks
Negative Ranks	0 ^a	0.00	0.00	6	17.50	105.00
Positive Ranks	37 ^b	19.00	703.00	30	18.70	561.00
Ties	24 ^c			25		
Total	61			61		

^aGA < IGA, ^bGA > IGA, ^cGA = IGA

Table 1. Summary of the reviewed studies

Study	Objective function							Optimization algorithm									
	<i>NS</i>	<i>CT</i>	<i>SI</i>	<i>VW</i>	<i>SOL</i>	<i>BD</i>	<i>MAD</i>	<i>GA</i>	<i>ACO</i>	<i>PSO</i>	<i>HH</i>	<i>SA</i>	<i>DE</i>	<i>IA</i>	<i>HGA</i>	<i>OHA</i>	<i>FSS</i>
Ponnambalam et al. (2000)	•		•**					•									
Liu et al. (2003)		•					•										•
Baykasoğlu (2006)	•		•									•					
Nearchou (2008)		•	•			•							•				
Özcan and Toklu (2009)	•		•	•							•						
Yu and Yin (2010)	•		•*					•									
Petropoulos and Nearchou (2011)		•				•	•				•						
Nearchou (2011)		•					•				•						
Nourmohammadi and Zandieh (2011)		•	•										•				
De Albuquerque et al., (2016)	•				•												•
Dou et al. (2017)	•		•*								•						
Zhang (2017)	•		•*												•		
Zhong and Ai (2017)	•	•		•					•								
Lalaoui and Afia, (2018)	•				•							•					
Zhang (2019)	•		•**													•	

*Only a case study or an example is considered; **The detailed results for *SI* are not reported.

Table 2. Comparison of IGA with other metaheuristics in terms of NS_{Best} , SI_{Best} , SI_{Avg} and SI_{Std}

Size	Problem	N	CT	NS_{Opt}	MRSA		HIH		GA				CPU time ^s	IGA				CPU time
					NS_{Best}	SI_{Best}	NS_{Best}	SI_{Best}	NS_{Best}	SI_{Best}	SI_{Avg}	SI_{Std}		NS_{Best}	SI_{Best}	SI_{Avg}	SI_{Std}	
	Case study	41	60	NA*	NA	NA	NA	NA	6	5.43	6.29	0.68	3.24	6	2.02	2.56	0.38	32.39
			65	NA	NA	NA	NA	NA	5	0.14	0.26	0.09	5.28	5	0.04	0.12	0.05	23.90
			70	NA	NA	NA	NA	NA	5	3.43	4.44	1.02	2.75	5	1.14	2.42	0.48	28.41
			75	NA	NA	NA	NA	NA	5	13.42	15.06	0.98	1.90	5	7.98	9.83	1.06	56.74
Small	Mertens	7	6	6	6	1.35	6	1.35	6	1.35	1.35	0.00	<1	6	1.35	1.35	0.00	<1
			7	5	5	1.41	5	1.41	5	1.41	1.41	0.00	<1	5	1.41	1.41	0.00	<1
			8	5	5	1.41	5	1.41	5	1.41	1.41	0.00	<1	5	1.41	1.41	0.00	<1
			10	3	3	0.58	3	0.58	3	0.58	0.58	0.00	<1	3	0.58	0.58	0.00	<1
			15	2	2	0.71	2	0.71	2	0.71	0.71	0.00	<1	2	0.71	0.71	0.00	<1
			18	2	2	3.54	2	3.54	2	3.54	3.54	0.00	<1	2	3.54	3.54	0.00	<1
	Jaeschke	9	6	8	8	1.70	8	1.70	8	1.70	1.70	0.00	<1	8	1.70	1.70	0.00	<1
			7	7	7	2.00	7	2.00	7	2.00	2.00	0.00	<1	7	2.00	2.00	0.00	<1
			8	6	6	2.35	6	2.35	6	2.35	2.35	0.00	<1	6	2.35	2.35	0.00	<1
			10	4	4	0.87	4	0.87	4	0.87	0.87	0.00	<1	4	0.87	0.87	0.00	<1
			18	3	3	7.51	3	7.51	3	7.51	7.51	0.00	<1	3	7.51	7.51	0.00	<1
	Jackson	11	7	8	8	1.66	8	1.66	8	1.66	1.66	0.00	<1	8	1.66	1.66	0.00	<1
			9	6	6	1.73	6	1.73	6	1.73	1.73	0.00	<1	6	1.73	1.73	0.00	<1
			10	5	5	1.10	5	1.10	5	1.10	1.10	0.00	<1	5	1.10	1.10	0.00	<1
			13	4	4	0.71	4	0.71	4	0.71	0.71	0.00	<1	4	0.71	0.71	0.00	<1
			14	4	4	0.71	4	0.71	4	0.71	0.71	0.00	<1	4	0.71	0.71	0.00	<1
			21	3	3	5.80	3	5.80	3	5.80	5.80	0.00	<1	3	5.80	5.80	0.00	<1
Medium	Mitchell	21	14	8	8	1.06	8	1.06	8	1.06	1.06	0.00	<1	8	1.06	1.06	0.00	<1
			15	8	8	2.32	8	2.32	8	2.32	2.32	0.00	<1	8	2.32	2.32	0.00	<1
			21	5	5	0.00	5	0.00	5	0.00	0.00	0.00	<1	5	0.00	0.00	0.00	<1
	Heskia	28	138	8	8	9.01	8	5.83	8	5.72	6.60	0.73	1.71	8	5.68	5.75	0.07	4.75
			205	5	5	0.45	5	0.45	5	0.45	13.08	11.55	1.68	5	0.45	2.37	6.08	2.54
			216	5	5	4.10	5	2.49	5	2.79	3.84	0.58	1.34	5	1.41	2.27	0.46	4.11
			256	4	4	0.00	4	0.00	4	0.00	0.00	0.00	<1	4	0.00	0.00	0.00	1.06
			324	4	4	73.88	4	62.95	4	64.49	65.37	0.80	<1	4	62.95	63.57	0.80	3.16
			342	3	3	0.82	3	0.82	3	0.82	0.82	0.00	<1	3	0.82	0.82	0.00	<1
	Sawyer	30	25	14	14	2.36	14	2.20	14	2.20	2.36	0.15	1.99	14	2.20	2.23	0.03	4.71
			27	13	13	1.71	13	1.36	13	1.80	2.21	0.34	2.03	13	1.36	1.41	0.05	8.83
			30	12	12	2.42	12	2.42	12	2.61	3.01	0.48	2.90	12	2.35	2.40	0.04	13.14
			36	10	10	2.05	10	1.84	10	2.93	3.00	0.05	2.79	10	1.84	1.93	0.07	9.63
			41	8	8	0.71	8	0.71	8	0.71	2.45	2.01	<1	8	0.71	0.71	0.00	<1

Table 2. Comparison of IGA with other metaheuristics in terms of NS_{Best} , SI_{Best} , SI_{Avg} and SI_{Std}

Size	Problem	N	CT	NS_{Opt}	MRSA		HIH		GA				CPU time ^s	IGA				
					NS_{Best}	SI_{Best}	NS_{Best}	SI_{Best}	NS_{Best}	SI_{Best}	SI_{Avg}	SI_{Std}		NS_{Best}	SI_{Best}	SI_{Avg}	SI_{Std}	CPU time
Large	Killbridge	45	54	7	7	2.65	7	2.39	7	2.39	4.57	1.33	1.87	7	2.27	2.37	0.05	1.88
			75	5	5	9.32	5	3.95	5	4.05	4.41	0.50	1.02	5	3.74	3.91	0.09	1.49
			57	10	10	1.00	10	0.89	10	1.10	1.31	0.39	2.82	10	0.89	0.99	0.03	12.16
			79	7	7	0.38	7	0.38	7	0.38	5.11	2.93	7.62	7	0.38	0.38	0.00	16.10
			92	6	6	0.00	6	0.00	6	0.00	10.18	5.78	4.57	6	0.00	2.41	3.95	9.50
			110	6	6	25.05	6	18.88	6	18.88	22.84	3.28	2.53	6	9.15	15.83	3.17	23.96
			138	4	4	0.00	4	0.00	4	0.00	0.00	0.00	<1	4	0.00	0.00	0.00	1.70
			184	3	3	0.00	3	0.00	3	0.00	0.00	0.00	3.05	3	0.00	0.00	0.00	12.00
	Tonge	70	176	21	21	8.42	21	8.70	21	12.04	17.21	2.90	17.32	21	7.89	8.92	0.65	152.65
			364	10	10	4.80	10	5.75	10	6.84	8.83	2.31	8.04	10	3.69	4.41	0.68	45.04
			410	9	9	6.90	9	6.20	9	9.73	11.18	1.37	12.34	9	3.82	4.51	0.40	44.64
			468	8	8	15.05	8	9.30	8	16.03	21.73	4.03	6.46	8	8.50	10.02	1.47	92.19
			527	7	7	14.54	7	9.85	7	12.12	14.93	2.12	6.50	7	5.73	7.15	0.95	47.73
	Arcus1	83	5048	16	16	284.79	16	260.25	16	294.22	298.27	3.76	15.67	16	249.19	251.91	3.49	165.71
			5853	14	14	293.29	14	225.64	14	290.47	335.51	32.67	28.39	14	97.39	206.60	42.83	177.06
			6842	12	12	522.81	12	387.14	12	431.96	506.88	44.15	15.43	12	325.61	362.01	39.43	155.28
			7571	11	11	650.62	11	557.36	11	570.33	627.44	54.21	27.89	11	303.52	412.33	102.30	108.38
			8412	10	10	1102.57	10	610.88	10	620.61	809.14	198.94	21.76	10	491.49	567.58	64.31	93.60
			8898	9	9	149.37	9	128.59	9	145.41	177.24	19.80	16.15	9	127.61	133.83	4.19	170.66
	Arcus2	111	10816	8	8	2387.01	8	1941.41	8	1934.73	2047.64	101.11	21.22	8	1869.45	1894.33	14.13	81.48
			5755	27	27	312.59	27	300.55	27	320.62	457.98	68.11	61.00	27	298.99	304.58	4.35	210.35
			8847	18	18	650.65	18	460.05	18	444.33	524.01	46.31	44.56	18	284.15	337.05	46.65	153.64
			10027	16	16	923.23	16	538.89	16	566.00	604.21	63.30	67.98	16	381.15	444.75	35.91	196.29
			10743	15	15	1138.43	15	690.81	15	759.36	822.21	54.34	83.47	15	394.24	474.27	52.89	210.08
			11378	14	14	988.74	14	395.41	14	546.86	600.73	51.74	86.42	14	286.60	323.97	16.54	166.81
			17067	9	9	492.40	9	186.28	9	220.71	287.40	35.69	30.25	9	89.88	157.24	60.98	166.21

Table 3. Comparison of IGA with other algorithms in terms of *OP* and average *RPI* for *SI_{Best}*

Problem	# of problems	IGA vs. MRSA		IGA vs. HIH		IGA vs. GA	
		<i>OP</i>	<i>RPI</i>	<i>OP</i>	<i>RPI</i>	<i>OP</i>	<i>RPI</i>
Case study	4	NA*	NA	NA	NA	100%	60%
Small	17	0%	0%	0%	0%	0%	0%
Medium	22	50%	14%	27%	5%	45%	9%
Large	18	100%	43%	100%	26%	100%	37%
Total	61	51%	19%	42%	10%	52%	18%

* Not available

Table A.1. The data for the case study

Task No.	Predecessor	Task time (Sec.)	Task No.	Predecessor	Task time (Sec.)
1	-	5.3	22	21, 26	11
2	1	3.1	23	-	4.9
3	2	6	24	23	4.5
4	3	7.1	25	19, 14, 22	7.2
5	2	6.5	26	-	10.3
6	5	7.1	27	25	2.4
7	4	11.1	28	26, 27	5.1
8	6	11.1	29	28	8.8
9	-	11.2	30	20, 26	15.7
10	-	3.3	31	30	11.9
11	10	12.6	32	31	5.3
12	11	15.5	33	31	4.9
13	-	3.6	34	32, 33	18.1
14	10	2.7	35	34	13.5
15	10	16.5	36	35	3.2
16	9	9.9	37	32, 33	2.7
17	10	8.5	38	37	4.5
18	17	4.4	39	-	3.3
19	10	2.7	40	38	4.7
20	14, 19	9.4	41	40	8.4
21	17	8.9

Figure 1. The flow diagram of IGA for SALBP-1