Solving Assembly Line Balancing Problems with Fish School Search algorithm

Isabela M. C. de Albuquerque,
João Monteiro Filho,
Fernando Buarque de Lima Neto and
Alany Maria de Oliveira Silva
Department of Computer Engineering
Polytechnical School of Pernambuco
Recife, Brazil
Email: {imca,jbmf,fbln,amos}@ecomp.poli.br

Abstract—**Assembly lines constitute the main production paradigm of the contemporary manufacturing industry. Thus, many optimization problems have been studied aiming to improve the efficacy of its use. In this context, the problem of balancing an assembly line plays a key role. This problem is of combinatorial nature and also NP-Hard. For this reason, many researchers on computational intelligence and industrial engineering have been conceiving algorithms for tackling many versions of assembly line balancing problems using different procedures. In this paper, the Fish School Search algorithm and a variation of it that incorporates a routine to avoid stagnation of the search process were applied in order to solve the Simple Assembly Line Balancing Problem-type 1. The results were compared with an exact solution procedure named SALOME and also with the Particle Swarm Optimization algorithm. Both proposed procedures were able to achieve good results and the stagnation avoidance routine incorporated to FSS allowed more uniform distributions of tasks among workstations in the assembly line and converged faster to optimal solutions.**

1. Introduction

As the main production system of the current manufacturing industry, the assembly line has its invention credited to Henry Ford, who in 1913 started the serial production of Ford automobiles. This innovative solution is known to be one of the most important technical innovations of the industrial age [1]. Thanks to the use of assembly lines, the production cost of an automobile has decreased drastically. Consequently, final customer prices decreased and the market experimented a considerable sales improvement [2].

Formally, an assembly line is a flow based production system in which work units, denominated workstations, are disposed in a serial manner [3]. Work pieces travel along the line moved by a transport system. One of the main elements related with an assembly line productivity is the distribution of assembly tasks on each workstation, i.e. balancing the assembly line. The set of problems regarding balancing assembly lines is known as the Assembly Line Balancing Problem (ALBP) family.

It is important to highlight that all relevant problem versions of the ALBP family are known to be combinatorial NP-Hard problems [4]. Hence, there has been a relevant research effort performed in order to solve these problems with metaheuristic procedures such as Genetic Algorithms [5] and Ant Colony Optimization [6]. A detailed review of problem versions and solutions procedures is provided by Sivasankaran and Shahabudeen [7].

From the literature reviewed, it is noticeable that mainly discrete domain metaheuristic approaches such as the aforementioned Genetic and Ant Colony algorithms have been employed in order to solve ALBP instances. The application of continuous optimization procedures such as Particle Swarm Optimization is rare in this particular family of problems [8]. Hence, in this work we intend to verify the effectiveness in the application of a continuous optimization technique in order to solve combinatorial discrete optimization tasks. This is important once a validation of this sort of approach would allow more general applications of metaheuristic procedures.

Fish School Search (FSS) algorithm embeds an inherent capability of automatically to choose if the search will be focused in an exploration or exploitation process [9]. Therefore, we verify if this specific technique is an effective option to solve ALBPs once the mapping procedures applied in order to convert continuous position arrays into discrete permutations use to generate plateaus in the search space.

In addition, a novel version of FSS, named FSS-Stagnation Avoidance Routine (FSS-SAR), is also applied in the solution of balancing problems. This version incorporates a mechanism responsible to enhance FSS's capability of avoiding that searching entities get trapped in plateaus during the search process [10]. Either the original (named Vanilla) and SAR versions were applied to solve three instances of the selected ALBP version with different complexity levels. Results obtained were validated using the exact solution procedure SALOME, a Branch and Bound technique developed for solving ALBPs, and also compared with the results achieved using the Particle Swarm Optimization (PSO) algorithm [11].

This paper is organized as follows: in Section 2 we provide some definitions and a brief description of the state-of-the-art of the Simple Assembly Line Balancing Problem. In Section 3 FSS algorithm and its operators are described. In Section 4 the solution procedure proposed in this paper is described. Section 5 provides details about parameters selection and Section 6 describes the results obtained. Section 7 presents conclusion and future works.

2. Simple Assembly Line Balancing Problem

In general, the problem regarding partitioning assembly work among workstations while optimizing some performance criteria is defined as an ALBP [4]. This assignment problem is normally limited in each station by the cycle time c which is defined as the maximum or mean time for each work cycle on a single station. Some ALBP different versions are very often tackled in literature [7]. A frequently studied family of ALBP due to its simplicity is the Simple Assembly Line Balancing Problem (SALBP). It can be defined as a member of ALBP in which [12]:

- There is no product variety: only one variant of a single homogeneous product is manufactured;
- There is no process alternatives: all operations are performed in a predetermined manner;
- The assembly line consists of stations arranged in a serial manner and being independent of each other, i.e., each workplace belongs to a single station;
- Static and deterministic task times;
- Indivisibility of tasks: an operation cannot be split up between multiple workplaces;
- One workplace per workstation: each workstation contains exactly one workplace.

SALBP is known to be a simplified balancing problem version due to many features of real assembly lines which are not taken into account [13]. However, there is a relevant research effort in recent years in order to tackle this problem with different approaches. Metaheuristic procedures have been playing an important role in its solution. In the field of Evolutionary Computation, Differential Evolution [14], [15], [16], Genetic Algorithms [17], [18] and Genetic Programming [19] procedures were applied. Swarm intelligence approaches were utilized with modified versions of Particle Swarm Optimization [20] and Ant Colony Optimization [21], [22]. Furthermore, specialist heuristics [23], [24] and exact procedures [25], [26] were proposed recently and stability analysis was studied [27], [28]. The design of effective priority rules was described in the work of Otto and Otto [29].

SALBP family contains four different problem versions: SALBP-1 minimizes the number of stations for a given cycle time; SALBP-2 minimizes the cycle time for a given number of stations; SALBP-E is a multi-objective version that tries to simultaneously optimize both number of stations and cycle time; SALBP-F consists on finding a feasible assignment given cycle time and number of stations. The present work is intended to solve SALBP-1.

3. Fish School Search Algorithm - Vanilla Version

First proposed by Bastos-Filho and Lima-Neto [9], FSS is a population based search algorithm inspired in the behavior of swimming fishes that expand and contract while looking for food. Each fish n-dimensional location represents a possible solution for an optimization problem. The algorithm makes use of a new feature named weight for all fish which represents cumulative account on how successful has been the search for each fish in the school. FSS is composed of Feeding and Movement operators, the latter being divided into three sub-components. The first one is the Individual component of the movement, in which every fish in the school performs a random move looking for promising regions in the search space. This component is computed using the following equation:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \textbf{rand(-1,1)}step_{ind}, \tag{1}$$

where $\mathbf{x}_i(t)$ and $\mathbf{x}_i(t+1)$ represent the position of a fish i before and after the individual moment operator, respectively. **rand(-1,1)** is a n-dimensional vector which components are uniformly distributed random numbers varying from -1 up to 1. $step_{ind}$ is a parameter that defines the maximum displacement allowed in this movement. A new position $\mathbf{x}_i(t+1)$ is only accepted if fitness improves as position changes. If it is not the case, the fish remains in the same position and $\mathbf{x}_i(t+1) = \mathbf{x}_i(t)$.

After computing the Individual component for each fish, the next operator to be applied is the Feeding. This is performed in order to update fishes weights W according to:

$$W_i(t+1) = W_i(t) + \frac{\Delta f_i}{max(|\Delta f|)}, \tag{2}$$

where $W_i(t)$ is the weight parameter for fish i, Δf_i is the fitness variation between the last and new positions and $max(|\Delta f|)$ represent the maximum fitness variation among all fishes in the school. W is only allowed to vary between 1 and W_{scale}, which is an user-defined parameter. Weights of all fishes are initialized with value $W_{scale}/2$. After feeding all fishes, the Collective-Instinctive component of the movement is computed. It consists of an average of individual movements and it is calculated based on the following equation:

$$\mathbf{I} = \frac{\sum_{i=1}^{N} \Delta\mathbf{x}_i \Delta f_i}{\sum_{i=1}^{N} \Delta f_i}. \tag{3}$$

Vector $\mathbf{I}$ represents the weighted displacements average of each fish. It means that fishes which experienced a higher improvement will attract other fishes into its position. After vector $\mathbf{I}$ computation, every fish will be encouraged to move according to:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{I}. \tag{4}$$

At last, the Collective-Volitive component of the movement is calculated. This component is responsible to regulate the exploration/exploitation school ability during the search process. First of all, school's barycenter $\mathbf{B}(t)$ is calculated according to:

$$\mathbf{B}(t) = \frac{\sum_{i=1}^{N} \mathbf{x}_i(t) W_i(t)}{\sum_{i=1}^{N} W_i(t)}, \tag{5}$$

and then, if school total weight $\sum_{i=1}^{N} W_i$, where N is the school size, has increased from the last to current iteration, fishes are attracted to barycenter and $\pm$ sign in equation 6 becomes a $-$. If school total weight has not improved, fishes are spread away from barycenter and $\pm$ sign becomes a $+$.

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) \pm step_{vol}\mathbf{rand(0,1)}* \frac{\mathbf{x}_i(t) - \mathbf{B}(t)}{distance(\mathbf{x}_i(t), \mathbf{B}(t))}. \tag{6}$$

$step_{vol}$ defines the maximum displacement performed size. $distance(\mathbf{x}_i(t), \mathbf{B}(t))$ is the euclidean distance between fish i position and school barycenter. **rand(0,1)** is a n-dimensional vector which components are random numbers uniformly distributed in the interval $[0; 1]$.

The parameters $step_{ind}$ and $step_{vol}$ decay linearly along with the number of iterations.

4. Solving SALBP-1 with FSS

The main issues regarding the solution of SALBP-1 using FSS are related to solution representation scheme and treatment of capacity and precedence constraints. These issues were tackled according to procedures commonly utilized in literature [30] and are summarized as:

1) **Solution representation:** it was chosen the Task-Oriented representation, which means that a solution is represented as an array with size equivalent to the number of assembly tasks to be allocated. Every array position contain a task index. However, in FSS search procedure, a fish position is represented by an array of real numbers. For this reason, the solution proposed in this work uses the random-keys [30] procedure to map a fish position array into a tasks indexes array. This procedure maps the smallest value in the position array of a fish into the number 1, the second smallest value into the number 2, and so on. This is repeated until all array values are mapped into a task index;
2) **Constraints treatment:** as mentioned before, two different types of constraints have to be taken into account in order to solve SALBP-1 instances. First of them is the precedence constraint. We applied the same procedure defined by Hamta et al. [30]. It receives a task indexes array, and based on a precedence graph corrects the sequence to be ordered in a sense that no successor precedes a predecessor in the array returned. After that, capacity constraint should be dealt with. To do so, we use the following procedure: a workstation is opened and then the maximum number of tasks in the array with total time smaller than cycle time is assigned to the opened workstation. After that, this workstation is closed and a new one is then opened. This procedure is repeated until all tasks are assigned to a workstation. A detailed explanation of capacity constraint treatment in Task-Oriented approach is provided by Scholl [12].

The methodology employed consists in running FSS in its common form and all the mapping, correction and assignment procedures take place when fitness calculations are demanded. A pseudocode for SALBP-1 solution with FSS is represented in the following:

```
 1: Initialize user parameters;
 2: Initialize fishes positions randomly;
 3: while Stopping condition is not met do
 4:     Run Individual component of the movement;
 5:     Run Feeding operator;
 6:     Run Collective-instinctive component of the movement;
 7:     Run Collective-volitive component of the movement;
 8:     for Each fish in the school do
 9:         Map fish position into a task indexes array;
10:         Correct indexes order according to the precedence graph;
11:         Assign tasks into workstations using cycle time constraint;
12:         Measure balancing fitness;
13:     end for
14: end while
```

4.1. Increasing Exploration in the Search Space

SALBP-1 utilizes the number of workstations as objective function. It implies that the search space is composed of many plateaus, once many solutions will map task assignments with the same number of stations. Although FSS is able to deal with search spaces containing plateaus due to its capability of automatically to regulate the exploit/explore behavior of the search entities, this fact can represent a disadvantage in the use of FSS in order to tackle SALBP-1, ever since the algorithm utilizes improvement information to guide the search process. This disadvantage mainly has effect in the Individual component of the movement. Once it only allows fish to move if there is fitness improvement, in case some fish is on a big plateau in the search space, it will get trapped there. Two decisions were made in order to tackle this issue. The first one is to change the objective function of SALBP-1. Instead of the common use of the number of workstation, we use:

$$minimize \left(m \times \sqrt{\sum_{k=1}^{m} (c - t_k)^2} \right), \tag{7}$$

where m is the number of stations, c is the cycle time and t_k is the workload at station k. This objective function simultaneously improves number of workstations and Smoothness of the line balancing and also modifies the search space allowing more variation in fitness values when fishes move. The Smoothness measures the uniformity of tasks distribution along workstations [12].

In addition, a new version of FSS was used to solve SALBP-1. In the Vanilla version the Individual component of the movement is only allowed to move a fish if it improves its fitness value. However, in a very smooth search space, there would be many moving trials with no success and the algorithm could fail to converge. To solve this, in the version of FSS proposed by Monteiro et al. [10], authors introduce a parameter α for which $0 \leq \alpha \leq 1$. Values for α decay exponentially along with the number of iterations and measure the probability of a worsening allowance. It means that, every time a fish tries to move to a position that does not improve its fitness in the Individual component of the movement, a random number is chosen and if it is smaller than α the movement is allowed. In this work α decays exponentially according to $\alpha = 0.8e^{-0.007currentIteration}$. In addition, in this version only fish which have improved their fitness contributes to vector **I** computation.

5. Parameters Selection

Based on the literature reviewed, this is the first application of FSS for solving SALBP-1. Thus, there is no reference indicating the parameters set in order to apply FSS in that specific problem. The methodology chosen to select parameters values for FSS-Vanilla and FSS-SAR was inspired in the Factorial analysis technique with two levels. It means that 3 parameters, $Step_{ind}$, $Step_{vol}$ and W_{scale} had two values defined, being the smallest of these values labeled as low level and the highest as high level. Considering the present study, this methodology application results in $2^3 = 8$ different combinations of values of the 3 selected parameters. These 3 parameter affect the exploratory ability of FSS and tuning them is a relevant issue to be solved in order to guarantee convergence to good results. High and low levels for each parameters were chosen based on values used by Bastos-Filho and Lima-Neto [9]. High values for $Step_{ind}$ and $Step_{vol}$ were set as 10% of search space width and low values were 1% and 0.01%, respectively. The considered search space was $[-100; 100]^n$ where n is the number of tasks. Table 1 summarizes the values used for each variable.

TABLE 1. HIGH AND LOW VALUES USED FOR EACH PARAMETER

Level	W_{scale}	$Step_{ind}$	$Step_{vol}$
High	10000	20	20
Low	1000	2	0.2

The 8 sets of parameters generated from the combination of high/low levels of the 3 parameters to tune were tested with a benchmark instance of SALBP-1 with a high level of difficult. In this case, level of difficult is related to the number of tasks that will be balanced and precedence graph complexity. We have chosen a difficult instance in order to generalize this parameters choice for problems with lower levels of difficult. All test instances used in this work were taken from **http://assembly-line-balancing.de**. The minimum number of stations for each of these instances is already known from the solution of those instances using an exact procedure called SALOME [12]. For this stage we used the instance referred as n = 100_34 and its minimum number of stations is 15. In order to choose the parameters we have run SALBP-1 with both versions of FSS 30 times. Table 2 summarizes configurations tested. An ID is assigned to each different configuration and here FSS-Vanilla is referred as FSS-V.

TABLE 2. CONFIGURATIONS OF THE TESTS EXECUTED

ID	Version	W_{scale}	$Step_{ind}$	$Step_{vol}$
1	FSS-V	Low	Low	Low
2	FSS-V	High	Low	Low
3	FSS-V	Low	High	Low
4	FSS-V	High	High	Low
5	FSS-V	Low	Low	High
6	FSS-V	High	Low	High
7	FSS-V	Low	High	High
8	FSS-V	High	High	High
9	FSS-SAR	Low	Low	Low
10	FSS-SAR	High	Low	Low
11	FSS-SAR	Low	High	Low
12	FSS-SAR	High	High	Low
13	FSS-SAR	Low	Low	High
14	FSS-SAR	High	Low	High
15	FSS-SAR	Low	High	High
16	FSS-SAR	High	High	High

Considering number of stations (ST), which is the main objective of SALBP-1, all configurations described in Table 2 were able to find the optimum solution in at least one of the 30 runs. For a more detailed analysis of which configurations and version of FSS is better, we also considered the Smoothness (SI) achieved by each parameter configuration. In addition, we also analyze the convergence for all configurations. In order to do so, we defined that the algorithm converged if solutions generated caused no improvement higher than 0.0001. The number of iterations until convergence (IUC) is defined as the iteration where occurred the last improvement. IDs of configurations that achieved minimum values for mean, standard deviation, best and worst values found for each criterion considering 30 executions are shown in Table 3.

TABLE 3. RESULTS OBTAINED DURING THE PARAMETERS SELECTION STAGE

Criteria	Mean	Std. Dev.	Best	Worst
ST	3, 4, 12, 15	3, 4, 12, 15	All	3, 4, 12, 15
Smoothness	12	12	3	5
IUC	13	13	8	12

From Table 3 it is possible to notice that most of configurations that have achieved the minimum values uses at least one of the steps in the high value. This characteristic approximates FSS-Vanilla and FSS-SAR to a random search process. As the idea of this work is evaluate efficacy on the use of both versions of FSS to solve SALBP-1, we decided to use a configuration that presented good results but also have steps parameters at low values, even if this choice did not provide the best results. Considering the idea mentioned above, the selected IDs were 2 and 10. Then, the resulting choice is $W_{scale} = 10000$, $Step_{ind} = 2$ and $Step_{vol} = 0.2$.

6. Tests and Results

In order to evaluate FSS-Vanilla and FSS-SAR performance in solving SALBP-1, we compared the results for number of stations obtained with SALBP-1 solution using PSO and optimum values obtained with the exact procedure SALOME. Thus, we used these optimum values to check whether FSS-Vanilla, FSS-SAR and PSO were able to achieve optimum results.

In the PSO version chosen [31], for a particle $\mathbf{x}_i$ its position in iteration $t+1$ is defined by:

$$\begin{aligned}\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \chi[\mathbf{v}_i + c_1 r_1(\mathbf{pb}_i - \mathbf{x}_i(t)) +\\ c_2 r_2(\mathbf{gb}_i - \mathbf{x}_i(t))],\end{aligned} \tag{8}$$

$$\chi = \frac{2}{|2-(c_1+c_2)-\sqrt{(c_1+c_2)((c_1+c_2)-4)}|}. \tag{9}$$

χ is known as Constriction Factor and r_1 and r_2 are uniformly distributed random numbers in the interval $[0;1]$. In this version c_1 and c_2 must satisfy $c_1 + c_2 \geq 4$. For this work we have chosen $c_1 = c_2 = 2.1$. Solution of SALBP-1 with PSO follows the same flow as for FSS, as described in section 4.

In order to validate SALBP-1 solution with FSS-Vanilla and FSS-SAR, both algorithms and PSO were used to solve three different instances of SALBP-1. The same instance used during parameters selection, now referred to as Large instance, was solved and in addition two more instances from the same repository were selected. These two new test cases are less complex than the larger one and are referred to as Small and Medium instances.

The minimum number of stations for the Small instance is 12 and 6 for the Medium one. All considered instances of SALBP-1 were solved using FSS-Vanilla, FSS-SAR and PSO with the parameters defined later and each test case was repeated 450 times. As in the parameter selection for FSS-Vanilla and FSS-SAR, number of stations, Smoothness and also the result of the convergence analysis, represented by IUC, were used as criteria to determine if FSS-Vanilla and FSS-SAR are able to obtain good solutions for SALBP-1.

The 450 results for each test case were grouped in samples of size 15 and the means of those were considered as input data for the statistical analysis, which results in 30 samples per algorithm for each instance.

In order to decide if a parametric or nonparametric statistical test is going to be taken, we applied the Shapiro-Wilk test [32] and then concluded that normality is guaranteed for the results obtained in all test cases. Thus, we decided for a parametric test and choose to apply the widely used one-way Analysis of Variance (ANOVA) technique [33] to conclude whether the results obtained are significantly different from each other and, if so, which algorithm is better for each criterion considered.

All the procedures tested were able to achieve the optimum number of stations in all test instances considered. Thus, we only analyzed in details the other criteria mentioned: Smoothness an IUC.

For all test instances, an one-way ANOVA with 95% of confidence was performed. Calculated degrees of freedom were $v_1 = 2$ and $v_2 = 87$, thus $F_{ref} = 4.89$.

6.1. Results for Smoothness

The values of $F_{calculated}$ for each ANOVA considering Smoothness values obtained for all test instances are shown in Table 4.

TABLE 4. ANOVA $F_{calculated}$ VALUES FOR SMOOTHNESS RESULTS

Criterion	Small	Medium	Large
Smoothness	64,01	91,42	561,81

From Table 4, as all values of $F_{calculated}$ were greater than F_{ref}, it is possible to conclude that at least one technique performance regarding the Smoothness index for all test instances is statistically different from the others. However, it is not possible to conclude yet which was the best performer.

In order to make a more detailed analysis and discover which technique performed better than the others, we calculate confidence intervals of the mean with a confidence level of 95% using the pooled standard deviation for the three test instances. Confidence intervals for Small, Medium and Large instances are shown in Figures 1, 2 and 3, respectively.

From Figures 1, 2 and 3 it is possible to notice that FSS-SAR has improved Smoothness values obtained with the Vanilla version, as we expect. For the Small and Medium instances, results obtained with FSS-SAR were statistically indistinguishable from the ones obtained with PSO, although in the Large instance PSO shows better results for this criterion.

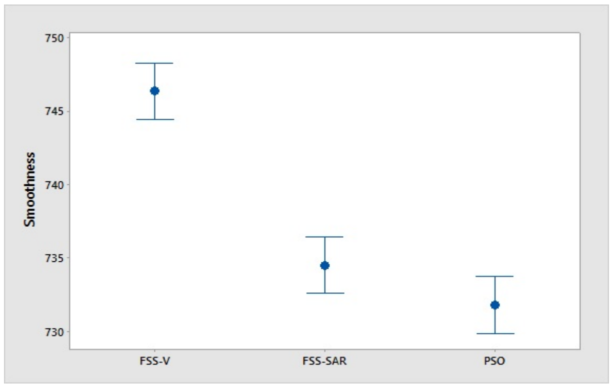

Figure 1. Pooled confidence intervals of Smoothness values obtained for the Small test set

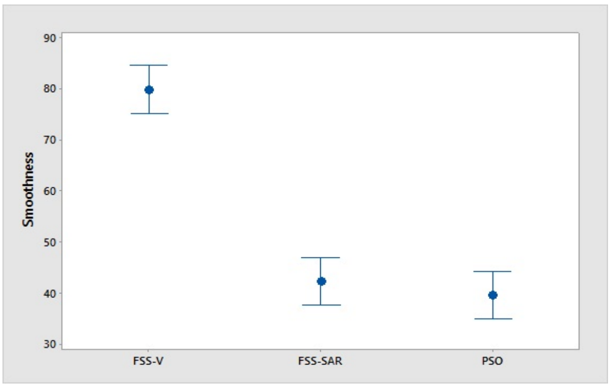

Figure 2. Pooled confidence intervals of Smoothness values obtained for the Medium test set

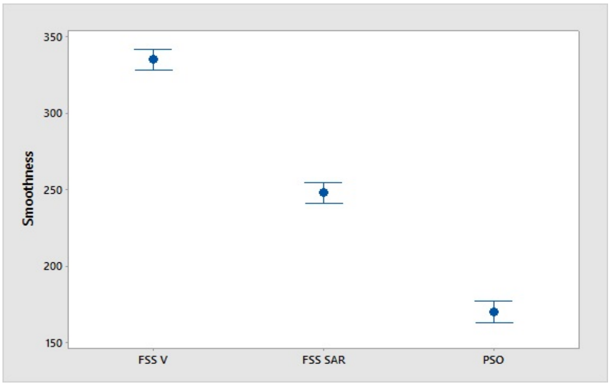

Figure 3. Pooled confidence intervals of Smoothness values obtained for the Large test set

6.2. Convergence analysis

Values of $F_{calculated}$ for each ANOVA considering the IUC values obtained for all test instances are shown in Table 5.

TABLE 5. ANOVA $F_{calculated}$ VALUES FOR IUC RESULTS

Criterion	Small	Medium	Large
IUC	172,48	104,8	41,66

From Table 5, as all values of $F_{calculated}$ were greater than F_{ref}, it is possible to conclude that at least one technique performance regarding the IUC criterion for all test instances is statistically different from the others. As we did for Smoothness, we calculate confidence intervals of the mean with a confidence level of 95% using the pooled standard deviation for the three test instances in order to make a more detailed analysis and discover which technique performed better than the others. Confidence intervals for Small, Medium and Large instances are shown in Figures 4, 5 and 6, respectively.

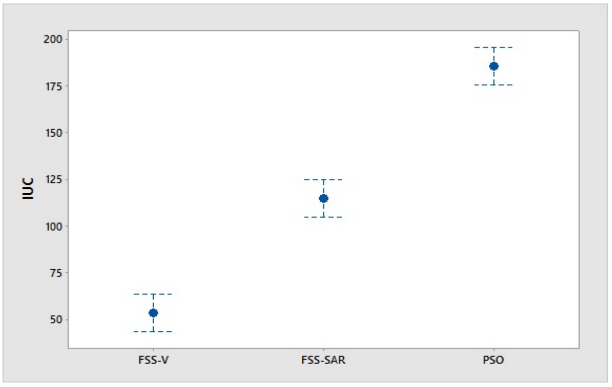

Figure 4. Pooled confidence intervals of IUC values obtained for the Small test set

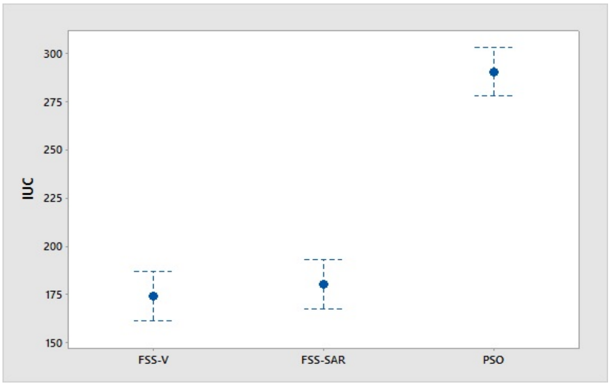

Figure 5. Pooled confidence intervals of IUC values obtained for the Medium test set

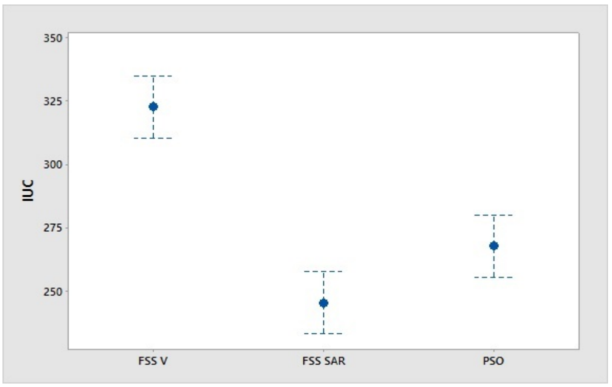

Figure 6. Pooled confidence intervals of IUC values obtained for the Large test set

Observing Figures 4, 5 and 6 it is possible to notice that, for the Small instance, FSS-Vanilla converged faster than FSS-SAR and PSO. In the Medium instance, both FSS-Vanilla and FSS-SAR converged faster than PSO, and it is not possible to tell which one, FSS-Vanilla or FSS-SAR, converged faster. At last, for the Large instance, FSS-SAR converged faster than FSS-Vanilla and PSO.

Summarizing, considering the Small and Medium instances, FSS-SAR achieved better values for Smoothness than FSS-Vanilla and statistically indifferent results in comparison with PSO. Furthermore, for the Small instance, FSS-Vanilla converged faster than FSS-SAR and PSO, but the Smoothness values achieved by this technique were worse than the obtained with FSS-SAR and PSO, thus, it seems that FSS converged to local optima. For the Medium instance, FSS-SAR converged faster than both FSS-Vanilla and PSO, although PSO achieved best values for Smoothness, hence, we can conclude that, in this case, FSS-SAR converged to local optima. For the Large instance, PSO achieved best values for Smoothness and it took a fewer number of iterations than FSS-Vanilla to converge.

7. Conclusion

This paper main contribution is the evaluation of the effectiveness in the application of a continuous optimization technique in order to solve the discrete combinatorial problem named Simple Assembly Line Balancing Problem-type 1. Two different versions of the Fish School Search algorithm were employed and compared against results obtained using an exact procedure named SALOME and Particle Swarm Optimization.

Considering the results obtained, it is possible to conclude that both versions of FSS were successful, once both were able to achieve the minimum number of stations for three different problem instances.

A more detailed analysis has shown that FSS-SAR presented better results considering two other criteria: Smoothness and IUC. We applied an one-way ANOVA in order to guarantee the reliability of the conclusions taken from the resulting data of the experiments performed. Thus, we can conclude that: (i) in comparison with FSS-Vanilla, FSS-SAR achieved better results for Smoothness but, as we expected, in most of the problems this version demands a greater IUC to be able to improve the results; (ii) in comparison with PSO, FSS-SAR converges faster and, except for the Large instance tested, it has achieved Smoothness values as good as the ones obtained with PSO.

As future work we propose to tackle other ALBP versions with FSS, including multi-objective variations of ALBP such as SALBP-E. Moreover, the present work could be extended by the comparison of FSS performance on solving SALBP-1 with larger data-sets and comparing its results with other metaheuristic approaches such as Genetic Algorithms and Ant Colony Optimization.

Acknowledgments

The authors thank to CAPES (Coordination for the Improvement of Higher-Education Personnel), Brazil, for partial financial support for this paper.

References

[1] Q. Tu, M. A. Vonderembse, and T. Ragu-Nathan, “The impact of time-based manufacturing practices on mass customization and value to customer,” *Journal of Operations management*, vol. 19, no. 2, pp. 201–217, 2001.

[2] “Ford-Timeline,” http://corporate.ford.com/company/history.html, accessed in: 2016-06-15.

[3] N. Boysen, M. Fliedner, and A. Scholl, “Assembly line balancing: Which model to use when?” *International Journal of Production Economics*, vol. 111, no. 2, pp. 509–528, 2008.

[4] C. Becker and A. Scholl, “A survey on problems and methods in generalized assembly line balancing,” *European Journal of Operational Research*, vol. 168, no. 3, pp. 694–715, 2006.

[5] Ö. Mutlu, O. Polat, and A. A. Supciller, “An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-ii,” *Computers & Operations Research*, vol. 40, no. 1, pp. 418–426, 2013.

[6] M. Chica, O. Cordón, S. Damas, and J. Bautista, “A new diversity induction mechanism for a multi-objective ant colony algorithm to solve a real-world time and space assembly line balancing problem,” *Memetic computing*, vol. 3, no. 1, pp. 15–24, 2011.

[7] P. Sivasankaran and P. Shahabudeen, “Literature review of assembly line balancing problems,” *The International Journal of Advanced Manufacturing Technology*, vol. 73, no. 9-12, pp. 1665–1694, 2014.

[8] A. C. Nearchou, “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, 2011.

[9] C. J. A. B. Filho, F. B. D. L. Neto, A. J. C. C. Lins, A. I. S. Nascimento, and M. P. Lima, “A novel search algorithm based on fish school behavior,” *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, pp. 2646–2651, 2008.

[10] J. B. Monteiro, I. M. C. Albuquerque, F. B. L. Neto, and F. V. S. Ferreira, “Optimizing multi-plateau functions with FSS-SAR (Stagnation Avoidance Routine),” *IEEE Symposium Series on Computational Intelligence*, 2016.

[11] R. C. Eberhart, J. Kennedy *et al.*, “A new optimizer using particle swarm theory,” in *Proceedings of the sixth international symposium on micro machine and human science*, vol. 1. New York, NY, 1995, pp. 39–43.

[12] A. Scholl, *Balancing and sequencing of assembly lines*. Physica-Verlag Heidelberg, 1999.

[13] J. Sternatz, “Enhanced multi-Hoffmann heuristic for efficiently solving real-world assembly line balancing problems in automotive industry,” *European Journal of Operational Research*, vol. 235, no. 3, pp. 740–754, 2014.

[14] A. Mozdgir, I. Mahdavi, I. S. Badeleh, and M. Solimanpur, “Using the Taguchi method to optimize the differential evolution algorithm parameters for minimizing the workload smoothness index in simple assembly line balancing,” *Mathematical and Computer Modelling*, vol. 57, no. 1-2, pp. 137–151, 2013.

[15] R. Pitakaso, P. Parawech, and G. Jirasirierd, “Comparisons of different mutation and recombination processes of the dea for salb-1,” in *Proceedings of the Institute of Industrial Engineers Asian Conference 2013*. Springer, 2013, pp. 1571–1579.

[16] R. Pitakaso, “Differential evolution algorithm for simple assembly line balancing type 1 (SALBP-1),” *Journal of Industrial and Production Engineering*, vol. 32, no. 2, pp. 104–114, 2015.

[17] T. Al-Hawari, M. Ali, O. Al-Araidah, and A. Mumani, “Development of a genetic algorithm for multi-objective assembly line balancing using multiple assignment approach,” *The International Journal of Advanced Manufacturing Technology*, vol. 77, no. 5-8, pp. 1419–1432, 2014.

[18] C. G. S. Sikora, T. C. Lopes, H. Silv, and L. Magat, “Genetic algorithm for type-2 assembly line balancing,” in *2015 Latin America Congress on Computational Intelligence (LA-CCI)*, vol. 41, 2015, pp. 1–6.

[19] A. Baykasoglu and L. Ozbakir, “Discovering task assignment rules for assembly line balancing via genetic programming,” *International Journal of Advanced Manufacturing Technology*, vol. 76, no. 1-4, pp. 417–434, 2014.

[20] J. Dou, J. Li, and C. Su, “A novel feasible task sequence-oriented discrete particle swarm algorithm for simple assembly line balancing problem of type 1,” *International Journal of Advanced Manufacturing Technology*, vol. 69, no. 9-12, pp. 2445–2457, 2013.

[21] Q. X. Zheng, Y. X. Li, M. Li, and Q. H. Tang, “An improved ant colony optimization for large-scale simple assembly line balancing problem of type-1,” *Applied Mechanics and Materials*, vol. 159, pp. 51–55, 2012.

[22] Y. G. Zhong and B. Ai, “A modified ant colony optimization algorithm for multi-objective assembly line balancing,” *Soft Computing*, 2016.

[23] M. C. D. O. Moreira, C. Miralles, and A. M. Costa, “Assembly Line Worker Integration and Balancing Problem,” *Anais do XLIV Simpósio Brasileiro de Pesquisa Operacional*, vol. 54, pp. 54–65, 2012.

[24] T. Pape, “Heuristics and lower bounds for the simple assembly line balancing problem type 1: Overview, computational tests and improvements,” *European Journal of Operational Research*, vol. 240, no. 1, pp. 32–42, 2015.

[25] M. Vilà and J. Pereira, “An enumeration procedure for the assembly line balancing problem based on branching by non-decreasing idle time,” *European Journal of Operational Research*, vol. 229, no. 1, pp. 106–113, 2013.

[26] M. Ritt and A. M. Costa, “Improved integer programming models for simple assembly line balancing and related problems,” *International Transactions in Operational Research*, vol. 00, pp. 1–15, 2015.

[27] E. Gurevsky, O. Battaïa, and A. Dolgui, “Stability measure for a generalized assembly line balancing problem,” *Discrete Applied Mathematics*, vol. 161, no. 3, pp. 377–394, 2013.

[28] Y. N. Sotskov, A. Dolgui, T. C. Lai, and A. Zatsiupa, “Enumerations and stability analysis of feasible and optimal line balances for simple assembly lines,” *Computers and Industrial Engineering*, vol. 90, no. m, pp. 241–258, 2015.

[29] A. Otto and C. Otto, “How to design effective priority rules: Example of simple assembly line balancing,” *Computers and Industrial Engineering*, vol. 69, no. 1, pp. 43–52, 2014.

[30] N. Hamta, S. M. T. Fatemi Ghomi, F. Jolai, and M. Akbarpour Shirazi, “A hybrid PSO algorithm for a multi-objective assembly line balancing problem with flexible operation times, sequence-dependent setup times and learning effect,” *International Journal of Production Economics*, vol. 141, no. 1, pp. 99–111, 2013.

[31] M. Clerc and J. Kennedy, “The particle swarm-explosion, stability, and convergence in a multidimensional complex space,” *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 1, pp. 58–73, 2002.

[32] N. M. Razali and Y. B. Wah, “Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests,” *Journal of Statistical Modeling and Analytics*, vol. 2, no. 1, pp. 21–33, 2011.

[33] B. J. Winer, D. R. Brown, and K. M. Michels, *Statistical Principles in Experimental Design*, vol. 2.