Local Search Heuristics for the Assembly Line Balancing Problem with Incompatibilities Between Tasks[*]

Joaquin Bautista, Raúl Suárez, Manuel Mateo, Ramón Companys

Institut d'Organització i Control de Sistemes Industrials (UPC)
Diagonal 647, planta 11, 08028 Barcelona, Spain. Phone: +34-934016653, Fax: +34-934016605.
Contact emails: bautista@ioc.upc.es, suarez@ioc.upc.es, mateo@ioc.upc.es

Abstract

This paper deals with the Assembly Line Balancing Problem considering incompatibilities between the tasks with the aim of, first, minimizing the number of workstations and, then, minimizing the cycle time for the minimum number of workstations. In order to solve the problem we propose the use of a Greedy Randomized Adaptive Search Procedure (GRASP) obtained from the application of some classic heuristics, based on priority rules, and a genetic algorithm that searches for the solution in the heuristic space. A computational experience is included to illustrate the performance of the proposed approach.

1 Introduction

The Assembly Line Balancing Problem (ALBP) can be divided into two groups, according to the classification proposed by Baybars [1]: Simple Assembly Line Balancing (SALBP) and General Assembly Line Balancing (GALBP). The first group determines the tasks assigned to a set of workstations with the same cycle time; each task has a deterministic duration and must be performed in only one of the workstations, either by human operators or by robots. Two goals can be considered in addition to the precedence relations between the tasks: the minimization of the number of workstations for a given cycle time (SALBP-1) and the minimization of the cycle time for a given number of workstations (SALBP-2). Any other variation of the problem is included in the second group.

In this work, an extension of SALBP-1 is presented, considering incompatibilities between groups of tasks, so that if two tasks are incompatible they cannot be assigned to the same workstation. As a secondary goal, we try to reach the minimum cycle time once the number of workstations has been determined.

[*]This work was partially supported by the CICYT Projects TAP98-0471 TAP98-0494 and TAP99-0839.

For the exact solution of the SALBP a number of different procedures have been applied, such as for instance, branch and bound [2,3], but these are only useful for low dimension examples because the problem is NP-hard. On the other hand, for the high dimension examples found in industry, heuristic procedures are accepted [4], both for SALBP-1 [5] and for SALBP-2 [6].

SALBP-1 is typically solved with greedy heuristics, based on the application of priority rules to assign the tasks to the workstations. The rules consider such aspects as the duration of the tasks, the number of tasks after a given one, the constraints on the minimum number of workstations to fulfill the assign, etc., or even a mixture of these aspects. The rules are used to establish an ordered list with the possible tasks, so in each selection the most suitable task (according to the chosen rule) can be chosen. Usually, this type of heuristics considers only one rule that completely determines the sequence of tasks (except when the rule includes any random selection). The solutions obtained through this approach are acceptable, and when a rule considers a combination of several aspects it can even lead to better solutions. Nevertheless, it is not possible to conclude that any rule is better than the others for all the instances of the problem.

Another type of heuristics, which is useful for local search methods, includes Hill Climbing (HC), Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithms (GA) [7]. This type of heuristics searches for other solutions, in a given neighborhood, beyond the direct solutions obtained through the previously mentioned greedy heuristics. Though the definition of a neighborhood may be general and valid for any combinatorial problem, knowledge of the specific problem is lost, unlike in the case of greedy heuristics.

A third type of heuristics to reach good line balancing are Greedy Randomized Adaptive Search Procedures

(GRASP), which allows the introduction of random effects into greedy heuristics that are specially adapted to the problem. These procedures have already been successfully used for different applications in industrial engineering, for instance in a number of characteristic problems of combinatorial optimization [8].

In this work we proposed two different procedures that consider simultaneously:

1) the main positive aspect of the first type of heuristics: the knowledge of SALBP given by the specific priority rules of the problem;
2) the main positive aspect of the last two types of heuristics: the ability to generate a great number of solutions in the search space (in any combinatorial problem).

In the following sections the two proposed procedures are introduced: in Section 2, a GRASP generated from the classic heuristics in the literature, and in Section 3, a Genetic Algorithm as a representative heuristic for local searches. Section 4 presents a computational analysis and Section 5 summarizes some conclusions of the work.

2 Basic Greedy Heuristics and GRASP

Figure 1 shows the flow chart of an algorithm for the generation of solutions to SALBP. The schema is valid for both greedy heuristics and the GRASP heuristics based on them.

The greedy heuristics assign the task (box 5) with the best index value, obtained from the application of priority rule(s) among the set of tasks compatible with the previously assigned tasks, the precedence relations and time constraints. The list of indexed tasks is generated (box 3) while some tasks are still to be assigned (box 1) and they can be assigned to the open workstation (box 2).

In Figure 1 there are two main circuits. The first one, on the left, is formed when a new workstation must be opened in the assignment. When there are no tasks in the list of candidates (box 4) and not all the tasks have been assigned (box 1), a new station is added to the previous ones in the current solution (box 2) and, for these new conditions, the list of candidate tasks is refreshed (box 3). The second circuit, on the right, updates the solution once a task is assigned (box 5); this involves recalculating the available time at the current station (box 6) and determining the new constraints for that station (box 7).

A local search procedure to optimize the balancing (box 8) can then be applied to the obtained incumbent solution in order to optimize the definitive solution. In the selection of tasks for the indexed list of candidates, at least one priority rule is required to define a heuristic. A sample of 13 basic rules, widely used in the literature, is presented in the Appendix.

These main features of the classical greedy heuristics have been adapted in the GRASPs, random selections being introduced in the generation of solutions by using a probability distribution (updated at each task assignment). Usually, the list of candidate tasks is limited in order to enhance the probability of the most suitable candidate tasks.

In our approach this probability distribution depends on an index resulting from the priority rule considered, so we call the procedure Greedy Randomize Weighted Adaptive Search Procedure, GRWASP.

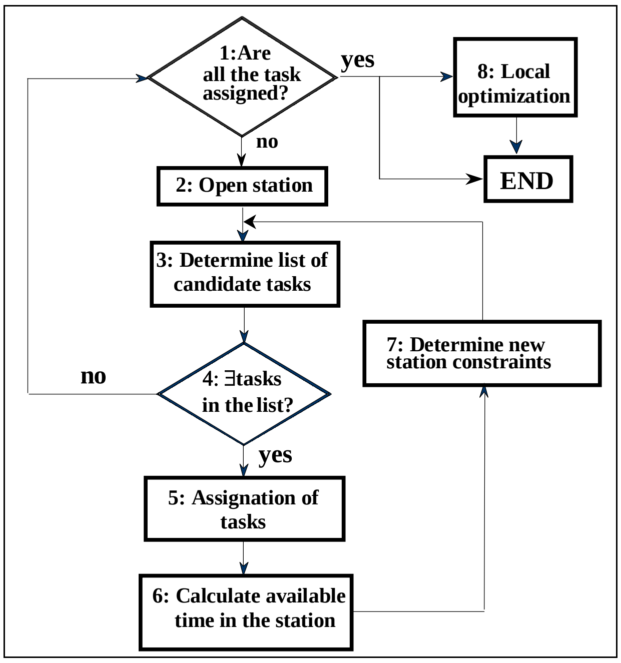
1:Are all the task assigned?
yes
8: Local optimization
no
2: Open station
END
3: Determine list of candidate tasks
7: Determine new station constraints
no
4: ∃tasks in the list?
yes
5: Assignation of tasks
6: Calculate available time in the station

Figure 1: Flow chart for greedy and GRASP approaches.

3 A Genetic Algorithm as a Local Search Heuristic

3.1. Neighborhoods in Local Search Methods

Local search methods are used to explore neighborhoods with “neighbor solutions”, a concept than can be defined in different ways. One way to define neighbor solutions of an initial solution is by characterizing the last one by a sequence of elements and, then, looking for other solutions by changing the order of elements in the sequence. For instance, the assignment of tasks to a set of workstations can be determined by the order in which the tasks must be executed, and altering the numbers in the sequence may lead to neighbors. The definition of neighborhoods is general and does not take advantage of the specific information about the problem to be solved, but it has the advantage of being applicable to any problem. Thus, knowledge of the problem can be used to create more suitable neighbors.

Storer [9] proposed other neighborhood definitions based on the relation between a heuristic h and the solution s obtained when h is applied to an instance of the problem p: $h(p) = s$. This relation allows the definition of neighborhoods both in the problem solution space and in the heuristic space.

In the solution space, the neighborhood is obtained by taking the following steps: 1) introducing random perturbations in the problem data (within reasonable bounds); 2) applying a heuristic procedure to obtain a solution; and 3) evaluating the solution with the original data.

In the heuristic space, two new variations on the available heuristics for SALBP are proposed:

1. Definition of a new hybrid rule as a weighted linear combination of the original set of rules ρ_i:
$$\rho = \sum_{\forall i} \pi_i \rho_i$$
2. Splitting the tasks to be assigned into subsets (usually called "windows") and using a particular rule for each subset. This allows the characterization of a solution by a vector of rules $r = (\rho_{[1]}, \rho_{[2]}, ..., \rho_{[N]})$; where N is the number of tasks and $\rho_{[k]}$ is the rule applied in the k assignment.

Thus, from the second approach, we can conclude that, given a vector of rules r and a procedure A, a heuristic can be defined as the pair (r, A): $h = h(r, A)$. In this way, all the heuristics resulting from the composition of a vector of rules and the procedure shown in Figure 1 are valid.

3.2. Application in a Genetic Algorithm

As a result of the above considerations, a genetic algorithm that includes the generation of solutions in the heuristic space is presented below.

Nomenclature:

I	number of individuals (vectors of rules) in the population.
L	number of iterations (generations).
p	instance of the problem to be solved.
Π_r	population of ancestors of the sequences of rules.
Π_h	population of ancestors of the heuristics.
Π_s	population of ancestors of the solutions.
Δ_r	population of descendants of the sequences of rules.
Δ_h	population of descendants of the heuristics.
Δ_s	population of descendants of the solutions.
Ω_r	population of eligible sequences of rules for the next iteration (generation).
r_i	element i of the sets Π_r , Δ_r , Λ_r and Ω_r.
h_i	element i of the sets Π_h , Δ_h and Λ_h.
s_i	element i of the sets Π_s , Δ_s and Λ_s.

Begin GA

Phase A. Initialization

0 Data initialization

0.1 Generate the initial populations Π_r with different homogeneous sequences of rules: $\Pi_r = \{ r_i = (\rho_{[1]},..., \rho_{[N]}) : \rho_{[1]} = .. = \rho_{[N]} \}$

0.2 Generate the initial population of heuristics: $\Pi_h = \{ h_i = h_i(r_i, A) : r_i \in \Pi_r \}$

0.3 Generate the population of solutions of p and evaluate their makespan:
$\Pi_s = \{ s_i = h_i(p) : h_i \in \Pi_h \}$

0.4 Determine the fitness f_j of the elements of Π_s as:
$$f_j = -NE_j + \left[\frac{\sum_{i=1}^{N} t_i}{C} \right]^+ + \frac{\sqrt{\sum_k^{NE_j} (c_k - C)^2}}{C\sqrt{NE_j}}$$
where:

t_i	duration of task i
C	cycle time
NE_j	number of stations in the j-th solution
c_k	occupied time in the k-th station ($1 \le k \le NE_j$)

0.5 Save as incumbent solution the pair (h^*, s^*) with greatest fitness

Phase B: Iterate through the following steps L times:

1. Selection of ancestors:
Build $I/2$ pairs of elements of Π_r according to the fitness of the elements of Π_s.
2. Choice of the pairs for the crossover:

2.1 Determine the probability of the current crossover: $P_c = P_c(\alpha)$ with
$$\alpha = \frac{1}{I(I-1)} \sum_{i=1}^{I} \frac{1}{N} \sum_{j \ne i} h_{i,j}$$

where $h_{i,j} \in \{0,1\} \wedge [\, h_{i,j} = 1 \Leftrightarrow \rho_{[k]} (\in r_i) = \rho_{[k]} (\in r_j) \; \forall k , 1 \le k \le N \,]$

```
    2.2 Assign a random number to each pair of rule sequences.
    2.3 Decide, for each pair of rule sequences, if a crossover should be performed according to the relation between the random number and the probability depending on population homogeneity $P_c$.
3   Generation of descendants:
    3.1 Crossover the selected pair of sequences of rules to generate two descendants, creating $\Delta_r$.
    3.2 Generate $\Delta_h$ and $\Delta_s$ from $\Delta_r$ as was done in 0.2 and 0.3 respectively.
    3.3 Determine the makespan of the elements of $\Delta_s$. If any element of $\Delta_s$ has a better makespan than the incumbent solution, then save as heuristic and incumbent solution the pair ($h^*$,$s^*$) associated with that element.
4   Mutation of descendants:
    4.1 Determine the probability of mutation of the current generation: $P_m = P_m(\alpha)$.
    4.2 Assign a random number to each element $\Delta_r$.
    4.3 Decide the elements of $\Delta_r$ to be mutated according to their random number and $P_m$.
    4.4 Mutate the chosen elements of $\Delta_r$ creating $\Lambda_r$.
    4.5 Generate $\Lambda_h$ and $\Lambda_s$ from $\Lambda_r$ as was done in 0.2 and 0.3 respectively.
    4.6 Determine the makespan of the elements of $\Lambda_s$. If any element of $\Lambda_s$ has a better makespan than the incumbent solution, then save as heuristic and incumbent solution the pair ($h^*$,$s^*$) associated with that element.
5.  Regeneration of the population:
    5.1 Build the population of eligible elements
        $\Omega_r \leftarrow \Pi_r + \Delta_r + \Lambda_r$
    5.2 Determine the fitness of the elements of the populations $\Delta_s$ and $\Lambda_s$ as was indicated in 0.5.
    5.3 Choose $I$ elements from $\Omega_r$ according to the fitness of the elements of $\Pi_s$ , $\Delta_s$ and $\Lambda_s$.
End GA
```

4 Computational analysis

The computational analysis was based on the following heuristics:

- 13 greedy (and deterministic) heuristics, found in the literature, corresponding to the rules in Appendix A.

- 6 traditional GRASP heuristics selected from the 13 rules mentioned above. The chosen GRASP heuristics were generated from the following rules:
  - Longest Processing Time (rule 1),
  - Greatest Ranked Positional Weight (rule 4),
  - Greatest Average Ranked Positional Weight (rule 5),
  - Greatest Processing Time divided by Upper Bound (rule 8),
  - Maximum Number of Successors divided by Slack (rule 11),
  - Bhattcharjee & Sahu (rule 12).

- The same 6 GRASP heuristics, but revised by introducing a selection probability for the task assignment proportional to the value of the parameter used in the rule. For instance, the value for the first rule is the longest processing time.

- 7 versions of the presented Genetic Algorithm, with different crossover and regeneration processes to generate solutions in the heuristic space. The probabilities of mutations and crossovers depend on the index of homogeneity in the population α, which is used as: $P_c = 1 - 0.5\alpha$ for crossovers, and $P_m = 0.05 + 0.95\alpha$ for mutations.

These procedures were used to solve 160 instances, divided into 4 groups of 40 instances with 20, 40, 60 and 80 tasks respectively. According to the order strength of Mastor and the Rachamadugu ratio of incompatibility, the values considered were between 0.05 and 0.75.

The results are illustrated in Figures 2 and 3. Figure 2 shows the number of times that the heuristics found the best solution (minimum number of workstations) in the 160 instances, and Figure 3 shows the ratio between the average of the values obtained by each procedure and the best obtained value.

Regarding the minimization of the number of workstations, the Genetic Algorithms were clearly better than the greedy heuristics based on priority rules without any local optimization. On average, the Genetic Algorithms and the proposed GRWASP (revised GRASP with weights) procedures achieve the best results, while the greedy heuristics produce the worst ones.

5 Conclusions

In this work a new approach to SALBP with incompatibilities between the tasks has been proposed. It includes the knowledge provided by the greedy (deterministic) heuristics of a given problem in the local search heuristics, such as HC, SA, TS, GA (in this paper a GA was presented) or in a GRASP. In this way, in the first case, a solution can be characterized by a sequence of priority rules and, in the second case, the tasks are randomly selected according to the value of fitness provided by a rule.

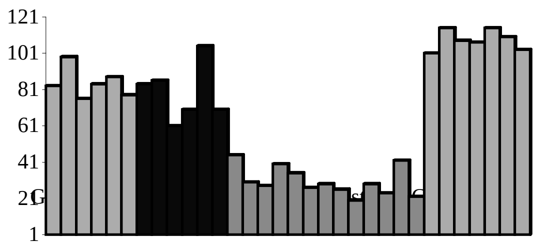

Figure 2. Number of times that the heuristics achieved the best solution in the 160 instances.

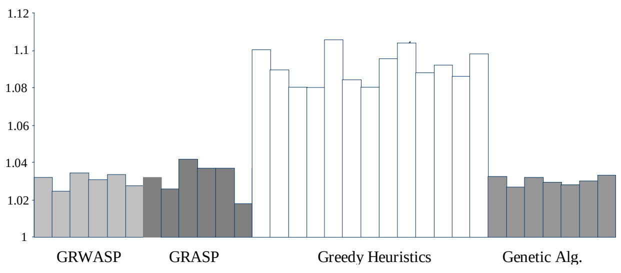

Figure 3. Results obtained for the 160 instances with the 32 mentioned procedures; the vertical axis indicates the ratio between the average of the values obtained by each procedure and the best obtained value.

Appendix A: List of Rules

Nomenclature:

i, j	indices for the tasks
N	number of tasks
C	cycle time
t_i	duration of task i
IS_i	set of tasks immediately after task i
S_i	set of tasks after task i
TP_i	set of precedent tasks of i
Li	level of task i in the precedence graph

Schedule the task z^* : $v(z^*)=\max_{i\in Z}[v(i)]$

Name	Rule
1.*Longest Processing Time*	$v(i) = t_i$
2.*Greatest Number of Immediate Successors*	$v(i) = \mid IS_i \mid$
3.*Greatest Number of Successors*	$v(i) = \mid \mathrm{S}_i \mid$
4.*Greatest Ranked Positional Weight*	$v(i) = t_i + \sum t_j \ (j\in S_i)$
5.*Greatest Average Ranked Positional Weight*	$v(i) = (t_i + \sum t_j \ (j\in \mathrm{S}_i)) / (\mid S_i \mid +1)$
6.*Smallest Upper Bound*	$v(i) = -UB_i = -N -1 + [(t_i + \sum t_j \ (j\in S_i))/C]^+$
7.*Smallest Upper Bound / Number of Successors*	$v(i) = -UB_i / (\mid S_i \mid + 1)$
8.*Greatest Processing Time / Upper Bound*	$v(i) = t_i / UB_i$
9.*Smallest Lower Bound*	$v(i) = -LB_i = - [(t_i + \sum t_j \ (j\in TP_i)) / C]^+$
10. *Minimum Slack*	$v(i) = -(UB_i - LB_i)$
11.*Maximum Number Successors / Slack*	$v(i) = \mid S_i \mid / (UB_i - LB_i)$
12.*Bhattcharjee & Sahu*	$v(i) = t_i + \mid S_i \mid$
13. *Kilbridge & Wester labels*	$v(i) = -L_i$

References

[1] Baybars, I. (1986): “A survey of exact algorithms for the simple assembly line balancing problem”. *Management Science*, Vol. 32, No. 8, pp. 909-932.

[2] Hoffmann, T.R. (1992): “Eureka. A hybrid system for assembly line balancing”. *Management Science*, Vol. 38, No. 1, pp. 39-47.

[3] Klein, R.; Scholl, A. (1996): “Maximizing the production rate in simple assembly line balancing – a branch and bound procedure”. *European Journal of Operational Research* Vol. 91, No. 2 Jun 7, pp. 367-385.

[4] Erel, Erdal; Sarin, Subhash C. (1998): “Survey of the assembly line balancing procedures”. *Production Planning and Control*, Vol. 9, No. 5, Jul-Aug 1998, pp. 414-434.

[5] Boctor, F.F. (1995): “A Multiple-rule Heuristic for Assembly Line Balancing”. Journal of the Operational Research Society, 46, pp. 62-69.

[6] Ugurdag, H.F.; Rachamadugu, R.; Papachristou, C.A. (1997): “Designing paced assembly lines with fixed number of stations”. *European Journal of Operational Research*, Vol. 102, No. 3, pp. 488-501.

[7] Rekiek, Brahim; Falkenauer, Emanuel; Delchambre, Alain (1997): “Multi-product resource planning”, Proceedings of the IEEE International Symposium on Assembly and Task Planning 1997. Marina del Ray, CA, USA, pp. 115-121.

[8] Díaz, A.; Glover, F.; Ghaziri, H; González, J.M.; Laguna, M.; Moscato, P.; Tseng, F. (1996): *Optimización heurística y redes neuronales.* Paraninfo.

[9] Storer, R.H.; Wu, S.D.; Vaccari, R. (1992): “New search spaces for sequencing problems with application to Job Shop Scheduling”. *Management Science.* Vol. 38, No. 10, pp.1495-1509.