THE ASSEMBLY LINE BALANCING PROBLEM
SOLVED BY HYBRID HEURISTIC PROCEDURES AND
DRIVEN EXPLORATION *

Joaquín Bautista, Manuel Mateo, Raquel Ferrer, Jordi Pereira, Ramón Companys
Polytechnic University of Catalonia (UPC)
ETSEIB. Av.Diagonal 647, Planta 7; 08028 Barcelona (Spain)
Telf.: 34 934011703, 34 934011702. Fax: 34 934016605
e-mail: bautista@oe.upc.es

ABSTRACT

This paper deals with the assembly line balancing problem with incompatibilities between tasks with two hierarchical objectives: minimising the number of stations, and once these are fixed, minimising the cycle time. It is solved through CPLEX optimiser software, in which we have introduced a bounding procedure and an exploration strategy based on greedy heuristic priority rules.

Keywords: Assembly lines, metaheuristics, driven exploration.

INTRODUCTION

Problems of assembly lines, known as ALBP (Assembly Line Balancing Problem), can be classified in two kinds according to Baybards (1986): SALBP (Simple Assembly Line Balancing Problem) and GALBP (General Assembly Line Balancing Problem).

The first group consists in determining a number of workstations (each one assigned to a worker or robot), with identical cycle time or production rate. Given a group of elementary tasks with pre-established processing times. Nevertheless, precedence relationships must be taken into account and tasks can only be assigned to one station. There are two possible objective functions:

(1) Minimising the number of stations with a fixed production rate; known as SALBP-1.
(2) Minimising the cycle time with a fixed number of stations: known as SALBP-2.

The second group, GALBP, includes the rest of problems. This category includes parallel workstations, tasks grouping, tasks incompatibilities, other objective functions, etc.

In this work we focus around an extension of the SALBP-1 problem, in which the treatment of incompatibilities between tasks (e.g. clean with dirty hands, left with right side of the line tasks, etc.) has been added. So two incompatible tasks cannot be assigned to the same workstation.

There are two hierarchical objectives: the first is minimising the number of workstations and, subordinated to this, maximising the production rate of the line, once the number of stations is fixed as a result of the first objective.
An exact procedure to solve SALBP has been treated by several authors. In this approach there are procedures based on branch and bound (B&B), as Hoffmann (1992), but it is, generally, only viable on small sized problems, due to the NP-hard nature of the problem. In order to solve real dimension problems, like those in car industry, a more suitable approach should be based on heuristic methods, as in Ugurdag et al. (1997), for SALBP-2, as in Boctor (1995) for SALBP-1.

For this last problem, it is important to remark the greedy heuristics based on the use of priority rules, which are focussed to the assignment of the tasks to the workstations. The rules generally refer or combine some aspects, the task processing time, the number of immediate successors, bounds on the number of stations up to finish the assignment, etc. At each iteration, those rules allow us to establish an order of the tasks. This order is used to select the most appropriate task from a set of (those tasks compatible with the partial solution already built). A task becomes eligible, when its predecessors have been assigned and for the current station the cycle time would not be surpassed. Usually, each heuristic has only an assigned rule, and this rule determines (unless using a random decision) the next task assigned. The application of this type of algorithms provides acceptable, as better as more aspects the rule combines. But beforehand it is impossible to determine which is the best rule for each problem, and unless random is added to the procedure, those procedures will always give the same solution.

A second class of heuristics, the GRASP (Greedy Randomised Adaptive Search Procedure) algorithms, can generate different solutions due to the random process implied. At each iteration, a selection of tasks is established, based on a priority rule, from the group of candidates. Those selected tasks are drawn for assignation, which can or cannot be modified by the priority value of the rule. This procedure has been applied successfully to several procedures for Industrial Engineering applications, as those described by González in Díaz et al. (1996); and specifically to this problem satisfactory results are described in Bautista et al. (2000).

Finally, a third class of heuristics are local search methods or neighbourhood exploration, as Diaz et al. (1996), cited: Hill Climbing (HC), Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithms (GA). Those heuristics give us alternative ways of searching solutions in a definite space, called as neighbourhood. Even though, if this definition of neighbourhood is general and valid for any combinatorial problem, the knowledge of this specific problem, which is used by the greedy heuristics, is left aside.

Bautista et al. (2000) proposed two different procedures, which simultaneously use the main aspect of the first class of heuristics and the main aspect of the other two: 1) the knowledge about SALBP given by the specific priority rules, 2) the generation of a large amount of solutions in the search space, desirable in any combinatorial problem. In this work, the specific applications are two procedures: first, a GRWASP algorithm (Greedy Randomised Weighted Adaptive Search Procedure), and second, a Genetic Algorithm which generates sets of heuristic rules to explore the heuristic space.

In this study a SALBP with incompatibilities between tasks is proposed, starting from the LMP model and adding some improvements to limit the group of feasible solutions: adjustment of the cycle time, determination of lonely tasks, determination of the range of workstations for each task to be assigned, determination of lower and upper bounds to the number of workstations, etc. The proposed procedure searches for in the heuristic space (composed of 13 basic priority rules, with the possible addition of pure random assignment). In this way, the heuristic $h$ in this space is defined as a vector of rules $r$ and a
task assignment algorithm $A$. Each solution obtained in the search is submitted to an optimality test through the CPLEX solver, with priority imposed rules in order to fix the value of some binary variables (assignment of a task to a workstation), in the branching procedure. Such priorities are related to the vector of rules used in the generation of each solution.

The scheme of paper is as follows. The metaheuristics used to find an initial solution of each problem are described in Section 2. The hybrid procedure of resolution with the optimality test is developed in Section 3, which is complemented by a computational experience in Section 4. Finally, Section 5 shows the conclusions of this work.

2. METAHEURISTICS

Basic Greedy and GRASP heuristics

Figure 1 shows a framework of a heuristic procedure to generate a solution for the SALBP problem.

This framework is valid for the greedy deterministic heuristics, as well as for the GRASP.

The former procedures assign the tasks, only one at a time, which is the one with the best value for the priority rule (or rules) applied in the algorithm. The selection is made between all the compatible tasks (depending on the already built partial solution), which satisfy precedence relation, incompatibilities and remaining time for the workstation. A local optimisation procedure may be (or not) applied to this solution.

In the later algorithms, at any iteration the number of candidates can be limited. During the assignment process a task is selected by drawing lots from a list of candidate tasks. The selection probabilities can depend on the value of the index (GRWASP: Greedy Randomised Weighted Adaptative Search Procedure). The found solution is then improved by a local optimisation procedure.

In order to define a heuristic, there must be logically at least a priority rule for the selection procedure; In Appendix I, a list of 13 basic rules are shown, to which the random selection may be added.

Local Search heuristics

The local search methods are used to explore neighbourhood solutions. To define a neighbourhood (group of solutions directly related to a given solution) it is necessary to have a solution characterised through a sequence of elements (e.g. a sequence of the assembly tasks, that implies the order in which they are assigned) and interchanging some element positions following some rules. This definition of a neighbourhood is general and does not take advantage of some specific and relevant information of the particular problem, even though it gives the advantage of universality and applicability to any sequencing problem.

Storer (1992) presented other alternative methods to define a neighbourhood, based on the relationship between a heuristic $h$ and a solution $s$, obtained of its application from a particular problem $p$: $h(p) = s$. This relationship allow us to establish a neighbourhood among the problem space and the heuristic space.
The definition of neighbourhood in the problem space is based on the idea of random perturbations on the problem data, within reasonable levels, and afterwards the application of a heuristic to obtain a solution. The evaluation of the solution is, of course, done with the original data.

On the other hand, the key aspect to define neighbourhoods in the heuristic space is based on the possibility to develop new versions from the set of special available heuristics for the problem. This last option can be developed in two ways for the SALB problems:

1. Defining a new hybrid rule \( r \) through a linear weighted combination of the set of originally available rules for assignation \( \rho \): \( \rho = \sum_{i} \pi_{i} \rho_{i} \)

2. Dividing the assignation decisions of tasks to the open stations in sets or windows, and associate to each window a rule. This way offers the possibility to characterise a solution through a vector of rules as shown: \( r = (\rho_{1}, \rho_{2}, ..., \rho_{N}) \); where \( N \) is the number of tasks and \( \rho_{k} \) is the rule to apply in \( k \)-th assignation decision.

Focussing on this second way to search through the heuristic space, we can conclude that given a vector of rules \( r \) and a procedure \( A \), a heuristic \( h \) may be defined from a pair \((r, A)\): \( h = h(r, A) \). All resultant heuristics are OK vectors of rules, so the procedure shown in Figure 1 may be used. Below, two methods are applied to generate solutions in the heuristic space, a Hill Climbing procedure and a Genetic Algorithm.
Hill Climbing

0. Initialisation: Create a heuristic by drawing lots for each position of the vector of rules from a group of heuristics.

Iterate $L$ times through the next steps:

1. Determine at random a position of the vector of rules.
2. Assign a rule at random, to the given position (a different rule to the actual one).
3. Apply the new heuristic to obtain a solution and calculate the objective function value.
4. If the value of the objective functions is better or equal to the initial one, change the vector of rules to the new one. Go to 1.

Genetic Algorithm. Nomenclature

$I$ number of individuals (vectors of rules) in the population.
$L$ number of iterations (generations).
$p$ instance of the problem to be solved.
$\Pi_r$ population of ancestors of the sequences of rules.
$\Pi_h$ population of ancestors of the heuristics.
$\Pi_s$ population of ancestors of the solutions.
$\Delta_r$ population of descendants of the sequences of rules.
$\Delta_h$ population of descendants of the heuristics.
$\Delta_s$ population of descendants of the solutions.
$\Lambda_r$ population of muted descendants of the rules sequences
$\Lambda_h$ population of muted descendants of the heuristics
$\Lambda_s$ population of muted descendants of the solutions
$\Omega_r$ population of eligible sequences of rules for the next iteration (generation).
$r_i$ element $i$ of the sets $\Pi_r$, $\Delta_r$, $\Lambda_r$ and $\Omega_r$.
$h_i$ element $i$ of the sets $\Pi_h$, $\Delta_h$, $\Lambda_h$ and $\Omega_h$.
$s_i$ element $i$ of the sets $\Pi_s$, $\Delta_s$, $\Lambda_s$ and $\Omega_s$.

Genetic Algorithm. Description

0 Data initialisation

0.1 Generate the initial populations $\Pi_r$ with homogeneous and different sequences of rules: $\Pi_r = \{ r_i = (\rho_1, ..., \rho_N) : \rho_1 = .. = \rho_N \}$

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

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

0.4 Determine the fitness $f_j$ of the elements of $\Pi_s$ as:

$$f_j = -NE_j + \left[ \sum_{i=1}^{\text{NE}_j} t_i \right] + \frac{\sqrt{\sum_{k=1}^{\text{NE}_j} (c_k - C)^2}}{C \sqrt{\text{NE}_j}}$$

where:

$t_i$ duration for task $i$

$C$ cycle time

$NE_j$ number of stations in the $j$-th solution

$c_k$ occupied time in the $k$-th station ($1 \leq k \leq \text{NE}_j$)
0.5 Save as incumbent the pair \((h^\ast, s^\ast)\) with greater fitness

Phase B: Iterate through the following steps \(L\) times:

1. Selection of ancestors:
   Build \(I/2\) pairs of elements of \(\Pi_r\) according to the fitness of the elements of \(\Pi_r\).

2. Choice of the pairs for the crossover:
   2.1 Determine the probability of the current crossover: \(P_c = P_c(\alpha)\) with
   \[
   \alpha = \frac{1}{I(I-1)} \sum_{i=1}^{I} \frac{1}{N} \sum_{r \neq r'} h_{i,r} \cdot h_{i',r'}
   \]
   where \(h_{i,j} \in \{0,1\} \wedge [ h_{i,j} = 1 \Leftrightarrow \rho_{[i]}(\in r_j) = \rho_{[i]}(\in r_j) \forall k, 1 \leq k \leq N ]\)
   2.2 Assign a random number to each pair of sequences of rules.
   2.3 Decide, for each pair of sequences of rules, if a crossover should be done according to the relation between the random number and the probability depending on the homogeneity of population \(P_c\)

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 it was done in 0.2 and 0.3 respectively.
   3.3 Determine the objective function value of the elements of \(\Delta_s\) If any element of \(\Delta_s\) has a better objective function value than the incumbent solution, then save as heuristic and incumbent solution the pair \((h^\ast, s^\ast)\) associated to 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 \(\Delta_h\) and \(\Delta_s\) from \(\Lambda_r\) as it was done in 0.2 and 0.3 respectively.
   4.6 Determine the objective function value of the elements of \(\Delta_s\) If any element of \(\Lambda_s\) has a better objective function value than the incumbent solution, then save as heuristic and incumbent solution the pair \((h^\ast, s^\ast)\) associated to 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 \(\Delta_r\) and \(\Lambda_r\) as it was indicated in 0.5.
   5.3 Choose \(I\) elements from \(\Omega_r\) according to the fitness of the elements of \(\Pi_r\), \(\Delta_r\) and \(\Lambda_r\).

3. HYBRID PROCEDURE

Figure 2 shows a framework related to the general procedure proposed in this study.
Figure 2: Framework of the hybrid procedure to solve the SALB problem.

Once the basic model is generated, a pre-processor is applied in order to: (1) reduce the number of variables and constraints; (2) adjust the minimum number of stations solving a knapsack problem; (3) determine the maximum number of tasks for each station; and (4) determine the adjusted times for each task. Afterwards, a heuristic procedure (any of the above described) establishes an order for the task assignation to a workstation. The order is used to give priorities to the binary variables (which are used to assign a task to a workstation), and so fix the values to be taken during the branch and bound procedure used by the CPLEX solver. SALBP-1 is solved, and then the SALBP-2, for a fixed number of stations found, in a similar way.

4. COMPUTATIONAL EXPERIENCE

The collection of problems used for the computational experience is formed by 120 instances: 12 groups (which combine the number of tasks and the cycle time) with 10 instances each one. The analysed cycle times are 12 s., 15 s. and 25 s.; while the number of tasks are 20, 40, 60 and 80 tasks, with processing times lower than 12 s.

All these problems were solved using the pure greedy heuristics associated to the 13 priority rules (Appendix I). This implies 1560 executions has been done with the hybrid procedure,
which requires of the heuristics and the CPLEX solver. Figure 3 shows the distribution of
the number of variables associated to the LMP models, in the commented blocks of 10
instances.

![Figure 3: Number of variables associated to the LMPs](image)

The execution time has been limited to 800 s. on any SALBP-1 instance and another 800 s.
to the SALBP-2, to every element. The computer used works at 250 MHz. For those
conditions, the most relevant results can be summarised as follows:

- Heuristics associated to the 13 priority rules reach the optimum in 36.6% of the
  problems.
- Hybrid B&B obtained a solution for all the instances.
- Hybrid B&B assured the optimum for the concatenated sequences of problems
  SALB1/SALB2 in more than 70% instances.
- Hybrid B&B improved the solution given by the heuristics in the rest of instances.
- Without taking care the 80 tasks instances, only solved through Hybrid B&B, the mean
  computing time is 27.53 min. for the B&B while the hybrid B&B mean time is 3.72
  min.
- The difference between the number of optimum solutions found using the hybrid
  procedure versus the traditional one grows dramatically as the problem the dimension
  of an instance (see Figure 4).

![Figure 4: Percentage of optimum solutions found by both procedures](image)
5. CONCLUSIONS

This work proposes a way to explore and obtain solutions to the SALBP problem with incompatibilities between tasks. The main idea consists in adding to graph exploration procedures the knowledge given by deterministic greedy heuristics or other procedures as the neighbourhood exploration. Once a solution is obtained by one of those procedures, represented by a sequence of tasks for workstation assignments, a priority to fix the variables is given for a driven B&B exploration. The exploitation has offered satisfactory results, which demonstrate the differences between a driven and a blind search.

6. REFERENCES


APPENDIX I

Nomenclature:

- \( i, j \) indexes for the tasks
- \( N \) number of tasks
- \( C \) cycle time
- \( t_i \) processing time of task \( i \)
- \( IS_i \) set of immediate tasks after task \( i \)
- \( S_i \) set of tasks after task \( i \)
- \( TP_i \) set of precedent tasks of task \( i \)
- \( Li \) level of task \( i \) in the precedence graph

Schedule the task \( z^* : \nu(z^*) = \max_{i \in A} \{ \nu(i) \} \)

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Metric</td>
<td>Formula</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>1. Longest Processing Time</td>
<td>( v(i) = t_i )</td>
</tr>
<tr>
<td>2. Greatest Number of Immediate Successors</td>
<td>( v(i) =</td>
</tr>
<tr>
<td>3. Greatest Number of Successors</td>
<td>( v(i) =</td>
</tr>
<tr>
<td>4. Greatest Ranked Positional Weight</td>
<td>( v(i) = t_i + \sum_{j \in S_i} t_j )</td>
</tr>
<tr>
<td>5. Greatest Average Ranked Positional Weight</td>
<td>( v(i) = (t_i + \sum_{j \in S_i} t_j) / (</td>
</tr>
<tr>
<td>6. Smallest Upper Bound</td>
<td>( v(i) = -UB_i = -N - 1 + (t_i + \sum_{j \in S_i} t_j) / C )</td>
</tr>
<tr>
<td>7. Smallest Upper Bound / Number of Successors</td>
<td>( v(i) = -UB_i / (</td>
</tr>
<tr>
<td>8. Greatest Processing Time / Upper Bound</td>
<td>( v(i) = t_i / UB_i )</td>
</tr>
<tr>
<td>9. Smallest Lower Bound</td>
<td>( v(i) = -LB_i = -[t_i + \sum_{j \in TP_i} t_j] / C )</td>
</tr>
<tr>
<td>10. Minimum Slack</td>
<td>( v(i) = - (UB_i - LB_i) )</td>
</tr>
<tr>
<td>11. Maximum Number Successors / Slack</td>
<td>( v(i) =</td>
</tr>
<tr>
<td>12. Bhattcharjee &amp; Sahu</td>
<td>( v(i) = t_i +</td>
</tr>
<tr>
<td>13. Labels Kilbridge &amp; Wester</td>
<td>( v(i) = -L_i )</td>
</tr>
</tbody>
</table>

* This work has been partially funded by the project TAP98-0494.