Metaheuristic procedures for the Lexicographic Bottleneck Assembly Line Balancing Problem

RAFAEL PASTOR
Institute of Industrial and Control Engineering and Department of Management
Universitat Politècnica de Catalunya, Spain
rafael.pastor@upc.edu

ALBERTO GARCÍA-VILLORIA
Institute of Industrial and Control Engineering and Department of Management
Universitat Politècnica de Catalunya, Spain
alberto.garcia-villoria@upc.edu

MANUEL LAGUNA
Leeds School of Business, University of Colorado at Boulder, USA
laguna@colorado.edu

RAFAEL MARTÍ
Department of Statistics and Operations Research
Universitat de València, Spain
rafael.marti@uv.es

ABSTRACT

The goal of this work is to develop an improved procedure for the solution of the lexicographic bottleneck variant of the assembly line balancing problem (LB-ALBP). The objective of the LB-ALBP is to minimize the workload of the most heavily loaded workstation, followed by the workload of the second most heavily loaded workstation and so on. This problem —recently introduced to the literature (Pastor 2011)— has practical relevance to manufacturing facilities. We design, implement and fine-tune GRASP, tabu search and scatter search heuristics for the LB-ALBP and show that our procedures are able to obtain solutions of a quality that outperforms previous approaches. We rely on both semi-greedy and memory-based designs that our experiments show to be effective. Experimental results verify the advantages of embedding such designs to improve the solution existing in the literature of this complex problem. Additionally, the extensive experimentation with 48 variants of GRASP, 12 of tabu search and 1 of scatter search establishes the benefits of adding enhanced search strategies to basic procedures.

Keywords: Assembly line balancing, lexicographic bottleneck, metaheuristics.

November 26, 2014

1. Introduction

Assembly lines are at the core of mass production systems, such as those in the automotive industry. An assembly line consists of a series of workstations in which the product flows. In each workstation, several tasks (assembly operations) are performed on the products determining the total duration or cycle time. The class of assembly line balancing problems (ALBP) consists of assigning the tasks to workstations to optimize one or multiple objectives while satisfying some specific conditions. The practical importance of this family of NP-hard problems (Wee and Magazine, 1982) has resulted in a vast literature. We refer the reader to Becker and Scholl (2006), Scholl and Becker (2006), Boysen *et al*. (2007, 2008), and Battaïa and Dolgui (2013) for some of the most recent surveys.

The simplest case of ALBP —referred to as simple assembly line balancing problem (SALBP)— consists of a serial line that processes a single product. In the SALBP, task times are assumed to be deterministic and precedence constraints are the only design restrictions (Scholl and Becker, 2006). This problem has been extensively studied in the literature and has been approached with both heuristic and metaheuristic procedures (e.g., Talbot *et al*., 1986; Ponnambalam *et al*., 1999; Corominas and Pastor, 2009), exact procedures based on binary or integer linear programming (e.g., White, 1961; Talbot and Patterson, 1984; Pastor and Ferrer, 2009), dynamic programming (e.g., Kao and Queyranne, 1982), and branch and bound (e.g., FABLE by Johnson, 1988; EUREKA by Hoffman, 1992; SALOME by Scholl and Klein, 1997).

To address more realistic versions of the ALBP, researchers have recently intensified their efforts by examining further aspects of actual manufacturing systems (Becker and Scholl, 2006). The following are some of the most prominent examples of ALBP features addressed in the literature: parallel tasks (e.g., Inman and León, 1994), incompatibility between tasks (e.g., Park *et al*., 1997), U-shaped lines (e.g., Miltenburg, 2002), mixed-models (e.g., Ding *et al*., 2006), stochastic task times (e.g., Gamberini *et al*., 2006), parallel workstations (e.g., Lusa, 2008), setup times between tasks (e.g., Martino and Pastor, 2010), alternative assembly subgraphs (e.g., Capacho et al., 2009), constrained resources (e.g., Corominas *et al*., 2011) and ergonomics considerations (e.g., Cheshmehgaz *et al*., 2012).

With respect to the objective function, we can distinguish two main groups. When the objective is to minimize the number of workstations for a given upper bound on the cycle time, the problem is known as ALBP-1. When the problem consists of minimizing the cycle time given a number of workstations, the problem is known as ALBP-2 (Baybars, 1986). Note that ALBP-2 is a problem with a *minimax* objective, since it attempts to minimize the workload of the workstation with the maximum load (i.e., the bottleneck). ALBP-2 ignores the workloads of those workstations that are not the bottleneck. However, as pointed out in previous works (Boysen *et al*. 2006), uniformly distributed workloads among workstations help to improve the reliability of the line and, as it is discussed in Boysen *et al*. (2006), the quality defects caused by workstations with disproportionately large processing times are avoided.

[Insert Figure 1 here]

Figure 1 shows the precedence graph of a process with 9 tasks, numbered from 1 to 9. Each node in the graph represents a task, with its associated processing time in parentheses, and precedence constraints are represented by arcs between nodes. Table 1 shows two designs that assign these 9 tasks to 5 workstations (W1, W2, …, W5). In both solutions, W1 has the maximum workload of 10, making this workstation the bottleneck that determines the cycle time for the design. According to ALBP-2, both of these assignments have the same objective function value.

[Insert Table 1 here]

Figure 2 is a graphical representation of the workloads corresponding to each design in Table 1.

[Insert Figure 2 here]

Although both designs have the same bottleneck (i.e., W1) and a cycle time of 10, it is clear that they are quite different in terms of their workload distribution. Discriminating between both solutions and, in general, among all the solutions with the same maximum workload, was the motivation of Pastor (2011) for proposing a new variant of the ALB problem: the lexicographic bottleneck ALBP (referred to as LB-ALBP), which takes into account the load of all workstations.

The LB-ALBP employs a hierarchical approach to first minimize the workload of the most heavily loaded workstation, followed by the workload of the second most heavily loaded workstation, followed by the third, and so on. It is therefore a multi-objective optimization problem with a hierarchical structure (Cortés *et al.*, 2006) in which the quality of the solution is determined by the order of the objectives instead of a trade-off among objectives. We cannot know a priori whether the workload of all workstations are important in practice with respect to the reliability of the line. For instance, if the second most workload is also equal (or close) to the cycle time, obviously the third most workload is important. And if the third most workload is also equal to or close to the cycle time, then the fourth most workload is important, and so on. Therefore, since a priori we do not know which are the optimal workloads, the LB-ALBP takes into account hierarchically all of them.

In mathematical terms, we consider m ordered workstations and n tasks, where each task i $(i = 1, \dots, n)$ is defined by its processing time t_i and a set P_i of its precedence tasks (i.e., those that must be processed before task i). A feasible solution is an assignment of tasks to workstations verifying their precedence relationships. Given a feasible solution s, let $w(s, j)$ be the workload of workstation j in solution s. As mentioned above, while the classical ALBP-2 seeks to minimize the maximum workload; i.e., $\max_{j=1,\dots,m} w(s,j)$, the LB-ALBP minimizes the *lexicographical bottleneck* objective.

Pastor (2011) proposed a function $\boldsymbol{\delta(s)}$ to evaluate the merit of a solution $\boldsymbol{s}$ of an LP-ALBP instance. $\boldsymbol{\delta}$ is based on computing differences between the workloads of *s* and an "ideal" distribution of workloads. The idea distribution may be interpreted as a lower bound given that in general this distribution does not correspond to a feasible solution. The ideal $\boldsymbol{j^{th}}$ workload is calculated as follows:

$$ideal(j) = max\left(\left\lceil\frac{T-\sum_{k=1}^{j-1} ideal(k)}{m-j+1}\right\rceil, t_{\sigma(j)}\right)$$

where $T = \sum_{i=1}^{n} t_i$, $\sigma(j)$ is the task with the j^{th} largest processing time and $\lceil x \rceil$ is the smallest integer that is equal to or greater than x. The equation produces ideal workloads in decreasing order. That is, $ideal(j) \geq ideal(j')$ if $j < j'$. The value of δ is then computed as follows:

$$\delta(s) = \frac{\sum_{j=1}^{m}\big((w\big(s, \pi(j)\big) - ideal(j)) \cdot \beta^{m-j+1}\big)}{ideal(1) \cdot \beta^{m-1}}$$

where β is a large value that reflect the hierarchy among workstations and $\pi(j)$ is the workstation with the j^{th} largest workload and therefore, for any solution s, $w\big(s, \pi(j)\big) \geq w\big(s, \pi(j')\big)$ if $j < j'$.

Note that $ideal(1) = max\left(t_{\sigma(1)}, \left\lceil\frac{T}{m}\right\rceil\right)$, the lower bound on the cycle time. In the example above, with $\beta = 100$ (the typical value employed in the literature, see Pastor (2011) and Pastor *et al*. (2012)), the ideal distribution of workloads is $ideal = (10,8,8,7,7)$, and the ordered workloads for the first design are 10, 10, 9, 6, and 5. Therefore, δ value for the first design is:

$$\frac{(10-10)\cdot 100^5 + (10-8)\cdot 100^4 + (9-8)\cdot 100^3 + (6-7)\cdot 100^2 + (5-7)\cdot 100^1}{10\cdot 100^4} = 0.201$$

The main objective of this work is to improve upon the current competency to find high quality solutions to the LB-ALBP. With this in mind, we explore the design, implementation and fine-tuning of greedy randomized adaptive search procedure (GRASP), tabu search (TS) and scatter search (SS) procedures. In metaheuristic search, the ability to find high quality solutions depends on an effective interplay between search intensification and diversification. Our experiments are designed to evaluate the role that special memory structures play in inducing search exploration and exploitation to achieve high quality outcomes. Comparisons against existing methods reveal that we have been able to establish a new state of the art for solving LB-ALBP instances.

2. Relevant Previous Work

We review a few aspects of the literature that are directly related to our work.

2.1. Workload Smoothing

The homogeneous distribution of the workload is a goal that appears in the ALBP literature. It has been usually achieved by minimizing the sum, for all workstations, of the differences between their workloads and the cycle time (e.g., Moodie and Young, 1965; Rekiek *et al*., 2002), or the average workload (e.g., Rachamadugu and Talbot, 1991; Merengo *et al*., 1999). The most common objective for smoothing the workload is to minimize the so-called "smoothness index" (SI) by Moodie and Young (1965), which using our notation, is calculated as follows for a solution s:

$$SI(s) = \sqrt{\sum_{j=1}^{m}(CT(s) - w(s,j))^2}$$

Although the workloads of optimal LB-ALBP solutions tend to be smooth, as explained in Pastor (2011), the LB-ALBP objective is different from the SI objective. We illustrate this as follows. Let the workloads for two solutions, $s1$ and $s2$, be $(34,33,33,29,19)$ and $(35,30,29,29,25)$, respectively. Their corresponding SI values are 15.87 and 14.04, making $s2$ better than $s1$ according to this smoothing objective. However, when considering the lexicographical objective of LB-ALBP, $s1$ is actually better solution than $s2$.

2.2. SALBP-2

The single objective in SALBP-2 is the minimization of the cycle time, which is the first objective of the LB-ALBP. The SALBP-2 has been solved by exact and heuristic methods (e.g., Scholl and Becker, 2006). Most heuristic solutions to the SALBP-2 are based on solving iteratively the simple ALBP-1 (SALBP-1) by applying the procedure in Figure 3, in which processing times of the tasks are assumed, without loss of generality to be integers, restricting CT to integer values.

[Insert Figure 3 here]

The greedy heuristic in Figure 3 is iterative and workstation-oriented; i.e., at each step the best candidate task (according to the chosen priority rule) is assigned to the workstation j under consideration. A task i is a candidate to be assigned to workstation j if all its precedent tasks have already been assigned and the sum of the processing time of the task and the current workload of the workstation does not exceed the desired cycle time. If there are no available candidate tasks (but there are still tasks to be assigned) then workstation j is closed and the next workstation $j+1$ is opened. The procedure ends when all tasks have been assigned. Most computational experiments reported in the literature indicate that workstation-oriented procedures provide better results than task-oriented ones, although they are not theoretically dominant (Scholl and Voß, 1996).

2.3. LB-ALBP

Two mixed integer linear programming (MILP) approaches, referred to as *GHM* and *SHM*, were proposed in (Pastor 2011) for the LB-ALBP. *GHM* directly minimizes $\sum_{j=1}^{m} \beta^{m-j+1}\, w(s, \pi(j))$, which is a weighted sum of functions; while *SHM* sequentially solves $m-1$ MILP submodels by not allowing to deteriorate the optimal values of the higher-priority objectives that have been obtained. Experiments clearly show that *SHM* performs better; although, only smallest instances were solved optimally within 5 hours of computational time per instance. For larger instances the author proposed three straightforward heuristics based on the previous MILP procedures. The heuristics consisted of several strategies to use a limited computational budget.

Pastor *et al*. (2012) proposed *V-LSPa,* a deterministic heuristic based on iteratively solving SALBP-2. *V-LSPa* sequentially solves and divides the problem into two smaller subproblems (with lower number of workstations and tasks to assign) using the most heavily loaded workstation in each subproblem as the split point. Then, each subproblem is solved as a SALBP-2 by means of the heuristic scheme shown in Figure 3. Twelve solutions are obtained by applying the SALBP-1 greedy heuristic step with the following twelve priority rules:

1. Maximum ranked positional weight
2. Maximum task time
3. Maximum total number of follower tasks
4. Maximum number of immediate follower tasks
5. Maximum average ranked positional weight
6. Maximum task time divided by upper bound
7. Maximum total task followers divided by task slack
8. Minimum lower bound
9. Minimum upper bound
10. Minimum slack
11. Minimum task number
12. Minimum upper bound divided by followers

The procedure selects the best of the twelve solutions according to the LB-ALBP objective. The solution to each subproblem is used to update the global solution of the original problem. Each time that the global solution is updated, a local search is applied. When a local optimum improves upon the incumbent solution, the process is reset starting from the new local optimum. The heuristic ends when no subproblems remain to be solved. The authors compared their best heuristic with the best in Pastor (2011), and *V-LSPa* obtains on average the best results.

3. GRASP

The greedy randomized adaptive search procedure (GRASP) metaheuristic was designed by Feo and Resende (1989) and belongs to the family of multi-start approaches. Each GRASP iteration consists of constructing and improving solutions. The constructions are semi-greedy, meaning that they involve the probabilistic selection of a reduced set of top choices. The improvement typically consists of a local search. Unlike tabu search, GRASP is a memoryless metaheuristic since no information is transferred from one iteration to the next.

Our construction procedures for the LB-ALBP are based on the scheme in Figure 3, where the heuristic for the SALBP-1 is replaced with the procedure in Figure 4. Therefore, instead of choosing the top task in the candidate list of tasks (CL) according to the priority rule, a so-called *restricted candidate list* (RCL) is created and the next task is randomly selected from this list. The procedure is outlined in Figure 4.

The construction procedure of Figure 4 starts with the first workstation and a new workstation is opened only if the current workstation does not have the capacity to perform an additional task and the set of unassigned tasks is not empty. We use the notation $s(j)$ to represent the set of tasks assigned to workstation j in solution s. We consider two strategies to generate RCL and two strategies to choose a task from the RCL. The strategies to build RCL are:

- *Cardinality-based* — Let b be a parameter controlling the size of the RCL. Then the RCL consists of the b tasks with the highest priority, given by the rule being applied. If $b > |CL|$, then $RCL = CL$.
- *Value-based* — Let α be a parameter controlling the merit (measured by the priority rule) of the tasks in the RCL. Then, the RCL consists of all tasks for which their priority is within α% of the task with the best priority.

The strategies to select a task from the RCL are:

- *Equal Probability* — All tasks in the RCL have equal probability to be selected. This is the classical selection strategy in the GRASP methodology.
- Priority-based Probability — The probability of selecting a given task is proportional (for rules 1 to 7) or inversely proportional (for rules 8 to 12) to its priority index value.

Combining the priority rules (twelve) and the strategies for building the RCL (two) and selecting a task from the RCL (two) results in a total of 48 variants of the procedure outlined in Figure 4. When a solution with m workstations is found, an attempt is made to improve upon the cycle time. This is done by reducing the current cycle time and calling the procedure in Figure 4 several times, up to an experimentally set limit of 10. Recall that heuristic in Figure 4 has stochastic elements and therefore a solution with m workstations and shorter cycle time may be found by multiple calls to the procedure.

[Insert Figure 4 here]

Our improvement method is the $LSPa$ local search in (Pastor *et al*. 2012), which is the best among those published in the literature for LB-ALBP. $LSPa$ is a hill climbing procedure that employs *trade and transfer* of tasks between pairs of workstations (Moodie and Young, 1965). In a solution s with workstations j and j' such that $w(s,j) > w(s,j')$, the neighborhood of s consists of all the solutions obtained by generating all feasible trades (i.e., those satisfying the precedence relations) between the tasks in workstation j and the tasks in workstation j', as well as all feasible transfers of the tasks in workstation j to workstation j'. $LSPa$ explores the neighborhood in the following order:

```
For $k = 1, \dots, m-1$ do
    For $l = m, \dots, k+1$ do
        $j = \pi(k)$
        $j' = \pi(l)$
    End for
End for
```

Recall that π represents the order according to decreasing workload values and therefore $\pi(1)$ is the workstation with the heaviest workload and $\pi(m)$ is the one with the lightest. These are the workstations that are paired first according to the procedure above. $LSPa$ uses a first-improving strategy, meaning that improving moves are immediately executed.

4. Tabu Search

Tabu search (TS) is a well-known metaheuristic originally proposed by Glover (1986) to emulate flexible and responsive memory of the form humans employ in solving challenging problems. Most of the tabu search implementations consist of straightforward short term memory structures, and are limited to a reduced subset of the elements that are part of the general methodology. For instance, constructive TS methods have been largely ignored in the literature, with the notable exception of (Duarte and Martí 2007). As reported in Glover and Laguna (1997), constructive TS procedures are based on memory structures used to favor or discourage the inclusion of an element in a solution during the constructive process. We couple a tabu search constructive procedure with a short term memory local search for this problem.

Workstation-oriented procedures for the ALBP assign a task i to a workstation j. We represent such assignment by the pair (i,j). Our tabu search implementation employs two memory structures: q (for quality) and f (for frequency). In $q(i,j)$, we record the average δ values of the solutions for which task i was assigned to workstation j and in $f(i,j)$ the number of times that the assignment (i,j) was made in all the solutions visited during the search. Since the workstations are ordered, their assigned number serves as an identifier. The q and f values are used to modify the attractiveness of an unassigned task during the solution construction process. In particular, we favor assignments with high quality and low frequency values, as indicated in the following greedy function:

$$g(i,j) = a(i) + \kappa \cdot r(U) \cdot \frac{q(i,j)}{q_{max}} - \gamma \cdot r(U) \cdot \frac{f(i,j)}{f_{max}}$$

Where

$a(i)$:	the priority index (attractiveness) of unassigned task i according to the rule being used
$r(U)$:	the range of $a(i)$ for all unassigned tasks ($r(U) = \max_{i \in U} a(i) - \min_{i \in U} a(i)$)
q_{max}:	the maximum value of $q(i,j)$
f_{max}:	the maximum value of $f(i,j)$
κ and γ:	parameters

The priority rules 1 to 7 indicate that the attractiveness of a task is given by maximizing the corresponding index, while for rules 8 to 12 the most attractive task is the one with the minimum index. We use the negative index value for rules 8 to 12 in order to always choose the assignment that maximizes the greedy function $g(i,j)$. Figure 5 shows the TS for the LB-ALBP.

[Insert Figure 5 here]

The construction procedure in Figure 5 is a modification of the one described in Section 2 above and that is based on the outline of Figure 3 and the twelve priority rules. The difference is that instead of using the unmodified index value, the constructions are done applying the greedy function $g(i,j)$ to choose the tasks and assign them to the workstations. Since in the first iteration the memory structures are empty, $g(i,j) = a(i)$ and the construction is identical to the original procedure suggested by Pastor *et al*. (2012).

The improvement method is a short-term memory tabu search based on the local search *LSPa*. At each iteration, the entire trade and transfer neighborhoods are explored and the best move is selected. If the best move leads the search to a solution that is better than the current best solution, then the move is made. Otherwise, the move to be made is the non-tabu trade or transfer with the best value even if it leads the search to a solution that is inferior to the current solution. After a move that trades task i in workstation j for task i' in workstation j', the tabu attributes to be stored in short-term memory are the pairs (i,j) and (i',j'). After a move that transfers task i in workstation j to workstation j', the tabu attribute to be stored in short-term memory is the pair (i,j). Attributes remain tabu-active for $TabuTenure$ iterations. A trade $(i \rightarrow j', i' \rightarrow j)$ is tabu if both pairs (i,j') and (i',j) are tabu-active. A transfer $(i \rightarrow j')$ is tabu if the pair (i,j') is tabu-active. The improvement procedure stops after $MaxIter$ consecutive iterations without improving the best solution found during the current application of the improvement method.

5. Scatter Search

The scatter search (SS) metaheuristic was first introduced by Glover (1977). As other evolutionary methods, SS operates at each iteration on a reference set of solutions rather than on a single solution at a time, as TS and GRASP do. SS combines the solutions of the reference set to create new ones in order to improve it iteratively. However, in contrast to other evolutionary methods, a *good* reference set of solutions not only implies having high quality solutions, but also diversity. For a detailed description of the SS methodology see Laguna and Martí (2003).

Our SS procedure is based on the following well-known five methods (Glover 1998) to implement it: a *diversification generation method* to generate a pool of diverse trial solutions, an *improvement method* to transform a trial solution into an enhanced trial solution, a *reference set update method* to build and maintain a reference set consisting in $bs/2$ high quality solutions and $bs/2$ diverse solutions (where bs is a parameter), a *subset generation method* to produce subsets (usually pairs) of solutions in the reference set for creating combined solutions, and a *solution combination method* to transform the subsets of solutions into combined solutions. Figure 6 outlines the designed procedure.

[Insert Figure 6 here]

$GeneratePool()$ is used at the beginning of the search to build the set $Pool$ of bs^2 solutions. To generate the solutions, we propose the stochastic construction procedure used for GRASP (see Section 3 and Figure 4) with priority rule 1 (maximum ranked positional weight), cardinality-based RCL with $b = 4$, and priority-based selection probability. This procedure and settings not only provide solutions of reasonable quality (as it is shown in Section 6.2) but also diverse solutions.

$GenerateRefSet(Pool)$ constructs the reference set $RefSet$ using the solutions in the current set $Pool$. The solutions in $RefSet$ will be combined to generate new solutions. According to the SS methodology, the $RefSet$ solutions should be good, according to its δ value, and diverse, according to a distance metric. We evaluate the distance between two solutions $s1$ and $s2$, $d(s1, s2)$, as follows: $d(s1, s2) = \sum_{i=1}^{n} |wa(i, s1) - wa(i, s2)|$, where $wa(i, s)$ is the workstation assigned to task i in solution s. For example, the distance between the two designs shown in Table 1 is $|1-1| + |2-3| + |2-2| + |3-2| + |4-3| + |3-4| + |4-5| + |5-4| + |5-5| = 6$. The construction of the reference set starts with the addition of the $bs/2$ best local optima in the $Pool$ solutions (and the original solutions are removed from $Pool$). The local search proposed is *LSPa*. Then, the minimum distance from each solution in $Pool$ to the solution in $RefSet$ is computed. The solution with the maximum of these minimum distances is selected. This solution is added to $RefSet$ and deleted from $Pool$ and the minimum distances are updated. This process is repeated $bs/2$ times.

$CombineSolutions(RefSet)$ applies a combination method, $Combine(s1, s2)$, to all pairs of solutions $s1$ and $s2$ in the current $RefSet$. Since $Combine(s1, s2)$ may produce a different solution from $Combine(s2, s1)$, it is therefore applied bs^2 at each iteration of SS (recall that the size of $RefSet$ is bs). $Combine(s1, s2)$ does not operate directly with the solutions but with their representations as sequences of tasks ordered according to their execution. For example, the first design shown in Table 1 is represented as the sequence $(1,2,3,4,6,5,7,8,9)$.

The design that we propose of $Combine(s1, s2)$ is known in the GA literature as the fragment reordering crossover, and it has been specifically designed for ALBPs (Rubinovitz and Levitin, 1995). Additionally, it has the advantage that always returns a feasible sequence of tasks (that is, the precedence constraints are satisfied) and no reparation mechanism is needed. The combination method works as follows. First, the combined sequence of tasks, $CSeq$, is equal to the tasks sequence of $s1$. Then all the tasks of a random fragment in $CSeq$ are reallocated within this fragment according to the order in which they appeared in the tasks sequence of $s2$. Finally, the sequence of tasks $CSeq$ has to be decoded into a solution s. We use the decoding method proposed in Rubinovitz and Levitin (1995). It aims to divide the tasks sequence seq between the m workstations (without changing the order of the tasks) looking for a maximum equality between the workloads of all workstations. The decoding method consists in executing the recursive division method $Divide(s, seq, fp, lp, M, j)$ as $Divide(s, CSeq\ ,1, n, m, 1)$. Figure 7 shows the division method, where $T(j, seq)$ is the task at position p of the sequence seq (and recall that $s(j)$ is the set of tasks assigned to workstation j).

[Insert Figure 7 here]

Finally, $Improve(Pool)$ applies the local search *LSPa* to the best $bs/2$ solutions in $Pool$ (and the original solutions are removed).

6. Computational Experiments

The designed procedures were implemented in Java SE 1.6.21 and run on a PC 3.16 GHz Pentium Intel Core 2 Duo E8500 with 3.46 GB of RAM. Our experiments to compare existing procedures with our proposed procedures are performed on the same 301 problem instances used in (Pastor *et al.* 2012). The test set is generated using seventeen cases with characteristics listed in Table 2. For each case, Table 2 shows its name, the number of tasks, the minimum, maximum and average processing times, the order strength of the precedence graph, and the range on the number of workstations.

[Insert Table 2 here]

6.1. Fine-tuning of Search Parameters

Our computational experience starts with a set of preliminary experiments for parameter tuning. For each of the seventeen cases in Table 2, we generate two instances. The first one with number of workstations set at the minimum value in the last column of the table minus one. The second instance has a number of workstations equal to the maximum value in the last column of the table plus one. For example, for the Arcus1 case we generate two training instances, one with $m = 2$ and one with $m = 23$. This results in a training set of 34 instances.

The fine-tuning of GRASP involves the selection of a priority rule (12 choices), the strategies to build the RCL (cardinality-based and value-based) together with the associated parameters (α and b), and the strategies to select a task from the RCL (equal probability and priority-based probability). We consider five values for α (from 0.1 to 0.5 in increments of 0.1) and four values for b (2, 3, 4 and 5). This results in a full factorial design with 216 ($12 \times 5 \times 2 + 12 \times 4 \times 2$) parameter settings. Each setting is applied to the training set of 34 instances, resulting in 7,344 executions of GRASP. The number of iterations was set to 1,000 for these runs. This experiment identifies the following three top parameter settings (according to the average δ value obtained) for the standard GRASP (equal selection probability) and the non-standard GRASP (priority-based selection probability) implementations, where PR is the priority rule and $\bar{\delta}$ is the average δ value:

- Standard GRASP: $PR = 9$, $\alpha = 0.1$ ($\bar{\delta} = 1.59167$), $PR = 9$, $\alpha = 0.2$ ($\bar{\delta} = 1.7577$) and $PR = 4$, $\alpha = 0.2$ ($\bar{\delta} = 1.99617$).
- Non-standard GRASP: $PR = 1$, $b = 4$ ($\bar{\delta} = 1.57504$), $PR = 7$, $b = 4$ ($\bar{\delta} = 1.59290$) and $PR = 1$, $b = 3$ ($\bar{\delta} = 1.75245$).

We applied the *non-parametric Friedman test* for multiple correlated samples to the solutions obtained by each of the standard and non-standard GRASPS. This test computes, for each instance, the rank value of each setting according to solution quality (where rank 1 is assigned to the best method and rank 3 to the worst one). Then, it calculates the average rank values of each method across the 34 training instances solved. If the averages differ greatly, the associated p-value or significance will be small. The resulting p-value of 0.55 and 0.58 obtained in this experiment for the standard and non-standard GRASPs, respectively, indicates that there are not statistically significant differences among their respective three best settings. Specifically, the rank values produced by this test for the standard GRASP are 1.91 ($PR = 9$, $\alpha = 0.1$), 1.96 ($PR = 9$, $\alpha = 0.2$) and 2.13 ($PR = 4$, $\alpha = 0.2$), and for the non-standard GRASP are 1.90 ($PR = 7$, $b = 4$), 1.99 ($PR = 1$, $b = 4$) and 2.12 ($PR = 1$, $b = 3$).

Since the best parameter settings for each type of GRASP are not statistically different, we focus on the settings that return the lowest average δ value. We refer as GRASP1 to the standard GRASP $PR = 9$, $\alpha = 0.1$, and as GRASP2 to the non-standard GRASP with $PR = 1$, $b = 4$. It must be noted that the two best methods among the 216 GRASP variants tested do not apply the same strategy neither to build the RCL nor to select an element from it. In particular, GRASP1 builds the RCL based on the value of the candidate elements, while GRASP2 simply selects a pre-established number of candidate elements to construct RCL. On the other hand, GRASP1 selects an element in RCL according to a uniform distribution, as it is customary in the GRASP methodology, while GRASP2 employs a biased selection process.

The tabu search implementation has five parameters: a priority rule, κ and γ for the construction phase and $TabuTenure$ and $MaxIter$ for the improvement phase. Since, in our design, the $TabuTenure$ depends on the number of tasks in the problem, we tune this parameter indirectly by searching for the best value of θ and making $TabuTenure = \theta n$. We used CALIBRA (Adenso-Díaz and Laguna 2006) to find effective values for these parameters. CALIBRA is a tool specifically designed for fine-tuning the parameters of algorithms and is based on using conjointly Taguchi's fractional factorial experimental designs and a local search procedure. This tool, which can be downloaded at http://coruxa.epsig.uniovi.es/~adenso/file_d.html, automatically returns the best parameter values. We executed CALIBRA runs of 1000 iterations with the following settings:

- Priority rule = {1, 2, 3, …, 12}
- $\kappa = \{0.01, 0.02, \dots, 1.0\}$
- $\gamma = \{1.0, 1.1, \dots, 10.0\}$
- $\theta = \{0.25, 0.5, 0.75\}$
- $MaxIter = \{50, 100, 150, 200\}$

CALIBRA identified the best parameter setting as priority rule 1, $\kappa = 0.06$, $\gamma = 4.8$, $\theta = 0.25$ and $MaxIter = 150$. We refer to this setting as TS.

The scatter search implementation has one parameter: bs. We calibrated this parameter considering the following values: $bs = \{2, 4, \dots, 30\}$. Again, the number of iterations was set to 1,000 for these runs. The three best settings are identified as $bs = 18$ ($\bar{\delta} = 1.55453$), $bs = 14$ $(\bar{\delta} = 1.61131)$ and $bs = 16$ ($\bar{\delta} = 1.62152$). We performed a Friedman test and the resulting p-value of 0.26 indicates that there are not statistically significant differences among the 3 settings. Specifically, the rank values produced by this test are 1.81 ($bs = 16$), 2.06 ($bs = 18$) and 2.13 ($bs = 14$). Thus, we focus on the setting that return the best average δ value ($bs = 18$) and refer to this setting as SS.

6.2. Comparison of our GRASP, TS and SS Implementations

We now test the performance of the GRASP configurations that were identified as the best (GRASP1 and GRASP2) and the fine-tuned tabu search and scatter search procedures (TS and SS, respectively). For this experiment, we consider all 301 test instances and a run time of 1,000 seconds per procedure and instance. Quality is measured with the average δ values (denoted by $\bar{\delta}$) and the number of times a procedure matches the best-known solution (denoted by #*Best*). We first compare the performance of the procedures with and without the improving phase. The results are shown in Table 3.

[Insert Table 3 here]

Overall, results in Table 3 confirm the effectiveness of including an improving phase. All procedures yield better results when an improving phase is added. According to these results, GRASP2 outperforms the other competing procedures. It seems that in this context there is no gain in adding memory to the construction process (as TS and SS do). However, there seems to be an advantage in using a non-uniform probabilistic selection of the tasks within the GRASP construction.

We applied a Friedman test to the best solutions obtained by each of the four methods with the improving phase. The resulting p-value of 0.000 obtained in this experiment clearly indicates that there are statistically significant differences among the four methods tested. Specifically, the rank values produced by this test are 1.70 (GRASP2), 2.46 (SS), 2.88 (GRASP1), and 2.96 (TS). If we apply the same test to the four methods without the improvement phase, we also obtain a p-value of 0.000; but now the ranks are 1.74 (GRASP2), 1.84 (SS), 3.14 (TS), and 3.27 (GRASP1).

Considering that GRASP2 and SS obtain the best average results, we compared both with a well-known nonparametric test for pairwise comparisons: the *Wilcoxon test*. This test is designed to answer the question: Do the two samples (i.e., the objective function values of the solutions obtained by GRASP2 and SS) represent two different populations? We perform two pairwise comparisons, with and without the local search. The resulting p-value of 0.000 with the improving phase indicates that the values belong to two different populations. On the other hand, a p-value of 0.039 is obtained without the improving phase. Since this last value is borderline, we apply the *non-parametric sign test* for paired samples. The resulting p-value of 0.95 indicates that we cannot ensure that GRASP2 performs better than SS without their improving phases. We can therefore conclude that when solving LB-ALBP, the non-standard GRASP2 (with the improving phase), in which the random selection from the RCL is biased, is the best method.

Next, we analyze the influence of the characteristics of the instances (as outlined in Table 2) on the quality of the results. Specifically, we create subset of instances according to the order strength and the number of tasks. The order strength is respectively classified (according to Pastor *et al*. 2012) as low, medium and high according to the ranges (22.67, 25.80), (40.38, 59.50) and (77.55, 83.82). Likewise, the number of tasks is respectively considered low, medium and high according to the ranges (29, 35), (45, 111) and (148, 297). Tables 4 and 5 report the results obtained by each procedure for combinations of order strength and number of tasks. The column headers use L, M and H for low medium and high, respectively. The first letter is for the order strength and the second letter for the number of tasks. This is followed by the number of instances in the subset, which are shown between parentheses. Table 4 shows the average δ-values and Table 5 contains the number of best solutions found by each procedure.

[Insert Table 4 here]

[Insert Table 5 here]

The results in Tables 4 and 5 indicate that GRASP2 is on average superior to the other two procedures regardless of the characteristics of the instances although there are two exceptions. First, when the order strength is low and the number of tasks is medium, TS slightly outperforms GRASP2, with an average objective function value that is 0.6% better and 2 more best solutions. Second, when the order strength is medium and the number of tasks is high, SS slightly outperforms GRASP2, with an average objective function value that is 0.4% better and 4 more best solutions.

6.3. Comparison against the State of the Art

In our last experiment, we compare GRASP2 and SS, both with their respective improvement method, with the best heuristic for the LB-ALBP reported in the literature, namely *V-LSPa* (Pastor, *et al*. 2012). Since this method presents an average CPU time of 50 seconds, we run our both procedures for the same running time per procedure and instance to perform a fair comparison. Table 6 reports the average δ values (denoted by $\bar{\delta}$) and the number of times a procedure matches the best-known solution (denoted by #*Best*) on this experiment.

[Insert Table 6 here]

Table 6 shows that GRASP2 seems to improve upon SS on a short term horizon (i.e., when both methods are run for 50 seconds), since it is able to obtain a smaller value of $\bar{\delta}$ and a larger number of best solutions. Moreover, GRASP2 clearly outperforms the best previously published method, *V-LSPa*. We performed a Friedman test in an attempt to draw some statistically significant conclusions. The resulting p-value of 0.000 indicates that there are statistically significant differences among the 3 methods. Specifically, the rank values produced by this test are 1.70 (GRASP2), 1.82 (SS), and 2.47 (*V-LSPa*). Thus, it is confirmed that our two methods, GRASP2 and SS, obtain better solutions than the best published heuristic. Moreover, the p-value of 0.119 obtained with the Wilcoxon test and the p-value of 0.087 obtained with the sign test when comparing GRASP2 and SS outlines that we cannot ensure that GRASP2 performs better than SS.

To complement the information shown in Table 6, we run our best method, GRASP2, and the best previous method, *V-LSPa*, for 1000 seconds, and track their solution quality over the execution of the procedure. Specifically, the profile of the $\bar{\delta}$ values for a 1,000-second execution is shown in Figure 8. This figure shows the value of $\bar{\delta}$ for the best solutions found at intervals of 10 seconds.

[Insert Figure 8 here]

From Figure 7 we can confirm that GRASP2 obtains better solutions than *V-LSPa* for similar computing time. The $\bar{\delta}$ value for GRASP2 within 50 seconds is 1.81175 and, even within the first 10 seconds, the average solution quality of 2.00885 is considerably better than *V-LSPa's* (2.63578).

7. Conclusions and future research

The Assembly Line Balancing is a difficult combinatorial optimization problem. Our goal is to improve the state of the art resolution of the lexicographical bottleneck version of the assembly line problem, which can be considered particularly challenging. In particular, we have implemented 48 different GRASPs, 12 tabu search variants and 1 scatter search procedure. Moreover, we have tested this large number of variants according to different search parameters on several types of instances. Through extensive experimentation, we have been able to determine that the designed best heuristics outperform the incumbent state of the art. We show that our proposals benefit from the addition of enhanced search strategies. Additionally, we can conclude that in this problem, the non-standard GRASP implementation, in which the random selection from the RCL is biased, performs better than the classical one based on a uniform random distribution.

The solution of the LB-ALBP has been studied in the context in which a single model is processed in the line and its shape is straight. An interesting issue that is worth exploring in the future is to extend the LB-ALBP in the presence of mixed-model assembly lines or U-shaped lines.

Acknowledgments

This research has been partially supported by the *Ministerio de Educación Cultura y Deporte o*f Spain (Grant Ref. PRX12/00016) and by the *Ministerio de Economía y Competitividad* of Spain (Grant Ref. TIN2012-35632). We would like to thank the reviewers for their valuable comments and suggestions that improved the quality of the paper.

References

Adenso-Díaz, B., Laguna, M., 2006. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54, 99-114.

Battaïa, O., Dolgui, A., 2013. A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, 142, 259-277.

Baybars, I., 1986. A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, 32, 909-932.

Becker, C., Scholl, A., 2006. A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168, 694-715.

Boysen, N., Fliedner, M., Scholl, A., 2006. A classification of assembly line balancing problems. *Working paper 12/2006*, Friedrich-Schiller-Universität Jena: Thuringia, Germany.

Boysen, N., Fliedner, M., Scholl, A., 2007. A classification of assembly line balancing problems. *European Journal of Operational Research*, 183, 674-693.

Boysen, N., Fliedner, M., Scholl, A., 2008. Assembly line balancing: Which model to use when? *International Journal of Production Economics*, 111, 509-528.

Capacho, L., Pastor, R., Dolgui, A., Gunshinskaya, O., 2009. An evaluation of constructive heuristic methods for solving the alternative subgraphs assembly line balancing problem. *Journal of Heuristics*, 15, 109-132.

Cheshmehgaz, H.R., Haron, H., Kazemipour, F., Desa, M.I., 2012. Accumulated risk of body postures in assembly line balancing problem and modelling through a multi-criteria fuzzy-genetic algorithm. *Computers & Industrial Engineering*, 63, 503-512.

Corominas, A., Pastor, R., 2009. A note on "A comparative evaluation of assembly line balancing heuristics". *International Journal of Advanced Manufacturing Technology*, 44, 817.

Corominas, A., Ferrer, L., Pastor, R., 2011. Assembly line balancing: general resource-constrained case. *International Journal of Production Research*, 49, 3527-3542.

Cortés, P., Muñuzuri, J., Onieva, L., Larrañeta, J., Vozmediano J.M., Alarcón, J.C., 2006. Andalucía assesses the investment needed to deploy a fiber-optic network. *Interfaces*, 36, 105-117.

Ding, F-Y., Zhu, J., Sun, H., 2006. Comparing two weighted approaches for sequencing mixed-model assembly lines with multiple objectives. *International Journal of Production Economics*, 102, 108-131.

Duarte, A., Martí, R., 2007. Tabu search and GRASP for the maximum diversity problem. *European Journal of Operational Research*, 178, 71-84.

Feo, T.A., Resende, M.G.C., 1989. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8, 67-81.

Gamberini, R., Grassi, A., Rimini, B., 2006. A new multi-objective heuristic algorithm for solving the stochastic assembly line re-balancing problem. *International Journal of Production Economics*, 102, 226-243.

Glover, F., 1977. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8, 156-166.

Glover, F., 1986. Future paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 5, 533-549.

Glover, F., Laguna, M., 1997. *Tabu Search*. Kluwer Academic Publisher.

Glover, F., 1998. A Template for Scatter Search and Path Relinking. In: Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (Eds.), Artificial Evolution*, Lecture Notes in Computer Science* 1363, Springer, pp. 13-54.

Hoffman, T.R., 1992. EUREKA: a hybrid system for assembly line balancing. *Management Science*, 38, 39-47.

Inman, R.R., Leon, M., 1994. Scheduling duplicate serial stations in transfer lines. *International Journal of Production Research*, 32, 2631-2644.

Johnson, R.V., 1988. Optimally balancing large assembly lines with Fable. *Management Science*, 34, 240-253.

Kao, E.P.C., Queyranne, M., 1982. On dynamic programming methods for assembly line balancing. *Operations Research*, 30, 375-390.

Laguna, M., Martí, R., 2003. *Scatter Search*. Kluwer Academic Publisher.

Lusa, A, 2008. A survey of the literature on the multiple or parallel assembly line balancing problem. *European Journal of Industrial Engineering*, 2, 50-72.

Martino, L., Pastor, R., 2010. Heuristic procedures for solving the general assembly line balancing problem with setups. *International Journal of Production Research*, 48, 1787-1804.

Merengo, C., Nava, F., Pozzetti, A., 1999. Balancing and sequencing manual mixed-model assembly lines. *International Journal of Production Research*, 37, 2835-2860.

Miltenburg, J., 2002. Balancing and scheduling mixed-model U-shaped production lines. *International Journal of Flexible Manufacturing Systems*, 14, 1119-151.

Moodie, C.L., Young, H.H., 1965. A heuristic method of assembly line balancing for assumptions of constant or variable work element times. *Journal of Industrial Engineering*, 16, 23-29.

Park, K., Park, S., Kim, W., 1997. A heuristic for an assembly line balancing problem with incompatibility, range, and partial precedence constraints. *Computers & Industrial Engineering*, 32, 321-332.

Pastor, R., 2011. LB-ALBP: the lexicographic bottleneck assembly line balancing problem. *International Journal of Production Research*, 49, 2425-2442.

Pastor, R., Chueca, I., García-Villoria, A., 2012. A heuristic procedure for solving the Lexicographic Bottleneck Assembly Line Balancing Problem (LB-ALBP). *International Journal of Production Research*, 50, 1862-1876.

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

Ponnambalam, S.G., Aravindan, P., Mogileeswar, G., 1999. A comparative evaluation of assembly line balancing heuristics. *International Journal of Advanced Manufacturing Technology*, 15, 577-586.

Rachamadugu, R., Talbot, B., 1991. Improving the equality of workload assignments in assembly lines. *International Journal of Production Research*, 29, 619-633.

Rekiek, B., Lit, P., Delchambre, A., 2002. Hybrid assembly line design and user's preferences. *International Journal of Production Research*, 40, 1095-1111.

Rubinovitz, J., Levitin, G., 1995. Genetic algorithm for assembly line balancing. *International Journal of Production Economics*, 41, 343-354.

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

Scholl, A., Klein, R., 1997. SALOME: a bidirectional branch and bound procedure for assembly line balancing. *Informs Journal on Computing*, 9, 319-334.

Scholl, A., Voß, S., 1996. Simple assembly line balancing-Heuristic approaches. *Journal of Heuristics*, 2, 217-244.

Talbot, F.B., Patterson, J.H., 1984. An integer programming algorithm with network cuts for solving the assembly line balancing problem. *Management Science*, 30, 85-99.

Talbot, F., Patterson, J.H., Gehrlein, W.V., 1986. A comparative evaluation of heuristic line balancing techniques. *Management Science*, 32, 431-453.

Wee, T.S., Magazine, M.J., 1982. Assembly line balancing as generalized bin packing. *Operations Research Letters*, 1, 56-58.

White, W.W., 1961. Comments on a paper by Bowman. *Operations Research*, 9, 274-276.

Design	W1	W2	W3	W4	W5
1	1	2, 3	4, 6	5, 7	8,9
2	1	3, 4	2, 5	6, 8	7,9

Table 1 Task assignments of two designs

	Number	Processing Time			Order	Number of
Name	of Tasks	Minimum	Maximum	Average	Strength	Workstations
Arcus1	83	233	3691	912.1	59.09	3 to 22
Arcus2	111	10	5689	1354.9	40.38	3 to 27
Barthold	148	3	383	38.1	25.80	3 to 15
Barthol2	148	1	83	28.6	25.80	27 to 51
Buxey	29	1	25	11.2	50.74	7 to 14
Gunther	35	1	40	13.8	59.50	6 to 15
Hahn	53	40	1775	264.6	83.82	3 to 10
Killbridge	45	3	55	12.3	44.55	3 to 11
Lutz1	32	100	1400	441.9	83.47	8 to 12
Lutz2	89	1	10	5.4	77.55	9 to 28
Lutz3	89	1	74	18.5	77.55	3 to 23
Mukherje	94	8	171	44.8	44.80	3 to 26
Sawyer	30	1	25	10.8	44.83	7 to 14
Scholl	297	5	1386	234.5	58.16	25 to 52
Tonge	70	1	156	50.1	59.42	3 to 24
Warnecke	58	7	53	26.7	59.10	3 to 29
Wee-Mag	75	2	27	20.0	22.67	3 to 30

Table 2. Test problems

Measure	With improving phase				Without improving phase			
	GRASP1	GRASP2	TS	SS	GRASP1	GRASP2	TS	SS
$\bar{\delta}$	2.10587	1.62926	2.04308	1.81935	2.87519	1.91586	2.61364	2.05554
$\#Best$	96	240	97	121	37	82	31	75

Table 3 Performance comparison with and without the improving phase

Procedure	LM (28)	LH (38)	ML (26)	MM (127)	MH (28)	HL (5)	HM (49)
GRASP1	3.99137	1.28776	3.36447	1.52528	1.13074	3.50804	2.91401
GRASP2	3.96106	0.68476	2.93308	1.22891	0.78598	3.50687	1.66536
TS	3.93725	0.73804	3.42808	1.70481	1.02533	4.52720	2.44273
SS	4.13795	0.86063	3.10029	1.47599	0.78298	3.84472	1.83373

Table 4 $\bar{\delta}$ values for each combination of order strength and number of tasks

Procedure	LM (28)	LH (38)	ML (26)	MM (127)	MH (28)	HL (5)	HM (49)
GRASP1	19	9	13	39	1	1	14
GRASP2	22	31	26	104	10	4	43
TS	24	23	5	32	3	0	10
SS	13	17	15	35	14	1	26

Table 5 $\#Best$ values for each combination of order strength and number of tasks

Procedure	GRASP2	SS	*V-LSPa*
$\bar{\delta}$	1.81175	1.87038	2.63578
$\#Best$	163	149	75

Table 6 Performance comparison with the best previous method

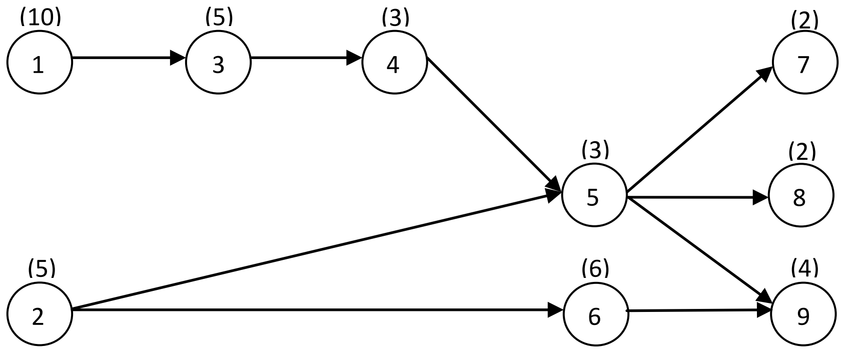

Figure 1. Precedence graph (task time in parenthesis)

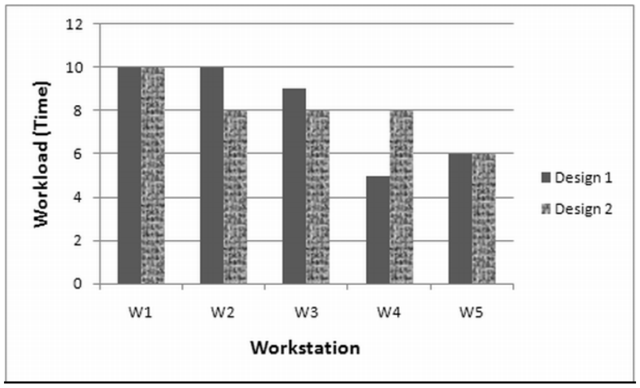

Figure 2 Workload profiles of two designs

```
Calculate the lower bound on the cycle time $LB = w_r(1)$
Initialize $CT \leftarrow LB - 1$
**Repeat**
    $CT \leftarrow CT + 1$
    Apply a greedy heuristic for the SALBP-1 with cycle time equal to $CT$
**Until** number of workstations in the solution is less than or equal to $m$
```

Figure 3 Simple heuristic for the SALBP-2

```
$U \leftarrow \{1, \dots, n\}$                                        List of unassigned tasks
$j \leftarrow 1; s(j) \leftarrow \emptyset; w(s,j) \leftarrow 0$                          Start from workstation 1
**Repeat**
    $CL \leftarrow \{i \in U: P_i \cap U = \emptyset \wedge t_i + w(s,j) \le CT\}$     Build candidate list
    **If** $CL \neq \emptyset$ **then**
        $RCL \leftarrow BuildRCL(CL)$                              Build RCL
        $i^* \leftarrow ChooseTask(RCL)$                           Select a task from $RCL$
        $s(j) \leftarrow s(j) \cup \{i^*\}$                              Add chosen task to workstation j
        $w(s,j) \leftarrow w(s,j) + t_{i^*}$                          Update workload of workstation j
        $U \leftarrow U \setminus \{i^*\}$                                  Update list of unassigned tasks
    Else
        $j \leftarrow j + 1; s(j) \leftarrow \emptyset; w(s,j) \leftarrow 0$                  Open a new workstation
    End if
**Until** $U = \emptyset$                                            End when all jobs are assigned
```

Figure 4 Semi-greedy procedure for SALBP-1

```
$q(i,j) \leftarrow 0; f(i,j) \leftarrow 0$                           Initialize memory structures
**Repeat**
    $s \leftarrow Construction(q, f)$                        Apply constructive heuristic
    $s \leftarrow Improvement(s)$                           Apply short-term tabu search
    $Update(s^*, q, f)$                                  Update best solution $s^*$ and memory
**Until** stopping criterion satisfied
```

Figure 5 Tabu search for the LB-ALBP

```
$Pool \leftarrow GeneratePool()$                             Initialize pool of solutions
**Repeat**
    $RefSet \leftarrow GenerateRefSet(Pool)$               Update *RefSet*
    $Pool \leftarrow CombineSolutions(RefSet)$             Update *Pool* by combining solutions
    $Improve(Pool)$                                     Improve the best solutions in *Pool*
    $Pool \leftarrow Pool \cup RefSet$                              Add the *RefSet* solutions to *Pool*
**Until** stopping criterion satisfied
```

Figure 6 Scatter search for the LB-ALBP

```
**if** $M = 1$ then $s(j) \leftarrow \cup_{p=fp}^{lp} \{T(p, seq)\}$
**Else**
    $M_1 \leftarrow \lfloor M/2 + 0.5 \rfloor$; $M_2 \leftarrow M - M_1$
    $c^* \leftarrow argmin_{c=fp}^{lp} \left| \frac{\sum_{p=fp}^{c} t_{T(p,seq)}}{\sum_{p=c+1}^{lp} t_{T(p,seq)}} - \frac{M_1}{M_2} \right|$
    $Divide(s, seq, fp, c^*, M_1, j)$
    $Divide(s, seq, c^* + 1, lp, M_2, j + M_1)$
**End if**
```

Figure 7. The division method $\boldsymbol{Divide}(\boldsymbol{s}, \boldsymbol{seq}, \boldsymbol{fp}, \boldsymbol{lp}, \boldsymbol{M}, \boldsymbol{j})$

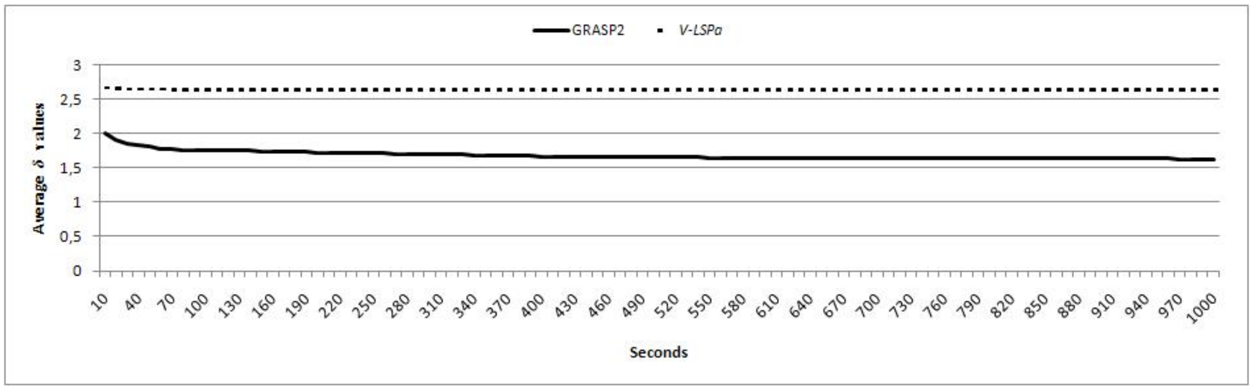

Figure 8 Average $\boldsymbol{\delta}$ values of the best solution found by GRASP2