Journal of the Operational Research Society (2013) 64, 1607–1613

www.palgrave-journals.com/jors/

Robust balancing of straight assembly lines with interval task times☆

E Gurevsky*, Ö Hazır, O Battaïa and A Dolgui
École Nationale Supérieure des Mines de Saint-Étienne, Saint-Étienne, France

This paper addresses the balancing problem for straight assembly lines where task times are not known exactly but given by intervals of their possible values. The objective is to assign the tasks to workstations minimizing the number of workstations while respecting precedence and cycle-time constraints. An adaptable robust optimization model is proposed to hedge against the worst-case scenario for task times. To find the optimal solution(s), a breadth-first search procedure is developed and evaluated on benchmark instances. The results obtained are analysed and some practical recommendations are given.
Journal of the Operational Research Society (2013) **64,** 1607–1613. doi:10.1057/jors.2012.139
Published online 5 December 2012

Keywords: assembly line balancing; uncertainty; robust optimization; branch-and-bound algorithm

1. Introduction

A straight assembly line represents a sequence of workstations that function simultaneously. A product item is released at the first workstation and then visits all workstations in a linear order. The transfer of a product item from a workstation to the next is synchronized via an automatic material handling system.

The problem here deals with assigning a given set of elementary tasks $V=\{1,2,\ldots,n\}$, required for completing a product, to workstations of the line. The objective is to minimize the number of workstations required taking into account *precedence*, *exclusion* and *cycle-time constraints*.

The *precedence constraints* define non-strict partial order relations among tasks and can be presented by an acyclic direct graph $G=(V,\mathcal{A})$. An arc (i,j) belongs to $\mathcal{A}$ iff task j must be assigned to a workstation that does not precede the workstation where task i is assigned. However, this partial order relation does not exclude assigning tasks i and j to the same workstation.

The *exclusion constraints* define the pairs of tasks that cannot be assigned to the same workstation because of their technological incompatibility. These constraints are represented by a family E of pairs of V such that the tasks of a subset $e\in E$ cannot be assigned to the same workstation.

The *cycle-time constraint* requires that the time spent by a product item at a workstation of the line must be less than a given value T_0 calculated on the basis of the line throughput required.

When exclusion constraints do not exist, this problem corresponds to the Simple Assembly Line Balancing Problem of type 1 (SALBP-1, see Baybars, 1986; Battini *et al*, 2007).

Contrary to previous publications, where the task times are assumed to be given and fixed, we consider that their exact values cannot be known *a priori* and even can vary during the life cycle of the line. This is a broader and more realistic assumption. For example, we have some experience working with a company producing car door locks for different automotive groups. This company uses automatic assembly lines. One of their problems is the micro-stoppages of workstations (from 10 to 30 s) that is especially caused by an error in automatic part positioning (or blocking). In such cases, an operator has to manually open the workstation and unblock or reposition the part. The task times are reliable and fixed, but when a stoppage occurs the corresponding task time is increased. The micro-stoppages are unforeseeable but concern only certain tasks. Therefore, an attentive management is able to give the list of tasks subject to micro-stoppages and evaluate the minimum and maximum stoppage time. This is one of many examples where we need to take into account this and other forms of uncertainties at the preliminary design stage of assembly lines.

In the literature, to model the variability of task times, authors used either normally distributed independent random variables with known means and variances (Erel *et al*, 2005; Chiang and Urban, 2006; Urban and Chiang, 2006; Ağpak and Gökçen, 2007; Baykasoğlu and

☆This research was financially supported by Saint-Étienne Metropole government and the European Project AMEPLM.
*Correspondence: E Gurevsky, LIMOS, Henri Fayol Institute, École Nationale Supérieure des Mines de Saint-Étienne, 158, cours Fauriel, 42023 Saint-Étienne Cédex 2, 42023, France.
E-mail: evgeny.gurevsky@gmail.com

Özbakır, 2007; Gamberini *et al*, 2009; Özcan, 2010; see also Dolgui and Proth, 2010, Chap. 8) or fuzzy numbers with given membership functions (Tsujimura *et al*, 1995; Gen *et al*, 1996; Hop, 2006). However, the application of such models in practice is often impossible because of insufficient preliminary information in order to deduct the required probability or possibility distribution functions. As a consequence, we assume that the task times are given by intervals of possible values where the lower and upper bounds determine the minimal and maximal admissible values of the corresponding task time, respectively. Distributions of probability cannot be known beforehand.

To the best of our knowledge, this model was first introduced in line-balancing context for SALBP-2, where the number of workstations is known and the objective is to minimize the line cycle time (Hazır and Dolgui, 2011).

A number of studies in optimization under interval and/or scenario uncertainty exist in the literature. Among the approaches frequently used, the min-max (or absolute robust) and min-max regret approaches must be mentioned (Mausser and Laguna, 1999; Aissi *et al*, 2009; Gabrel and Murat, 2010; Dolgui and Kovalev, 2012). The goal of the min-max approach is to find the solution that minimizes the worst-case performance, while the min-max regret approach seeks the solution that minimizes the maximal deviation from the optimum across all possible scenarios of problem parameters. The principal weakness of these two approaches is their 'conservatism', since they are based on the worst-case scenario that may never happen in practice.

To control the degree of conservatism, a new robust approach was proposed in Bertsimas and Sim (2003). It was remarked that it is quite improbable that all uncertain parameters change their values at the same time. Therefore, the authors assumed that only a subset of parameters could negatively affect the solution. Formally, each uncertain parameter is represented by an interval of possible values, the nominal value of a parameter is the middle of the interval. For each constraint i, the authors define a parameter $\Gamma_i > 0$ that is less than the number of uncertain parameters of this constraint. A solution is called robust (in sense of Bertsimas and Sim) if it is optimal among the solutions that remain feasible when at most Γ_i parameters of each constraint can deviate from their nominal values. This approach has been also applied to other optimization problems such as: inventory control (Bertsimas and Thiele, 2006), project scheduling (Hazır *et al*, 2011), portfolio optimization (Moon and Yao, 2011), and production planning (Alem and Morabito, 2011).

The remainder of the paper is organized as follows. In Section 2, deterministic and robust optimization models of SALBP-1 are introduced. A breadth-first search (BFS) procedure to find optimal solutions for the robust model proposed is described in Section 3. Experimental results for benchmark instances are presented in Section 4. Final remarks and conclusions are given in Section 5.

2. Problem definition

2.1. Deterministic model

In the deterministic model, elementary tasks from the set V are associated with a vector $\mathbf{t} = (t_1, t_2, \ldots, t_n) \in \mathbb{R}_+^n$ of task times, where t_j is the time of task $j \in V$ and $\mathbb{R}_+$ is the set of all positive real numbers. Using the notations introduced above, the deterministic mathematical model of SALBP-1 is expressed as follows:

$$\text{Minimize} \quad \sum_{k=1}^{m} z_k, \tag{1}$$

subject to

$$\sum_{k=1}^{m} x_{jk} = 1, \quad \forall j \in V, \tag{2}$$

$$\sum_{k=1}^{m} k x_{ik} \leqslant \sum_{k=1}^{m} k x_{jk}, \quad \forall (i,j) \in \mathcal{A}, \tag{3}$$

$$x_{ik} + x_{jk} \leqslant 1, \quad \forall \{i,j\} \in E, \quad \forall k \in \{1,2,\ldots,m\}, \tag{4}$$

$$\sum_{j \in V} t_j x_{jk} \leqslant T_0 z_k, \quad \forall k \in \{1,2,\ldots,m\}, \tag{5}$$

$$z_k \in \{0,1\}, \quad \forall k \in \{1,2,\ldots,m\},$$

$$x_{jk} \in \{0,1\}, \quad \forall j \in V, \quad \forall k \in \{1,2,\ldots,m\},$$

where m is the maximal admissible number of workstations; z_k equals 1 if station k is opened, 0 otherwise; x_{jk} equals 1 if task j is assigned to workstation k, 0 otherwise; (1) represents the objective function; the equality (2) imposes that each task is assigned to only one workstation; the precedence constraints are modelled by the inequalities (3); the inequalities (4) introduce the exclusion constraints; the inequalities (5) check the cycle-time constraint.

2.2. Robust model

As it was mentioned in Section 1, we suppose that each t_j, $j \in V$ is given by an interval $[a_j, b_j]$, where $0 < a_j \leqslant b_j$. The nominal value of the task time is the lower bound of the interval, while the difference $c_j := b_j - a_j$ represents the limit level of possible retardations, delays or stoppages for t_j. It is obvious that the worst-case scenario of task times (WCSTT) corresponds to $t_j = b_j$ for each $j \in V$. Since this situation is quite improbable in practice (or impossible as in our case of the assembly line for car door locks), several supplementary assumptions must be taken into account.

We assume that each task j from V can be either *uncertain* or *deterministic*:

- Processing time for an uncertain task can vary and the case $t_j = b_j$ is possible for it.
- Processing time for a deterministic task remains fixed and equals a_j.

The uncertain tasks are not known *a priori*, but its number per workstation is limited and linearly depends on the total number of tasks assigned. To integrate this conception, we introduce a real-valued parameter $\theta \in [0, 1]$, which represents a quota of uncertain tasks per workstation. Thus, for the model considered, following the Bertsimas and Sim principle, a solution is called *robust* if it is optimal among the solutions that remain feasible when at most $\theta \cdot |V_k|$ tasks, for each workstation k, can become uncertain. Here, V_k is the set of tasks assigned to workstation k.

Obviously, if $\theta = 0$ (optimistic case), the problem becomes deterministic and there are no uncertain tasks; if $\theta = 1$ (pessimistic case), all tasks are uncertain and the worst-case scenario will be considered.

Formally, to introduce the parameter θ, the constraints (5) in the deterministic model are replaced with (5′):

$$\sum_{j \in V} a_j x_{jk} + \gamma_k(x) \leqslant T_0 z_k, \quad \forall k \in \{1, 2, \ldots, m\}, \qquad (5')$$

where

$$\gamma_k(x) := \max\left\{ \sum_{j \in V} c_j x_{jk} y_{jk} \Big| \sum_{j \in V} y_{jk} \leqslant \theta \sum_{j \in V} x_{jk}; y_{jk} \in \{0, 1\} \right\}$$

is a knapsack problem that determines the worst-case scenario of the processing times of the tasks assigned to workstation k with respect to parameter θ. Here, binary variable y_{jk} is equal to 1 if task j is considered to be uncertain in workstation k, 0 otherwise.

Almost all the well-known branch-and-bound algorithms developed for the deterministic case of SALBP-1 (see Scholl, 1999, Chap. 4) are based on local lower bound calculation techniques that are not really suitable for the problem considered here. Thus, a new branch-and-bound method is suggested in the next section.

3. Branch and bound algorithm

The branch-and-bound algorithm is based on a systematic exploration of all candidate solutions by developing an enumeration tree. In our context, the tree has the following structure. Each *level* of the tree is composed of *nodes* corresponding to partially constructed solutions. Thus, for example, level k contains all the partial solutions in which k-th workstation is already loaded. Level 0 uniquely contains the root of the tree represented by an empty solution. *Leaf nodes* express either dominated partial solutions or optimal ones.

To find optimal solutions, BFS is used. That is, starting from the root, all possible partial solutions of level 1 are first generated and evaluated. Then all partial solutions of level 2 are generated from the non-dominated solutions of level 1. This continues until all the generated nodes of the current level are leaf nodes.

3.1. Construction of partial solutions

To give the details of the BFS strategy, we must first introduce some notations:

- L_k is the set of all partial solutions of level k,
- V_k^s is the set of tasks of workstation k for solution s.

The following dominance rule is used to reduce search space.

Proposition 1 (Dominance rule) *If s_1 and s_2 belong to L_k, $k \geqslant 1$ such that $\cup_{i=1}^{k} V_i^{s_1} \supseteq \cup_{i=1}^{k} V_i^{s_2}$, then s_2 is dominated.*

Proof Let us introduce the following auxiliary notation: $Q(s, k)$ is the set of feasible complete solutions in which the first k workstations are identical to those of solution s. Then the proposition is equivalent to the following. Prove that $\min\{w^s : s \in Q(s_1, k)\} \leqslant \min\{w^s : s \in Q(s_2, k)\}$, where w^s is the number of workstations of solution s. Assume on the contrary that there is $s' \in Q(s_2, k)$ such that $w^{s'} < \min\{w^s : s \in Q(s_1, k)\}$. Then, construct solution s'' as follows:

- $V_i^{s''} = V_i^{s_1}$ for each $i \in \{1, \ldots, k\}$,
- $V_i^{s''} = V_i^{s'} \setminus U$ for each $i \in \{k+1, \ldots, w^{s'}\}$, where $U = \cup_{i=1}^{k} V_i^{s_1} \setminus \cup_{i=1}^{k} V_i^{s_2}$,

we conclude that $w^{s''} = w^{s'}$ and $s'' \in Q(s_1, k)$. This contradicts the suggestion assumed. □

To construct L_{k+1} from L_k, set W_{k+1} is used. At the beginning, it is composed of all the non-dominated partial solutions of L_k. Thereafter, each solution from W_{k+1} is considered and all possible loads of $(k+1)$-th workstations are generated from it. All the partial solutions generated in this way constitute L_{k+1}. Then the next level is considered. This continues until all tasks are assigned and an optimal solution is obtained.

The load of $(k+1)$-th workstation of a solution $s \in W_{k+1}$ is formed by assigning step-by-step as many tasks as possible (Jackson dominance rule, see Jackson, 1956). To choose a task to be assigned to V_{k+1}^s, the so-called Candidate List $\mathcal{CL}(s, k+1)$ is created. List $\mathcal{CL}(s, k+1)$ contains all the tasks that can be assigned to workstation $k+1$ of solution s. This list is generated in the following manner: the set of unassigned tasks is looked through and task j is added to $\mathcal{CL}(s, k+1)$ if all following conditions are satisfied:[1]

- all predecessors of j have been already assigned,

[1] Without loss of generality, it is assumed that tasks' order numbers in the graph of the precedence constraints are topologically sorted.

- the order number of j is greater than the order number of all the tasks already assigned to workstation $k+1$ (to avoid task permutations),
- j is not linked by exclusion constraints with other tasks already assigned to workstation $k+1$,
- assigning j to workstation $k+1$ does not violate the cycle-time constraint with respect to WCSTT:

$$\sum_{i\in V_k^s\cup\{j\}} a_i + \max\left\{\sum_{i\in U} c_i | U\subseteq V_k^s\cup\{j\}, |U|\leqslant\theta\cdot|V_k^s\cup\{j\}|\right\}\leqslant T_0.$$

If $\mathcal{CL}(s,k+1)=\varnothing$, no more tasks can be assigned to workstation $k+1$, s is moved from W_{k+1} to L_{k+1}. If $\mathcal{CL}(s,k+1)\neq\varnothing$, new $|\mathcal{CL}(s,k+1)|$ solutions are generated from s by adding only one task $j\in\mathcal{CL}(s,k+1)$ to V_k^s at a time. All such solutions are added to W_{k+1}, s is deleted from W_{k+1}. This continues until W_{k+1} is an empty set. An overall scheme of this approach is given by the following algorithm.

$k\leftarrow 1, V_1^s\leftarrow\varnothing, L_1\leftarrow\varnothing, W_1\leftarrow\{s\}$;
repeat
  repeat
    Consider any $s\in W_k$;
    Construct list $C\mathcal{L}(s,k)$;
    if $C\mathcal{L}(s,k)\neq\varnothing$ **then**
      foreach $j\in C\mathcal{L}(s,k)$ **do**
        `/* Construct a new partial solution s' */`
        $s'\leftarrow s$, Add j to $V_k^{s'}$;
        if $\cup_{i=1}^{k} V_i^{s'}=V$ **then**
          return s' `/* Optimal solution is found */`
        else
          Add s' to W_k
    else
      Add s to L_k
    Delete s from W_k;
  until $W_k=\varnothing$;
  Compose W_{k+1} by the non-dominated partial solutions from L_k;
  $k\leftarrow k+1$
until false;

In the next section, this method is evaluated on benchmark instances.

4. Computational experiments

The experiments were carried out on Intel Celeron 550 (2 GHz, 1 GB RAM). The algorithm suggested has been developed in C++ and evaluated on three series of benchmark data sets (see http://www.assembly-line-balancing.de) presented in Table 1, where

Table 1 Benchmark instances

Name	n	T_0	$\|E\|$	*OS, %*
Small size				
Bowman	8	26	4	75.00
Jackson	11	11	10	58.18
Jaeschke	9	10	4	83.33
Mansoor	11	67	7	60.00
Mertens	7	10	2	52.38
Medium size				
Buxey	29	38	54	50.74
Gunther	35	61	81	59.50
Heskia	28	163	342	22.49
Mitchell	21	20	39	70.95
Lutz1	32	2100	83	83.47
Roszieg	25	20	32	71.67
Sawyer	30	38	75	44.83
Large size				
Hahn	53	2663	228	83.82
Lutz2	89	16	21	77.55
Lutz3	89	113	150	77.55
Tonge	70	235	572	59.42

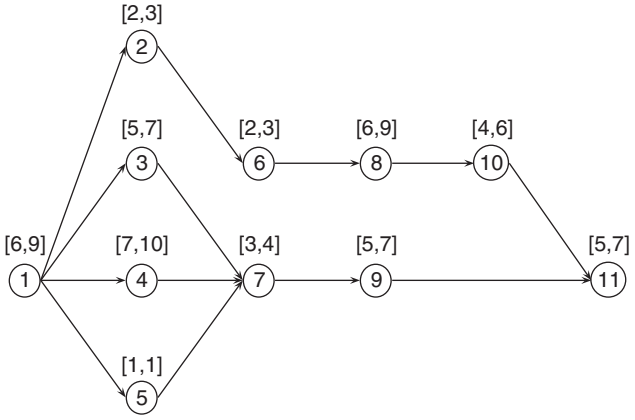

Figure 1 Precedence constraints with interval task times of Jackson instance.

- $OS=\frac{2|\mathcal{A}|}{|V|(|V|-1)}\cdot 100\%$ is the order strength of the precedence constraints represented by graph $G=(V,\mathcal{A})$.

For each instance, 11 tests have been executed with parameter θ from $\{0.0, 0.1, \ldots, 1.0\}$. For each task $j\in V$, task time lower bound a_j has been initiated by the nominal task time value t_j from the corresponding benchmark instance, while upper bound b_j has been set as $1.5t_j$.

Table 2 CPU time (s) for medium size instances

Test name	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Buxey	0.06	0.08	0.14	0.14	0.08	0.11	0.11	0.09	0.08	0.08	0.08
Gunther	0.02	0.02	0.06	0.03	0.03	0.02	0.03	0.02	0.02	0.02	0.02
Heskia	27.01	26.89	1437.7	682.14	209.53	44.58	64.5	5.8	4.63	4.81	2.47
Mitchell	0	0	0	0	0	0	0	0.02	0	0	0
Lutz1	0	0	0	0	0.02	0	0	0	0	0	0
Roszieg	0	0	0	0	0	0	0	0	0	0	0
Sawyer	0.25	0.25	0.38	0.39	0.23	0.25	0.27	0.27	0.31	0.33	0.2

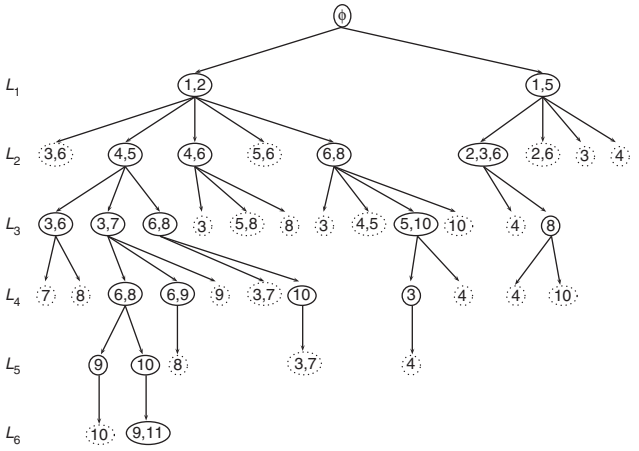

Figure 2 Enumeration tree of the breadth-first search for Jackson instance.

Below, an illustrative example for Jackson instance is given. The graph representing the precedence constraints with interval times for 11 tasks is demonstrated in Figure 1. The following initial data are used: $E=\{\{1,4\},\ \{1,8\},\ \{3,5\},\ \{3,11\},\ \{6,7\},\ \{7,8\},\ \{7,9\},\ \{7,11\},\ \{8,10\},\ \{10,11\}\}$ and $\theta=0.5$. Figure 2 shows the branching process of BFS, where the nodes represent only the load of the workstations constructed at the corresponding level, dotted nodes reflect the last workstations of dominated partial solutions. Here, an optimal solution s contains six workstations $V_1^s=\{1,2\}$, $V_2^s=\{4,5\}$, $V_3^s=\{3,7\}$, $V_4^s=\{6,8\}$, $V_5^s=\{10\}$, $V_6^s=\{9,11\}$. Note also the effect of the dominance rule (Proposition 1). For the example presented, more than half of the partial solutions are dominated at each level starting from the second.

Tables 2 and 3 represent the CPU time for medium and large size instances, respectively. The optimal number of workstations with respect to parameter θ for small, medium, and large size instances are illustrated in Tables 4–6, respectively. The resolution time for small size instances is not presented, since each test was solved in less than 0.15 s.

Analysing the results obtained, we conclude that almost all the tests for medium size instances are solved in less than 0.4 s. Only for the Heskia instance is the CPU time rather great. Nevertheless, it does not exceed 228.19 s on average. The graph of CPU times for this instance with

Table 3 CPU time (s) for large size instances

Test name	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Hahn	0.02	0	0	0	0.02	0.02	0	0.02	0.02	0	0.02
Lutz2	21.36	21.53	14.86	37.45	116.25	3.8	3.91	7.44	7.73	7.69	2.78
Lutz3	0.52	0.53	2.34	17.86	3.16	0.75	0.74	1.58	1.88	2.64	1.30
Tonge	0.27	0.25	0.97	1.06	1.80	3.22	4.42	3.52	3.17	3.38	4.02

Table 4 Workstations number evolution for small size instances

Test name	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Bowman	4	4	4	4	4	5	5	5	5	5	5
Jackson	5	5	5	5	5	6	6	6	6	6	8
Jaeschke	4	4	4	4	5	6	6	6	6	6	7
Mansoor	3	3	3	4	4	5	5	5	5	5	5
Martens	3	3	3	3	3	5	5	5	5	5	5

Table 5 Workstations number evolution for medium size instances

Test name	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Buxey	9	9	9	10	11	11	11	11	11	11	14
Gunther	9	9	9	10	10	12	12	12	12	12	14
Heskia	7	7	7	7	8	8	8	8	8	8	10
Mitchell	6	6	6	6	7	8	8	8	8	8	8
Lutz1	8	8	9	10	10	10	10	10	10	10	12
Roszieg	7	7	7	8	8	8	8	9	9	9	10
Sawyer	9	9	9	10	10	11	11	11	11	11	14

Table 6 Workstations number evolution for large size instances

Test name	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Hahn	8	8	8	8	8	9	9	9	9	9	11
Lutz2	32	32	32	33	35	42	42	43	43	43	49
Lutz3	17	17	18	18	19	21	21	21	21	21	24
Tonge	22	22	22	22	23	24	24	24	24	24	26

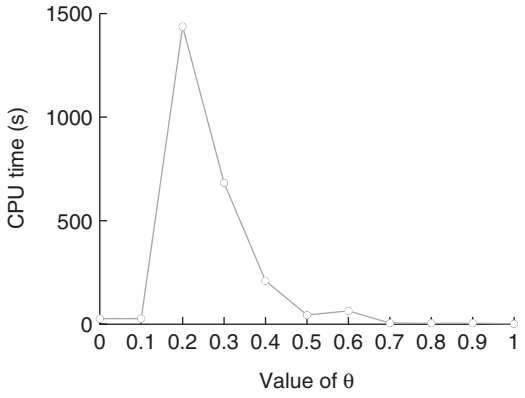

Figure 3 CPU time for the Heskia instance.

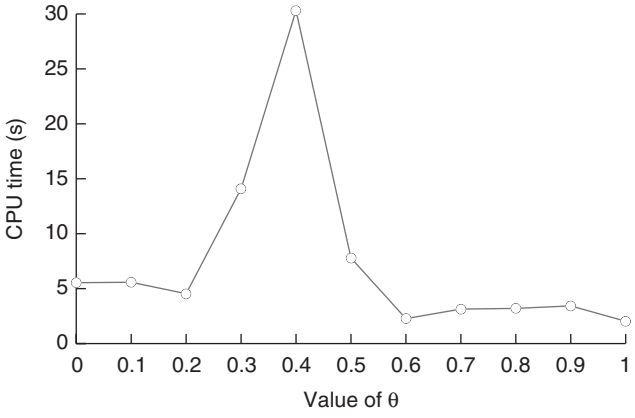

Figure 4 Average CPU time for large size instances.

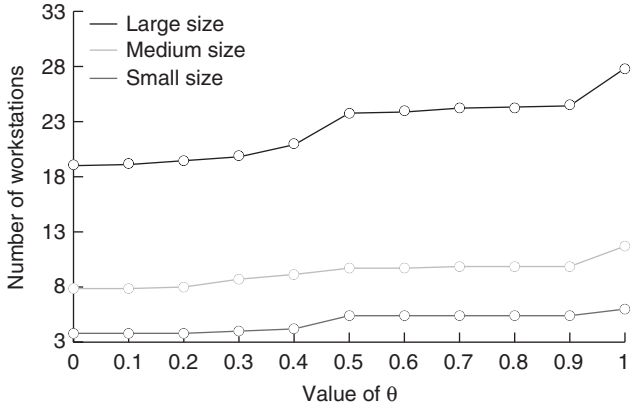

Figure 5 Average number of workstations for three series.

respect to values of θ is shown in Figure 3. When we look at large size instances, the average CPU time of their tests does not exceed 30.31 s. The corresponding CPU time graph is given in Figure 4. The spikes of the CPU times in Figures 3 and 4 for $\theta = 0.2$ and $\theta = 0.4$, respectively, can be attributed to the sudden increase in the size of the enumeration tree. Whereas for two extreme cases, $\theta = 0$ and $\theta = 1$, a different CPU time curve is observed. Indeed, if $\theta = 0$, workstations with relatively greater number of assigned tasks are generated and, as a consequence, this leads to enumeration trees with rather small number of levels. When $\theta = 1$, a moderate number of small-scale nodes are constructed at each level. This is caused by essential decreasing alternatives to assign tasks to workstations. Therefore, the branching process from one level to another one is rapidly implemented.

Figure 5 shows the graph of the average number of workstations for small, medium, and large size instances with respect to θ. To study the effect of θ, an indicator

$$\mathcal{Q} = \text{avg}_{i \in \mathcal{I}_{10}} \frac{w_i - w_{i-1}}{w_{i-1}}$$

was introduced. Here, $\mathcal{I}_n = \{1, 2, \ldots, n\}$, w_i is the number of workstations in the optimal solution found for $\theta = i/10$. Indicator $\mathcal{Q}$ seeks the average relative augmentation of the number of workstations with respect to θ. Concerning $\mathcal{Q}$, we conclude that there is no significant difference in its progression ($\mathcal{Q}$ is respectively equal to 0.04, 0.04, and 0.03), in spite of the fact that the graphs corresponding to small and medium size instances are smoother than for the larger-sized ones. This illustrates the robustness of the model proposed with regard to the parameter θ and size of problem instances.

5. Conclusions

In this paper, a robust model for SALBP-1 with interval-given task times was proposed. To solve this model, a branch-and-bound procedure was developed and evaluated on benchmark instances.

The model presented includes a workstation-oriented real parameter $\theta \in [0, 1]$. It offers to designers the possibility to find a compromise between the optimistic ($\theta = 0$) and pessimistic ($\theta = 1$) cases. In practice, this value can reflect the risk level non-accepted by a decision maker.

For the case of random micro-stoppages at workstations, as for the assembly line producing car door locks, the parameter θ can be viewed as the probability of forecasted stoppages of a workstation per cycle. In practice, testing several values of θ is recommended.

Note that the approach proposed could be used for a decision support at the preliminary stage of assembly line design. It can also be useful for decision makers when they have to choose investment strategies.

A future research perspective is to seek for tight lower and upper bounds for branch-and-bound techniques. Another way is to extend the robust model proposed to assembly lines with a more complex structure such as *U*-shaped or mixed-model assembly lines.

Note that the introduced robust counterpart of SALBP-1 can be easily extended to the more generalized case where

the value of θ depends on workstation. This generalization can be successfully applied in practice under the condition that a decision maker can distinguish precisely a workstation with different levels of variability of task times.

Acknowledgements—We thank Chris Yukna for his help in editing the English language used in this paper.

References

Ağpak K and Gökçen H (2007). A chance-constrained approach to stochastic line balancing problem. *European Journal of Operational Research* **180**(3): 1098–1115.

Aissi H, Bazgan C and Vanderpooten D (2009). Min-max and min-max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research* **197**(2): 427–438.

Alem D and Morabito R (2011). Production planning in furniture settings via robust optimization. *Computers & Operations Research* **39**(2): 139–150.

Battini D, Faccio M, Ferrari E, Persona A and Sgarbossa F (2007). Design configuration for a mixed-model assembly system in case of low product demand. *The International Journal of Advanced Manufacturing Technology* **34**(1–2): 188–200.

Baybars I (1986). A survey of exact algorithms for the simple assembly line balancing problem. *Management Science* **32**(8): 909–932.

Baykasoğlu A and Özbakir L (2007). Stochastic U-line balancing using genetic algorithms. *The International Journal of Advanced Manufacturing Technology* **32**(1–2): 139–147.

Bertsimas D and Sim M (2003). Robust discrete optimization and network flows. *Mathematical Programming* **98**(1–3): 49–71.

Bertsimas D and Thiele A (2006). A robust optimization approach to inventory theory. *Operations Research* **54**(1): 150–168.

Chiang W-C and Urban T (2006). The stochastic U-line balancing problem: A heuristic procedure. *European Journal of Operational Research* **175**(3): 1767–1781.

Dolgui A and Kovalev S (2012). Min–max and min–max (relative) regret approaches to representatives selection problem. *4OR:A Quarterly Journal of Operations Research* **10**(2): 181–192.

Dolgui A and Proth J-M (2010). *Supply Chain Engineering: Useful Methods and Techniques*. Springer-Verlag: London.

Erel E, Sabuncuoglu I and Sekerci H (2005). Stochastic assembly line balancing using beam search. *International Journal of Production Research* **43**(7): 1411–1426.

Gabrel V and Murat C (2010). Robustness and duality in linear programming. *Journal of the Operational Research Society* **61**(8): 1288–1296.

Gamberini R, Gebennini E, Grassi A and Regattieri A (2009). A multiple single-pass heuristic algorithm solving the stochastic assembly line rebalancing problem. *International Journal of Production Research* **47**(8): 2141–2164.

Gen M, Tsujimura Y and Li Y (1996). Fuzzy assembly line balancing using genetic algorithms. *Computers & Industrial Engineering* **31**(3–4): 631–634.

Hazır O and Dolgui A (2011). Simple assembly line balancing under uncertainty: A robust approach. In: *Proceedings of the International Conference on Industrial Engineering and Systems Management (IESM '2011)*, 25–27 May 2011; Metz, France, pp 87–92.

Hazır O, Erel E and Günalay Y (2011). Robust optimization models for the discrete time/cost trade-off problem. *International Journal of Production Economics* **130**(1): 87–95.

Hop N (2006). A heuristic solution for fuzzy mixed-model line balancing problem. *European Journal of Operational Research* **168**(3): 798–810.

Jackson J (1956). A computing procedure for a line balancing problem. *Management Science* **2**(3): 261–271.

Mausser HE and Laguna M (1999). Minimising the maximum relative regret for linear programmes with interval objective function coefficients. *Journal of the Operational Research Society* **50**(10): 1063–1070.

Moon Y and Yao T (2011). A robust mean absolute deviation model for portfolio optimization. *Computers & Operations Research* **38**(9): 1251–1258.

Özcan U (2010). Balancing stochastic two-sided assembly lines: A chance-constrained, piecewise-linear, mixed integer program and a simulated annealing algorithm. *European Journal of Operational Research* **205**(1): 81–97.

Scholl A (1999). *Balancing and Sequencing of Assembly Lines*. 2nd edn. Physica-Verlag: Heidelberg.

Tsujimura Y, Gen M and Kubota E (1995). Solving fuzzy assembly-line balancing problem with genetic algorithms. *Computers & Industrial Engineering* **29**(1–4): 543–547.

Urban T and Chiang W-C (2006). An optimal piecewise-linear program for the U-line balancing problem with stochastic task times. *European Journal of Operational Research* **168**(3): 771–782.

Received June 2011;
accepted October 2012 after two revisions