Int. J. Production Economics 141 (2013) 552–560

Contents lists available at SciVerse ScienceDirect

Int. J. Production Economics

journal homepage: www.elsevier.com/locate/ijpe

Hybrid heuristics for planning job rotation schedules in assembly lines with heterogeneous workers

Mayron César O. Moreira, Alysson M. Costa *

Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Brazil

ARTICLE INFO

Article history:
Received 22 December 2011
Accepted 14 September 2012
Available online 25 September 2012

Keywords:
Assembly lines
Job rotation
Hybrid algorithm

ABSTRACT

We investigate the problem of balancing assembly lines with heterogeneous workers while considering job rotation schedules. This problem typically occurs in assembly lines in sheltered work centers for disabled. We propose a hybrid algorithm that uses a Mixed Integer Programming (MIP) to select appropriate schedules from a pool of heuristically constructed solutions. A local search based on MIP neighborhoods is used as a post-optimization method. Our results show that this approach is fast, flexible and accurate when compared with current available methods.

1. Introduction

We deal with the problem of balancing assembly lines with heterogeneous workers while considering job rotation schedules. This problem has been introduced in the literature by Costa and Miralles (2009), based on an application found in the context of sheltered work centers for the disabled, and is described in the following.

Given a set of ordered workstations, a set of workers and a partially ordered set of tasks, a schedule is defined as an assignment of workers and tasks to the ordered workstations which respects the tasks partial ordering. The load of a workstation is the time the worker assigned to that workstation needs to execute the tasks assigned to the workstation while the cycle time of a schedule is the maximum load among all workstations. The planning of job rotation schedules in assembly lines with heterogeneous workers consists in finding a set of schedules, one for each subperiod of a complete rotation period. The problem has two main goals: the minimization of the sum of cycle times and the maximization of the number of different tasks each worker executes in a complete rotation period, considering all the subperiod schedules. The first metric is clearly associated with the line efficiency while the second one relates to ergonomic and training objectives.

Costa and Miralles (2009) have modeled the first goal as constraints and used the second goal as objective function. The rationale used was to maximize the job rotation gains while guaranteeing a minimum level of efficiency. This level could be adjusted according to the external demand of products: in periods with high demand, highly productive schedules would be preferred and in periods with low demand the efficiency constraints could be softened in order to obtain schedules with high job rotation characteristics. The authors have used the proposed model in a decomposition approach that solves mixed-integer programming (MIP) subproblems in order to decide the schedules for each single period. This approach presents good results but is not scalable, since the MIP subproblems are already of high complexity.

In this paper, we propose a new solution method for this problem. The method is composed of three main steps. First, a pool of single-period schedules is obtained with different heuristic methodologies (which are borrowed from the literature or proposed in this paper). A mixed-integer problem is then used to select appropriate schedules from the pool and compose a complete rotation planning. Finally, a post-optimization routine based on large-scale MIP neighborhoods is used to improve the solution.

The proposed approach is evaluated with a rigorous computational study in which we compare versions of the same algorithm with different pool sizes and single-period heuristics. With this study, the most important characteristics of the methodology are highlighted. These include the ability to generate a diverse set of single-schedule solutions and the power of appropriately combining these solutions in complete rotation schedules. This differs from the available method in the literature in its capacity of considering the full schedule period. Moreover, the proposed method is scalable to much larger instances, as shown by the computational experiments.

The remainder of this paper is organized as follows. In the following section, we review the related literature. A formal definition of the problem and the relevant mathematical models are presented in Section 3. In Section 4, we detail the heuristic

* Corresponding author.
E-mail addresses: mayron@icmc.usp.br (M.C.O. Moreira), alysson@icmc.usp.br (A.M. Costa).

0925-5273/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ijpe.2012.09.011

methods used to solve the single-period problems and in Section 5 the proposed approach is described. Section 6 discusses the obtained results and, in Section 7, final comments and conclusions are presented.

2. Literature review

Assembly lines are flow oriented industrial production systems in which the tasks needed to assemble a product are executed sequentially, by different workers or machines. The *simple assembly line balancing problem* (SALBP) is the fundamental associated optimization problem which main assumption is the fact that workers are homogeneous with respect to task execution times. The rich literature on the SALBP and some closely related problems contains a number of both classical and recent review and research papers (Salveson, 1955; Tonge, 1961; Baybars, 1986; Ghosh and Gagnon, 1989; Scholl, 1999; Scholl and Becker, 2006; Boysen et al., 2007, 2008, 2012; Boysen and Fliedner, 2008; Scholl et al., 2010).

An extension of the SALBP in which task execution times are worker-dependent has been proposed in the literature by Miralles et al. (2007) and named the *assembly line worker assignment and balancing problem* (ALWABP). In this extension, each task has different execution times depending on the selected worker. A branch-and-bound algorithm has been developed for the ALWABP (Miralles et al., 2008) but due to the problem's complexity, most of the research has focused on heuristic solution methodologies. These include simple heuristics such as greedy algorithms (Moreira et al., 2012), classical metaheuristics such as tabu search (Moreira and Costa, 2009), Beam search (Blum and Miralles, 2011) and Genetic Algorithms (Moreira et al., 2012) and new metaheuristics such as clustering search (Chaves et al., 2009).

We are interested in studying the ALWABP when job rotation is taken into consideration. The importance of job rotation in assembly lines has been largely discussed in the literature. Indeed, many authors relates job rotation planning to ergonomic factors, directly linking good schedules to the well being of assembly line workers. Some authors have addressed the problem of minimizing the number of tasks with potential for injury that are executed by a single worker (Carnahan et al., 2000; Tharmmaphornphilas and Norman, 2007; Aryanezhad et al., 2008) while others have concentrated on minimizing the amount of strenuous jobs executed by a worker (Seçkiner and Kurt, 2007, 2008). Sato and Coury (2009), in turn, have studied the effects of the job rotation concerning discomfort, disorders and sick leave of workers, while Azizi et al. (2010) studied the influence of job rotation strategies in employee's boredom.

All the mentioned studies kept the SALBP assumption that task execution times were worker-independent. To our knowledge, Costa and Miralles (2009) were the only authors to consider job rotation planning in the ALWABP. These authors analyzed assembly lines in sheltered work centers for the disabled. In this context, job rotation might not only bring the usual benefits such as increased motivation and less injuries but also be associated with therapeutical treatments.

The strategy proposed by Costa and Miralles (2009) relied on the decomposition of the original problem in a planning problem in which the schedules for the periods were obtained sequentially. This allowed the obtention of good quality solutions for small instances but had the limitation of not being scalable, since it depended on the resolution of complex mixed-integer problems.

3. Formal definitions and mathematical formulations

We first present the ALWABP model, associated with the problem of scheduling a single period of the job rotation plan. Let S be a set of ordered workstations, W be a set of workers and $(N, \leq)$ be a partially ordered set of tasks. For tasks $i,j \in N$, we write $i<j$ when $i \leq j$ and $i \neq j$ and $i \lessdot j$ when i immediately precedes j. Each worker $w \in W \backslash W_i$ executes a task $i \in N$ in time p_{wi}, where W_i is the set of workers that are unable to execute task i. With this notation, a mathematical model can be written for the ALWABP as

$$\text{Min } C \tag{1}$$

subject to

$$\sum_{s \in S} \sum_{w \in W \backslash W_i} x_{swi} = 1 \quad \forall i \in N, \tag{2}$$

$$\sum_{i \in N} \sum_{w \in W \backslash W_i} x_{swi} \geq 1 \quad \forall s \in S, \tag{3}$$

$$\sum_{s \in S} y_{sw} = 1 \quad \forall w \in W, \tag{4}$$

$$\sum_{w \in W} y_{sw} = 1 \quad \forall s \in S, \tag{5}$$

$$\sum_{s \in S} \sum_{w \in W \backslash W_i} s \cdot x_{swi} \leq \sum_{s \in S} \sum_{w \in W \backslash W_j} s \cdot x_{swj} \quad \forall i, j \in N,\ i \lessdot j, \tag{6}$$

$$\sum_{i \in N} \sum_{w \in W \backslash W_i} p_{wi} \cdot x_{swi} \leq C \quad \forall s \in S, \tag{7}$$

$$\sum_{i \in N | w \notin W_i} x_{swi} \leq |N| y_{sw} \quad \forall s \in S,\ \forall w \in W, \tag{8}$$

$$y_{sw} \in \{0,1\} \quad \forall s \in S,\ \forall w \in W, \tag{9}$$

$$x_{swi} \in \{0,1\} \quad \forall s \in S,\ \forall w \in W \backslash W_i,\ \forall i \in N. \tag{10}$$

where the variables are:

x_{swi} binary variable; equals 1 only if task i is assigned to worker w at workstation s,
y_{sw} binary variable; equals 1 only if the worker w is assigned to workstation s,
C cycle time.

Model (1)–(10) minimizes the cycle time. Constraints (2) indicate that each task must be executed, while (3) guarantee that each workstation will have at least one task. Constraints (4) and (5) ensure that each worker is assigned to a single workstation and each workstation receives a single worker, respectively. Precedence constraints between tasks are given by constraints (6). Constraints (7) and (8) allow a worker to execute more than one task provided that the line cycle time is respected.

Costa and Miralles (2009) extended the model presented above to the planning of job rotation schedules, where a set of $\overline{T}$ subperiods, $T=[1 \ldots \overline{T}]$, is given and schedules are obtained for each subperiod $t \in T$. The new model variables are:

x_{swit} binary variable; equals 1 only if task i is assigned to worker w at workstation s at period t,
y_{swt} binary variable; equals 1 only if the worker w is assigned to workstation s at period t,
z_{wi} binary variable; equals 1 only if the worker w executes task i in at least one subperiod.

where the new binary variables z_{wi} indicate if a worker w ever executes a task i. With the additional notation:

t index for rotation subperiods,
C_t cycle time of subperiod t,
$\underline{C}$ maximum mean allowed cycle time.

The job rotation model for the ALWABP can be defined as follows:

$$\text{Max } z = \sum_{w \in W} \sum_{i \in N} z_{wi} \tag{11}$$

subject to

$$\sum_{s \in S} \sum_{w \in W \setminus W_i} x_{swit} = 1 \quad \forall i \in N,\ \forall t \in T, \tag{12}$$

$$\sum_{i \in N} \sum_{w \in W \setminus W_i} x_{swit} \geq 1 \quad \forall s \in S,\ \forall t \in T, \tag{13}$$

$$\sum_{s \in S} y_{swt} = 1 \quad \forall w \in W,\ \forall t \in T, \tag{14}$$

$$\sum_{w \in W} y_{swt} = 1 \quad \forall s \in S,\ \forall t \in T, \tag{15}$$

$$\sum_{s \in S} \sum_{w \in W \setminus W_i} s \cdot x_{swit} \leq \sum_{s \in S} \sum_{w \in W \setminus W_j} s \cdot x_{swjt} \quad \forall i,j \in N, i.j,\ \forall t \in T, \tag{16}$$

$$\sum_{i \in N | w \in W \setminus W_i} p_{wi} \cdot x_{swit} \leq C_t \quad \forall s \in S,\ \forall t \in T, \tag{17}$$

$$\sum_{i \in N | w \notin W_i} x_{swit} \leq |N| y_{swt} \quad \forall s \in S,\ \forall w \in W,\ \forall t \in T, \tag{18}$$

$$\sum_{t=1}^{|T|} C_t \leq |T| \underline{C}, \tag{19}$$

$$z_{wi} \leq \sum_{t=1}^{|T|} \sum_{s \in S} x_{swit} \quad \forall i \in N,\ \forall w \in W \setminus W_i, \tag{20}$$

$$y_{swt} \in \{0,1\} \quad \forall s \in S,\ \forall w \in W,\ \forall t \in T, \tag{21}$$

$$x_{swit} \in \{0,1\} \quad \forall s \in S,\ \forall i \in N,\ \forall w \in W \setminus W_i,\ t \in T, \tag{22}$$

$$z_{wi} \in \{0,1\} \quad \forall w \in W,\ \forall i \in N. \tag{23}$$

The objective function (11) maximizes the number of different tasks executed by each worker. Constraints (12)–(18) enforce that the original problem constraints are respected for each subperiod. To guarantee that the mean cycle time of the rotation period does not exceed the desired cycle time $\underline{C}$, constraints (19) are required. Finally, constraints (20) compute the execution (or not) of task i by worker w.

4. Heuristic methods for the ALWABP

Good quality schedules for single periods are key to the obtention of good quality job rotation plans. In this section, we describe the approaches that are used to obtain such schedules. Four methods are briefly presented: the constructive heuristics and the random-key genetic algorithm proposed by Moreira et al. (2012) and two new methods, an extension of the minimalist tabu search of Moreira and Costa (2009) and a GRASP metaheuristic. These two latter methods are described in a slightly more detailed manner. The rationale for using these specific methods was the search for solutions with different characteristics. We use both populational and neighborhood-based methods and both sophisticated and simple meta-heuristics and heuristics.

4.1. Constructive heuristics (CH)

Moreira et al. (2012) have developed simple constructive heuristics for the ALWABP based on the constructive heuristics proposed by Scholl and Voß (1996) for the SALBP. The idea is to sequentially add tasks to the workstations, respecting the precedence constraints and using an order given by a heuristic priority rule. Two main differences are present in the new algorithm: first of all, most of task priority rules proposed in (Scholl and Voß 1996) are dependent on task execution times, which is not suitable for ALWABP, since task time relies upon the chosen worker. Second, there is no strategy to select the worker to be assigned to each workstation.

The authors have proposed 16 heuristic rules adapted to the ALWABP case for prioritizing which tasks should be considered first, and three similar criteria for assigning workers. The method runs very fast with any of the criteria combination (fractions of seconds are needed to run the largest instances) and obtains reasonable results (average gaps of 18% are obtained with the best criteria).

4.2. Hybrid genetic algorithm (HGA)

The same authors have used the simple heuristic procedures mentioned above as decoders in a biased random-key genetic algorithm (Moreira et al., 2012). The fitness of each individual is given by the value returned by the constructive heuristics. Each individual defines the values of the priority criterion to be used when evaluating the fitness, i.e., the chromosome is a matrix of dimensions $|N| \times |W|$ in which each cell corresponds to the priority value of a given pair task $\times$ worker. Elitism is enforced by maintaining a set of *elite solutions*. At the selection phase of the algorithm, one of the mating parents always comes from this elite set. The algorithm also uses a local search heuristic (based on insertion and swap moves) that is applied to each new individual. Another feature is the *immigration* phase, which consists in the insertion of randomly generated individuals at each iteration of the algorithm. With this algorithm, the authors presented some of the best results available in the literature.

4.3. Tabu search (TS)

We extend the Tabu Search algorithm presented in Moreira and Costa (2009) for the ALWABP with the introduction of intensification and diversification procedures. The original algorithm is based on three types of move:

- Insertion: moves a task from one workstation to another;
- Tasks swap: swaps two tasks between two workstations;
- Workers swap: swaps two workers between two workstations.

Infeasible solutions are allowed but penalized. We consider two kinds of infeasibilities: (a) associated with violated precedence constraints (I_p) and (b) associated with the assignment of tasks to workers unable to execute them (I_w). The objective function is given by $f = C + \alpha_w \cdot I_w + \alpha_p \cdot I_p$, with $\alpha_w, \alpha_p \in \mathbb{R}$, and these infeasibilities are calculated as follows:

$$I_w = \sum_{i \in N} \sum_{w \in W_i} \sum_{s \in S} x_{swi}, \tag{24}$$

$$I_p = \max\left\{0, \sum_{s=1}^{|S|-1} \sum_{s'=s+1}^{|S|} \sum_{i,j \in N | i<j} x_{swi} \cdot x_{s'wj} \cdot (s'-s)\right\}. \tag{25}$$

We use dynamic penalty factors, as proposed by Gendreau et al. (1994). In this scheme, a penalty factor is increased after

each iteration in which the solution was infeasible with respect to the associated constraint, and decreased otherwise.

Algorithm 1. Tabu search algorithm for the ALWABP.

1: **input**: **s**: initial solution generated randomly or by a heuristic;
2: $\overline{s} \leftarrow s$,
3: Initialize penalty factors;
4: **while** stopping criterion not reached **do**
5: let n be the best neighbor in $N(s)$;
6: **if** n is tabu **then**
7: **if** n is feasible and $f(n) < f(\overline{s})$ **then**
8: Apply LS1, LS2 and LS3 on n;
9: $\overline{s} \leftarrow n$, $s \leftarrow n$;
10: **else**
11: let n be the best non-tabu neighbor in $N(s)$;
12: **end if**
13: **end if**
14: $s \leftarrow n$; update penalty factors;
15: apply intensification (IP1 and IP2) or diversification (DP1 and DP2) if criterion is reached;
16: apply restart if criterion is reached;
17: **end while**
18: **output**: solution $\overline{s}$.

Algorithm 1 gives an overview of the method. An aspiration criterion is used to accept tabu solutions which are feasible and are better than the best solution found so far (lines 6–9). If no such solution is found, the method finds the best non-tabu solution in the neighborhood of the current solution (line 11). The appropriate penalty weights are doubled (halved) every time the best neighbor is an infeasible (a feasible) solution with respect to the associated constraint (line 14).

This method is an extension of the method presented by Moreira and Costa (2009). In this new version, we added several features to the algorithm, which are described in the following. Three local search procedures precede the execution of the aspiration criteria: LS1, LS2, and LS3. LS1 effects a local search with insertion movements that only considers tasks belonging to critical workstations (without idle time), swap tasks, swap workers and enchained swap tasks (sequence of two swap movements). LS2 fixes the workers at their current stations and makes a call to the constructive heuristic presented in Section 4.1. LS3 applies worker swap movements and also calls the constructive heuristic.

Two intensification (IP1 and IP2) or two diversification (DP1 and DP2) movements are used (line 15). The algorithm alternates between them every 2000 iterations. IP1 (DP1) makes a call to LS1 fixing in the solution the most (less) common tasks $\times$ workstations pairs. IP2 (DP2) makes a call to LS2 after fixing in the solution the most (less) common workers $\times$ workstations pairs.

A restart is made after 5000 iterations (line 16). A 10,000 iterations limit is used as stopping criterion.

4.4. GRASP

The *greedy randomized adaptative search procedure* (GRASP) is a multi-start metaheuristic proposed by Feo and Resende (1989). It consists of two phases: initial solution construction and solution improvement via local search.

We propose a GRASP for the ALWABP which strongly relies on the constructive heuristics presented in Section 4.1. The main idea is to obtain a different constructive solution at each GRASP iteration. A pseudo-code is presented in Algorithm 2.

Algorithm 2. GRASP algorithm for the ALWABP.

1: **input**: P: set of priority rule to be used;
2: $\overline{s} \leftarrow$ best known solution (empty if no solution is known);
3: **for** $p \in P$ **do**
4: **while** stopping criteria not reached **do**
5: Run randomized constructive heuristic with rule p, yielding solution n;
6: Apply LS1 on n;
7: **if** $f(n) < f(\overline{s})$ **then**
8: $\overline{s} \leftarrow n$;
9: **end if**
10: **end while**
11: **end for**
12: **output**: solution $\overline{s}$.

In this implementation we used P containing the three rules that presented the best results in Moreira et al. (2012), all of them based on tasks positional weights. For each rule, the stopping criterion was set in terms of numbers of iterations (1000 iterations for each rule).

At each step of the construction (line 5), a restricted candidate list is formed by the first $|N|/5$ tasks presenting the best values of the used priority rule. Therefore, the considered priority rule is computed for all available tasks at a given moment and the tasks presenting the best criterion values are included in the list. Then, the task to be inserted is chosen randomly from this list. The initial obtained solution is improved by means of a local search procedure similar to LS1, which was described in the last subsection. Finally, the best solution is updated in line 8.

5. Hybrid algorithm

A recent trend on optimization is the hybridization of solution methods, which might include metaheuristics, mathematical programming and human iterative procedures. Blum et al. (2011) argue that the focus on research has notably shifted from an algorithm-oriented point of view to a problem-oriented point of view, where the goal of researchers is no longer to promote a certain metaheuristic but rather to efficiently solve a problem. This claim is supported by the large amount of research that has been recently devoted to hybrid methods (El-Abd and Kamel, 2005; Puchinger and Raidl, 2005; Maniezzo et al., 2009; Günther et al., 2010).

We propose a hybrid algorithm for the job rotation scheduling problem with heterogeneous workers (HAJR) that follows these ideas and makes use of the heuristic and metaheuristic procedures presented in the previous section and of additional mathematical programming models to obtain good rotation schedules. The HAJR first creates a *pool J* of *warehouse solutions* (Rochat and Taillard, 1995) for the ALWABP, with the application of the four heuristic methods cited previously. Then, an integer linear programming model is used to select $\overline{T}$ of these schedules, while respecting the coupling constraints (19), and maximizing the objective function (11). Finally, two local search approaches based on large MIP-neighborhoods are executed in order to improve the objective function.

In the following, we give a description of the implemented HAJR. Section 5.1 explains the construction and management of the warehouse solutions. Section 5.2 explains the selection of the $\overline{T}$ schedules that will belong to the initial job rotation solution, J', and Section 5.3 describes how this initial solution is improved. A complete view on the algorithm is finally presented.

5.1. Management of feasible solutions

The pool of feasible solutions J is created with the execution of the heuristic methods described in Section 4. The main idea is to generate a set of diverse, good-quality solutions. It is important to limit the size of the pool, P_s, since it affects the time needed in the next phase of the method. Therefore, the set includes:

1. s_{HGA} feasible solutions from the last generation of the HGA, $s_{HGA} \leq 100$;
2. Sixteen feasible solutions obtained with the best configuration of the constructive heuristics (see Moreira et al., 2012);
3. $P_s - s_{HGA} - 16$ feasible solutions obtained with the TS and with the GRASP, equally distributed.

5.2. Solutions selection

We propose a mixed-integer programming model to select the $\overline{T}$ solutions that will constitute J', the subperiod schedules in the job rotation, from the set J of warehouse solutions. The idea of this model is to select schedules that will respect the coupling constraints concerning the minimum level of efficiency while maximizing the diversity of tasks criterion. The complete formulation is presented in the following:

$$\text{Max } z = \sum_{w \in W} \sum_{i \in N} z_{wi} \tag{26}$$

subject to

$$\sum_{k=1}^{|J|} C_k \cdot \beta_k \leq |T| \underline{C}, \tag{27}$$

$$z_{wi} \leq \sum_{k=1}^{|J|} a_{kwi} \cdot \beta_k \quad \forall w \in W,\ \forall i \in N, \tag{28}$$

$$\sum_{k=1}^{|J|} \beta_k = \overline{T}, \tag{29}$$

$$z_{wi} \in \{0,1\} \quad \forall w \in W,\ \forall i \in N, \tag{30}$$

$$\beta_k \in \{0,1\} \quad k = 1, \ldots, |J|, \tag{31}$$

where,

- C_k cycle time of solution $k \in J$,
- a_{kwi} binary parameter; equals to 1 only if task i is assigned to worker w in solution $k \in J$,
- β_k binary variable; equals to 1 only if solution $k \in J$ is chosen as subperiod schedule,
- z_{wi} binary variable; equals to 1 only if worker w executes task i in at least one subperiod.

The objective of the model (26) is the maximization of different tasks executed by workers in a complete job rotation period. Constraints (27) enforce the coupling constraints (19). Constraints (28) guarantee that variables z_{wi} represent the execution of task i by worker w. Finally, $\overline{T}$ solutions must be chosen according to constraints (29), one for each job rotation subperiod.

5.3. Local search

After an initial solution is found for the job rotation problem, two local search procedures are used to improve its quality. These local search procedures rely on large MIP-based neighborhoods and are described in the following.

Let $\overline{x}_{swit}$ and $\overline{y}_{sw}$ denote the variables assuming one in the current solution, given by the schedules in J'. The first procedure, named *local search for job rotation* 1 (LSJR1), receives as input a solution J' and allows minor modifications in its structure. Indeed, the local search allows tasks to remain in their current stations or to be assigned to immediately neighboring stations. This can be easily done by fixing the appropriate variables at zero and solving a new version of model (11)–(23), as shown below.

$$\text{Max } z = \sum_{w \in W} \sum_{i \in N} z_{wi} \tag{32}$$

subject to

(12)–(23)

$$x_{swit} \leq \overline{x}_{swit} + \sum_{w' \in W \setminus W_i} \overline{x}_{s-1w'it} + \sum_{w' \in W \setminus W_i} \overline{x}_{s+1w'it}$$
$$\forall s \in S,\ \forall w \in W \setminus W_i,\ \forall i \in N, \forall t \in T, \tag{33}$$

$$y_{swt} = \overline{y}_{swt}$$
$$\forall s \in S, \forall w \in W, \forall t \in T. \tag{34}$$

Constraints (33) enforces variables x_{swit} to be set at zero if task i is not assigned to station s nor to any of its neighbors ($s-1$ and $s+1$) in the original solution J'. Note that variables x_{s-1wit} (x_{s+1wit}) must be ignored in the constraints if station s is the first (last) station in the line. Constraints (34) ensure that the workers are kept at their current positions.

The second procedure, named *local search for job rotation 2* (LSJR2) receives as input a feasible job rotation solution and a subperiod $\widehat{t}$. It then reduces the search space by fixing the solution at all subperiods with the exception of subperiod $\widehat{t}$ and solves the resulting model, shown below

$$\text{Max } z = \sum_{w \in W} \sum_{i \in N} z_{wi} \tag{35}$$

subject to

(12)–(23)

$$x_{swit} = \overline{x}_{swit} \quad \forall s \in S, \forall w \in W \setminus W_i, \forall i \in N, \forall t \in T \setminus \widehat{\{t\}}, \tag{36}$$

$$y_{swt} = \overline{y}_{swt} \quad \forall s \in S, \forall w \in W, \forall t \in T, \tag{37}$$

In this case, constraints (36) ensure that all tasks from a subperiod different from $\widehat{t}$ must remain in their original workstations. Finally, to guarantee the original allocations of workers in the $\overline{T}$ subperiods, constraints (37) are required.

5.4. Summary of the algorithm

A complete view of the HAJR is shown in Algorithm 3. The algorithm is a sequential application of the several construction and improvement methods described so far. First, the four heuristic methods are applied (lines 1–5): the final pool will include 16 solutions obtained with the constructive heuristics (line 2) and all feasible solutions of the last generation of the genetic algorithm with population size equal to 100 (line 3). The remaining solutions in the pool are obtained from the Tabu Search and the GRASP metaheuristics. In order to keep a limited size of P_s solutions, a maximum of $(P_s - |J|)/2$ solutions are added by each of these methods, where $|J|$ is the number of solutions added so far (lines 4–5).

After the pool is obtained, model (26)–(31) is used to select $|T|$ single schedule solutions and form the first complete solution (line 6). This solution is then improved with the MIP-based local search procedures of Sections 5.3. First, model (32)–(34) is used to search for better solutions which keep the tasks at the same or at neighboring stations (line 7). Finally, the MIP model (35)–(37) is used to look for better solutions while keeping almost all

variables fixed and releasing the values of the variables of a period, for all periods, one at a time.

Algorithm 3. Hybrid Algorithm for the Job Rotation (HAJR).

1: Construct pool of solutions J:
2: Add to J the 16 solutions obtained with the Constructive Heuristics of Section 4.1;
3: Add to J all feasible solutions found in the last generation of the biased random-key genetic algorithm of Section 4.2;
4: Add to J at most $(P_s-|J|)/2$ feasible solutions found with the tabu search algorithm of Section 4.3;
5: Add to J $(P_s-|J|)$ feasible solutions found with the GRASP method of Section 4.4;
6: Run model (26)–(31) using J to select $|T|$ solutions and add them to J';
7: Run model (32)–(34) on J': $J' \leftarrow LSJR1(J')$;
8: Run model (35)–(37) for each schedule period:
9: **for** $\widehat{t}=1\ldots\overline{T}$ **do**
10: $J' \leftarrow LSJR2(J',\widehat{t})$;
11: **end for**
12: **output**: set of selected schedules, J'.

6. Computational experiments

In this section, we describe a number of computational experiments and analyze their results to examine the performance of the proposed HAJR. Section 6.1 presents the test problems and methodology used while Section 6.2 shows the numerical results.

6.1. Test problems and methodology

In order to test the proposed methodology, we have used a set of instances available in the literature (Chaves et al., 2007). This data set is adapted from SALBP instances and grouped in four families: Roszieg, Heskia, Tonge and Wee-Mag, each one containing 80 instances. These data are generated in such a way that each family of problems contains 10 instances for each combination of three experimental factors: number of workers, the variability of task execution times, and the number of infeasible task-worker pairs. In each situation, each one of these factors assume a low or a high level.

As described in Chaves et al. (2007), for instances with low variability, the task execution times are obtained with an uniform distribution in the interval $[1,t_i]$, where t_i is the task execution time of the original SALBP instance. In case of high variability, the interval used is $[1,3t_i]$. The low and the high levels associated with the number of infeasible task-worker pairs are 10% and 20%, respectively. Table 1 lists the main characteristics of each family of instances.

We compare our results with the *decomposition algorithm* (DA) proposed by Costa and Miralles (2009). As these authors, we consider a parameter R indicating the allowed percentage cycle time increase. We use values of R as 5%, 10%, 25% and 50%.

Table 1
Instance characteristics.

Family	$\|N\|$	$\|W\|$	Order strength
Roszieg	25	4 (groups 1–4) or 6 (groups 5–8)	71.67
Heskia	28	4 (groups 1–4) or 7 (groups 5–8)	22.49
Tonge	70	10 (groups 1–4) or 17 (groups 5–8)	59.42
Wee-Mag	75	11 (groups 1–4) or 19 (groups 5–8)	22.67

Concerning the number of subperiods, we analyzed $|T|=2$, 4 and 7. The combination of the different values of R and $|T|$ yield 12 scenarios.

For the Roszieg and Heskia families, we use the optimal single period cycle time as the base cycle time value $\underline{C}$. For instances of Tonge and Wee-Mag families, $\underline{C}$ is determined as the best value of cycle time present in the pool of solutions.

The algorithms were implemented in C++ and all numerical tests were carried out on a Pentium Core 2 Duo with 2.2 GHz and 3.0 Gb RAM. For the resolution of linear models, we used CPLEX 12.1, with a time limit of 3600 s to solve each model. Each instance was executed five times and the same pool of initial solutions was used to solve the 12 scenarios.

6.2. Numerical results

We first compare the results of our proposed algorithm with those obtained by the *decomposition algorithm* of Costa and Miralles (2009), which was run on a Pentium machine with 2.33 GHz and 4.0 Gb RAM. Their algorithm heuristically decomposes the problem by period. A version of model (1)–(10) is then run for each period. Once a period is solved, its solution is kept fixed. The objective function at each run is modified to maximize the number of new different tasks chosen for that period and a constraint limiting the cycle time in each period is included.

We use the following metrics:

- Gap(i): average deviation of a method to the best known solution for instance i

$$\text{Gap}(i)(\%)=\frac{z_i^{best}-\overline{z}_i}{\overline{z}_i}\times 100\%, \tag{38}$$

  where $\overline{z}_i$ is the average objective function obtained by the algorithm being considered in all runs; z_i^{best} is the best known objective function considering both HAJR and DA;
- $\text{Gap}_b(i)$: best deviation of a method to the best known solution for instance i

$$\text{Gap}_b(i)(\%)=\frac{z_i^{best}-z_i^*}{z_i^*}\times 100\%, \tag{39}$$

  where z_i^* is the best objective function obtained by the algorithm being considered in all repetition runs;
- t and t_{max}: average and maximum computational time, in seconds, needed to solve instances of each family;
- $\#n_{hajr}$ and $\#n_{da}$: number of solutions improved by the HAJR and the DA, respectively, for each scenario.

In all results we used a pool of $P_s=10{,}000$ solutions. In preliminary tests, this was found to be a good compromise between solution quality and computational time. Much smaller pools led to lower solution quality while much larger ones increased computational times.

The results for the Roszieg and Heskia family are presented in Tables 2 and 3, respectively. In the tables, we present the average gaps over all 80 instances for each configuration R and $|T|$. Since each instance was run five times for the HAJR, we present the average results when considering the mean value of the five repetitions and when considering the best value over all repetitions: $\overline{Gap}=\sum_{i=1}^{80}Gap(i)$ and $\overline{Gap_b}=\sum_{i=1}^{80}Gap_b(i)$. Moreover, average and maximum computational times for each class are also reported.

We note that the HAJR outperforms the DA in all scenarios in terms of solution gaps and in terms of computational efficiency, with a lower value of t and t_{max} in most of cases. Although the computer we used in the new tests was a slightly better machine, the time comparisons in these smaller instances is not as relevant

Table 2
Comparision of HAJR and DA for instances of Roszieg family (best $\overline{\text{Gap}}$ in bold).

$\|T\|$	R	HAJR					DA			
		$\overline{\text{Gap}}$ (%)	$\overline{\text{Gap}}_b$ (%)	t (s)	t_{max} (s)	$\#n_{hajr}$	$\overline{\text{Gap}}$ (%)	t (s)	t_{max} (s)	$\#n_{da}$
2	1.05	**0.7 ± 1.7**	**0.1**	10.8	32.9	37	2.3 ± 3.0	73.7	305.1	15
	1.10	**0.5 ± 1.1**	**0.1**	11.0	32.9	34	2.1 ± 2.9	81.1	327.9	10
	1.25	**0.3 ± 0.7**	**0.1**	11.1	32.9	19	1.0 ± 2.0	66.3	325.6	11
	1.50	**0.1 ± 0.3**	**0.0**	11.1	32.9	10	0.5 ± 1.6	51.9	252.4	1
4	1.05	**1.4 ± 1.8**	**0.2**	12.1	32.9	55	6.5 ± 8.4	108.5	488.7	17
	1.10	**1.4 ± 1.7**	**0.1**	12.0	33.1	68	7.1 ± 8.4	125.4	508.6	9
	1.25	**1.4 ± 1.2**	**0.1**	13.3	42.5	63	4.9 ± 4.4	134.6	594.1	15
	1.50	**1.3 ± 1.2**	**0.2**	16.1	100.4	59	4.0 ± 3.5	84.8	295.0	20
7	1.05	**1.7 ± 2.1**	**0.3**	41.0	503.6	62	9.6 ± 10.3	151.2	720.1	16
	1.10	**1.4 ± 1.5**	**0.1**	54.8	1045.8	68	9.3 ± 11.9	183.2	796.9	10
	1.25	**1.2 ± 1.3**	**0.2**	41.3	396.5	53	4.0 ± 8.7	198.6	745.4	18
	1.50	**1.2 ± 1.8**	**0.4**	54.8	777.8	22	1.5 ± 6.5	134.5	528.7	31
Average		**1.0 ± 1.4**	**0.2**				4.4 ± 6.0			

Table 3
Comparision of HAJR and DA for instances of Heskia family (best $\overline{\text{Gap}}$ in bold).

$\|T\|$	R	HAJR					DA			
		$\overline{\text{Gap}}$ (%)	$\overline{\text{Gap}}_b$ (%)	t (s)	t_{max} (s)	$\#n_{hajr}$	$\overline{\text{Gap}}$ (%)	t (s)	t_{max} (s)	$\#n_{da}$
2	1.05	1.7 ± 3.8	**0.5**	33.5	75.4	39	**1.5 ± 1.7**	87.1	893.3	29
	1.10	**1.1 ± 1.6**	**0.3**	34.4	76.8	35	1.4 ± 1.8	102.7	795.9	27
	1.25	1.4 ± 1.1	**0.3**	35.1	81.0	12	**0.6 ± 1.2**	106.4	1124.9	48
	1.50	0.6 ± 0.8	0.1	35.7	83.0	2	**0.0 ± 0.3**	64.1	771.2	34
4	1.05	**3.3 ± 6.6**	**1.4**	36.4	136.4	48	4.1 ± 4.9	130.4	960.7	28
	1.10	**2.2 ± 2.0**	**0.7**	37.7	104.4	39	2.5 ± 3.6	187.3	1200.1	39
	1.25	2.7 ± 1.6	**1.0**	53.7	323.5	19	1.1 ± 1.8	242.5	1506.3	56
	1.50	2.7 ± 1.9	1.2	65.9	559.8	22	**1.0 ± 1.4**	267.0	1866.6	57
7	1.05	**3.9 ± 8.6**	**2.0**	46.5	174.1	46	4.4 ± 5.6	188.0	1096.8	32
	1.10	**2.7 ± 2.4**	**1.1**	55.8	349.0	44	3.2 ± 3.8	271.2	1698.1	33
	1.25	2.4 ± 2.0	1.1	131.0	846.8	25	**0.9 ± 1.5**	396.1	2461.1	52
	1.50	2.9 ± 2.7	1.8	243.3	2076.0	6	**0.2 ± 0.7**	475.3	4260.2	63
Average		2.3 ± 2.9	1.0				**1.7 ± 2.4**			

as the fact that the proposed method is scalable to larger instances while the DA fails in solving even middle-sized ones. HAJR was also more robust than DA, as shown by standard deviations values. Besides, HAJR is able to improve a large number of solutions and present significantly lower gaps.

Table 3 shows the results for the Heskia instances. Overall, the HAJR performs better than the DA for the harder instances (those with smaller values of R). The smaller solution gaps of the HAJR over the DA represented more tasks executed by the workers over the rotation periods. For the Roszieg family, for instance, an average of 3.4% more tasks could be executed at each rotation of 2, 4 or 7 periods. For the larger instances, it is not possible to establish comparisons between the two methods, but just the local search procedures enabled improvements of around 10% in terms of different tasks that could be executed by the workers over the planning period, as can be seen in Tables 4 and 5.

Tables 4 and 5 present the results of HAJR for the largest instances, from Tonge and Wee-Mag families. To measure the quality of solutions, besides using t and t_{max}, we consider the percentage improvement of a solution after local search LSJR1 (third column) and after local searches LSJR1+LSJR2 (fourth column) are applied. In both cases, the results are compared with the original solution obtained by the model (26)–(31). Note that the *decomposition algorithm* of Costa and Miralles (2009) is no longer used as a benchmark for it cannot solve large instances.

Table 4
Results of the HAJR for instances of the Tonge family.

$\|T\|$	R	HAJR			
		After LSJR1 (%)	After LSJR1 + LSJR2 (%)	t (s)	t_{max} (s)
2	1.05	8.0	13.0	1710.6	3136.2
	1.10	3.2	5.2	1708.6	3137.5
	1.25	0.9	1.5	1712.3	3140.1
	1.50	0.5	0.7	1711.4	3142.5
4	1.05	10.5	16.1	1892.8	4365.7
	1.10	7.5	10.6	1949.1	5011.6
	1.25	4.8	6.5	2257.8	5103.4
	1.50	7.3	8.5	2151.7	5112.5
7	1.05	11.6	16.8	2927.5	5881.6
	1.10	9.4	12.6	3166.0	5893.3
	1.25	7.7	9.5	3931.9	8132.2
	1.50	13.2	14.8	5218.8	9345.9
Average		7.1	9.6		

The figures in the tables show that LSJR1 performs well in all the 12 scenarios, yielding improvements of up to 13% over the original solution given by model (26)–(31). The tables also show that LSJR2 can further improve these results. Considering the computational times, a maximum allowed time of 3h was used and never reached. Moreover, the local search procedures had a

Table 5
Results of the HAJR for instances of the Wee-Mag family.

$\|T\|$	R	HAJR			
		After LSJR1 (%)	After LSJR1 + LSJR2 (%)	t (s)	t_{max} (s)
2	1.05	6.5	16.3	926.2	4709.2
	1.10	3.7	9.6	885.1	1413.6
	1.25	2.1	4.8	908.1	1557.5
	1.50	0.9	1.3	978.1	2116.7
4	1.05	8.6	18.2	1497.7	4715.5
	1.10	5.8	11.7	1373.9	4709.6
	1.25	4.8	9.3	3120.7	4862.5
	1.50	5.7	9.5	5887.5	8385.1
7	1.05	9.0	16.7	2427.6	4786.2
	1.10	7.5	12.3	2596.2	4787.5
	1.25	5.7	9.2	3657.0	5264.7
	1.50	6.9	10.2	6814.5	8457.8
Average		5.6	10.8		

maximum allowed computational time of 7200 s (3600 s for each routine). In average, the time spent in both routines was 13 s, 35 s, 821 s and 1714 s for the instances in the Roszieg, Heskia, Tonge and Wee-Mag, respectively.

Since, for these larger instances, no other method in the literature was available for comparison, we implemented a simple random search algorithm (RSA). This kind of algorithms have been used in the literature to evaluate the difficulty of solving problem instances (see, for example, Otto et al., 2011). In our tests of the RSA, we used the same pool of solutions that was generated in the HAJR. From this pool, we randomly selected $|T|$ single schedule solutions and evaluated objective (1). This was repeated for 1000 s and the best feasible solution found was selected. The idea was to observe if a simple random method was able to perform as well as the proposed selection and improvement models described in Section 5, once good solutions for the ALWABP were available.

The computational tests showed that the RSA had difficulties even finding feasible solutions. Indeed, the method found an average of 29.9% of feasible solutions. Among these solutions the average obtained gap was 9.8%. It is important to highlight that these figures were obtained while considering both the easier (larger values of R) and the most difficult (smaller values of R) instances. When these results are evaluated by groups of instances, it becomes even clearer the inefficiency of the method for hard instances. Indeed, the RSA found an average of 0.0%, 1.3%, 28.2% and 86.7% of feasible instances for the instances with values of R equal to 1.05, 1.10, 1.25 and 1.50, respectively.

An interesting aspect to analyze is the composition of the final solution in terms of the original methods used to generate their single schedule solutions. Table 6 shows the percentage of solutions coming from the HGA, CH, TS and GRASP that were selected by the model (26)–(31). The proposed GRASP algorithm gives most of solutions in the last three families, while the TS was important in the Roszieg and Heskia families. Although the HGA and the CH contributed less in the first two cases, they had a fundamental role in the composition of the pool for the largest instances. Our conclusion is that the set of heuristic methods provide a diversity of solutions that was needed for the success of HAJR.

To further study this point, we analyzed the efficiency of the HAJR algorithm when just part of the methods described in Section 4 were used. Sixteen instances in families Tonge and Wee-Mag (the first instance in each family group) were solved for all combinations of parameters R and $|T|$ and the conclusion was that the diversity was needed for the method to obtain good

Table 6
Percentage of solutions coming from the heuristic methods chosen by model (26)–(31).

Family	Heuristic method			
	HGA (%)	CH (%)	TS (%)	GRASP
Roszieg	5.6	0.5	71.9	22.0
Heskia	5.4	3.7	28.9	61.9
Tonge	28.9	6.5	2.9	61.6
Wee-Mag	37.5	16.6	1.1	44.8

Table 7
Average solution gap obtained by HAJR with different pool construction methods (instances Tonge and Wee-Mag; $R=1.5$).

T	Heuristic methods			
	CH+HGA (%)	TS (%)	GRASP (%)	Complete HAJR (%)
2	0.2	0.5	0.1	0.1
4	5.2	9.8	2.0	1.0
7	8.5	17.1	2.7	1.2
Average	4.7	9.1	1.6	0.73

solutions. Indeed, when the method was run only with the Constructive Heuristics and the Genetic Algorithm, feasibility was reached in 65.9% of the cases. In these cases, an average gap of 4.7% was obtained. The results when only the Tabu Search or the GRASP methods were used were similar. When Tabu Search was used, the method obtained an average of 32.7% feasible instances (with an average gap, for these instances, of 9.1%) and when GRASP was used, the method obtained an average of 47.7% feasible instances (for which an average gap of 1.6% was reached). Table 7 shows the percentage gaps obtained by the complete HAJR and by the HAJR with each of the ALWABP solution methods. In order to enable a fair comparison, in the table, the gaps are computed only over the instances for which all methods obtained feasible solutions and, therefore, the situation with $R=1.5$ was selected for illustration.

In concordance with Table 7, GRASP was the pool initialization method that was able to obtain the better results. Nevertheless, it is still less effective than using a diverse set of methods, even for the easiest instances ($R=1.5$).

7. Conclusions

The purpose of this study was to examine the problem of balancing assembly lines with heterogeneous workers while considering job rotation schedules. We have developed a hybrid algorithm (HAJR) which uses heuristics methods, and mixed-integer programs to select the initial solutions and as post-optimization improvement methods. In a series of computational tests, this approach proved flexible, accurate and much more scalable when compared with the *decomposition algorithm* Costa and Miralles (2009). We believe this success is due to the diversity of solutions obtained by the different proposed heuristic algorithms that composed the structure of the HAJR, and to the efficiency of the proposed MIP neighborhoods.

Acknowledgments

We thank Dr. Marcus Ritt for kindly providing the results of the biased random-key genetic algorithm used in this paper. We

also thank three anonymous referees for a number of important comments which have helped to improve both the quality and the presentation of this paper. Finally, we are grateful to CAPES and FAPESP (Brazil) and MEC (Spain) and for financing this research under projects CAPES/DGU 258/12 and FAPESP 2010/19983-6.

References

Aryanezhad, M.B., Kheirkhah, a.S., Deljoo, V., Mirzapour Al-e-hashem, S.M.J., 2008. Designing safe job rotation schedules based upon workers' skills. The International Journal of Advanced Manufacturing Technology 41 (March), 193–199.

Azizi, N., Zolfaghari, S., Liang, M., 2010. Modeling job rotation in manufacturing systems: the study of employee's boredom and skill variations. International Journal of Production Economics 123, 69–85.

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

Blum, C., Miralles, C., 2011. On solving the assembly line worker assignment and balancing problem via beam search. Computers and Operations Research 38, 328–339.

Blum, C., Puchinger, J., Raidl, G., Roli, A., 2011. Hybrid metaheuristics. In: van Hentenryck, P., Milano, M. (Eds.), Hybrid Optimization, Springer Optimization and Its Applications, vol. 45. Springer, New York, pp. 305–335.

Boysen, N., Fliedner, M., 2008. A versatile algorithm for assembly line balancing. European Journal of Operational Research 184 (January), 39–56.

Boysen, N., Fliedner, M., Scholl, A., 2007. A classification of assembly line balancing problems. European Journal of Operational Research 183 (December), 674–693.

Boysen, N., Fliedner, M., Scholl, A., 2008. Assembly line balancing: which model to use when? International Journal of Production Economics 111 (February), 509–528.

Boysen, N., Scholl, A., Wopperer, N., 2012. Resequencing of mixed-model assembly lines: survey and research agenda. European Journal of Operational Research 216 (February), 594–604.

Carnahan, B.J., Redfern, M.S., Norman, B., 2000. Designing safe job rotation schedules using optimization and heuristic search. Ergonomics 43, 543–560.

Chaves, A.A., Lorena, L.A.N., Miralles, C., 2009. Hybrid metaheuristic for the assembly line worker assignment and balancing problem. Lecture Notes on Computer Science 5818, 1–14.

Chaves, A.A., Miralles, C., Lorena, L.A.N., 2007. Clustering search approach for the assembly line worker assignment and balancing problem. In: Proceedings of the 37th International Conference on Computers and Industrial Engineering, Alexandria, Egypt. pp. 1469–1478.

Costa, A.M., Miralles, C., 2009. Job rotation in assembly lines employing disabled workers. International Journal of Production Economics 120, 625–632.

El-Abd, M., Kamel, M., 2005. A taxonomy of cooperative search algorithms. Lecture Notes on Computer Science 3636, 32–41.

Feo, T., Resende, M., 1989. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8, 67–71.

Gendreau, M., Hertz, A., Laporte, G., 1994. A tabu search heuristic for the vehicle routing problem. Management Science 40, 1276–1290.

Ghosh, S., Gagnon, R.J., 1989. A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. International Journal of Production Research 27, 637–670.

Günther, R., Puchinger, J., Blum, C., 2010. Metaheuristic Hybrids, Handbook of Metaheuristics. Springer, pp. 469–496.

Maniezzo, V., Stützle, T., Voß, S. (Eds.), 2009. Matheuristics: Hybridizing Metaheuristics and Mathematical Programming. Springer.

Miralles, C., García-Sabater, J., Andrés, C., Cardós, M., 2007. Advantages of assembly lines in sheltered work centers for disabled: a case study. International Journal of Production Economics 110, 187–197.

Miralles, C., García-Sabater, J., Andrés, C., Cardós, M., 2008. Branch and bound procedures for solving the assembly line worker assignment and balancing problem: application to sheltered work centers for disabled. Discrete Applied Mathematics 156, 352–367.

Moreira, M.C.O., Costa, A.M., 2009. A minimalist yet efficient tabu search for balancing assembly lines with disabled workers. In: Annals of the XLI Simpósio Brasileiro de Pesquisa Operacional. Porto Seguro, Brazil, pp. 660–671.

Moreira, M.C.O., Ritt, M., Costa, A.M., Chaves, A.A., 2012. Simple heuristics for the assembly line worker assignment and balancing problem. Journal of Heuristics 18, 505–524.

Otto, A., Otto, C., Scholl, A., 2011. SALBPGen—a systematic data generator for (simple) assembly line balancing. Jena Research Papers in Business and Economics, School of Economics and Business Administration, Friedrich-Schiller-University Jena 05, pp. 1–27.

Puchinger, J., Raidl, G., 2005. Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification. Lecture Notes on Computer Science 3562, 41–53.

Rochat, Y., Taillard, E., 1995. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics 1, 147–167.

Salveson, M.E., 1955. The assembly line balancing problem. Journal of Industrial Engineering 6, 18–25.

Sato, T., Coury, H., 2009. Evaluation of musculoskeletal health outcomes in the context of job rotation and multifunctional jobs. Applied Ergonomics 40, 707–712.

Scholl, A., 1999. Balancing and Sequencing of Assembly Lines, 2nd ed. Physica-Verlag.

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., Fliedner, M., Boysen, N., 2010. Absalom: balancing assembly lines with assignment restrictions. European Journal of Operational Research 200 (February), 688–701.

Scholl, A., Voß, S., 1996. Simple assembly line balancing—heuristic approaches. Journal of Heuristics 2, 217–244.

Seçkiner, S.U., Kurt, M., 2007. A simulated annealing approach to the solution of job rotation scheduling problems. Applied Mathematics and Computation 188, 31–45.

Seçkiner, S.U., Kurt, M., 2008. Ant colony optimization for the job rotation scheduling problem. Applied Mathematics and Computation 201, 149–160.

Tharmmaphornphilas, W., Norman, B.A., 2007. A methodology to create robust job rotation schedules. Annals of Operations Research 155, 339–360.

Tonge, F.M., 1961. A Heuristic Program for Assembly Line Balancing. Prentice-Hall, Englewood Cliffs, NJ.