Including workers with disabilities in flow shop scheduling

Germano C. Carniel[1] and Alexander J. Benavides[1] and Marcus Ritt[1] and Cristóbal Miralles[2]

***Abstract*— Persons with disabilities have severe problems participating in the job market and their unemployment rate is usually much higher than the average of the population. This motivates the research of new modes of production which allow to include these persons at a low overhead.**

In this paper we study the inclusion of persons with disabilities into flow shops with the objective of minimizing the makespan. Since flow shops usually have only a few machines, we focus on the inclusion of one and two workers. We define the problem, propose mathematical models and a heuristic solution, as well as realistic test instances. In computational tests we evaluate the performance of the models and the heuristic, and assess the utility of such a model of inclusion. We conclude that the problem can be solved satisfactorily, and that including workers with disabilities into flow shops is economically feasible.

I. Introduction

In 2004, the World Health Survey and the Global Burden of Disease project estimated the population of persons with a disability of 15 years and older around 785 (15.6%) to 975 (19.4%) million. According to the World Health Organization and the International Labour Organization, unemployment rates are much higher for persons with disabilities than for persons without disabilities in both developed and developing countries. However, almost all tasks can be performed by persons with disabilities, since they often have the necessary skills, and most of them can be productive in an appropriate environment [1]. Motivated by similar studies that have demonstrated that such workers can be integrated successfully in assembly lines (e.g. [2]), we study the integration of workers with disabilities into flow shops with the objective of minimizing the makespan. Since flow shops have relatively few machines, and legislation usually foresees an integration of about 2% to 5% of workers with disabilities, we focus on the case of the integration of one and two workers into a flow shop.

In Section II we motivate two variants of the problem of the integration of workers with disabilities into flow shops. In Sections III and IV we formulate mathematical models and propose heuristic solutions for the problems. In Section V we propose instances modelling realistic conditions, and present computational experiments with the models and the heuristics. We conclude in Section VI.

[1] Germano C. Carniel, Alexander J. Benavides, and Marcus Ritt are with Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brazil.

[2] Cristóbal Miralles is with ROGLE, Departamento de Organización de Empresas, Universitat Politècnica de València, Spain.

A. Literature review

Several researchers have been studying the problem of integrating persons with disabilities in production processes. For assembly lines, [2] have proposed the Assembly Line Worker Assignment and Balancing Problem (ALWABP). In this problem the task execution time depends on the worker, and tasks as well as workers must be assigned to a fixed number of stations such that the production rate of the assembly line is maximized. This problem is NP-hard and research has focused mainly on heuristic methods for solving it [3]–[6] but effective exact solution techniques are available [7], [8].

[9] investigated flow shops with heterogeneous workers, where each machine is operated by a different worker. The specific case of integration of workers with disabilities into flow shops has, to the best of our knowledge, not been studied in the literature so far.

The flow shop scheduling problem (FSSP) has been extensively studied in the literature. Most research focuses on the simpler permutation flow shop problem (PFSSP), where all the jobs have to be processed in the same order on all machines, although [10] showed that there are instances for which the makespan of an optimal solution for the PFSSP is worse than the optimal solution for the non-permutation FSSP by more than a factor of $\sqrt{m}/2$. The PFSSP can be solved in polynomial time for two machines but is NP-hard for three or more machines [11]. A variant we study here is the *hybrid flowshop* where jobs are processed in stages, and each stage can have multiple parallel machines. For more details on flow shops and hybrid flow shops we refer the reader to the surveys of [12] and [13].

II. Workers with disabilities in Flow shops

In a FSSP, we have to schedule jobs $J_1, \dots, J_n$ on machines $M_1, \dots, M_m$. Each job J_i must be processed on machine M_r in time p_{ri} without preemption. The jobs cannot be processed in parallel, and the machines can process only one job at a time. In the PFSSP solutions are restricted to permutation schedules, where the jobs have to be processed on all machines in the same order.

The most common objective for the FSSP and the PFSSP is to minimize the makespan $C_{\max} = \max C_i$, i.e. the maximum over all completion times C_i, $i \in [n]$ of the jobs (We use the notation $[n] = \{1, 2, \dots, n\}$.). Examples of other objectives include the total completion time $\sum C_i$, or the total lateness or tardiness. Here we focus on the minimization of the makespan of the PFSSP.

	Regular				With disabilities			
Job	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4
J_1	1	2	2	1	2	4	2	∞
J_2	1	1	2	2	1	1	4	∞
J_3	2	1	1	2	4	2	1	∞
J_4	1	3	2	1	1	4	2	∞

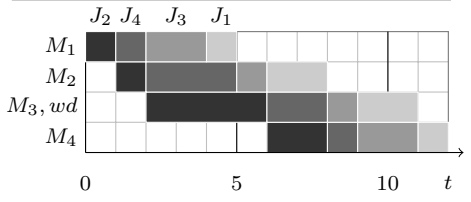

Fig. 1. An instance of PFSISP, where a single WWD must operate one machine in a flow shop. Above: Processing times for regular worker and WWD. Below: Gantt chart of optimal schedule of makespan 12 which assign the WWD to machine M_3. The optimal regular schedule has makespan 11.

	Regular				WWD 1				WWD 2			
Job	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4	M_1	M_2	M_3	M_4
J_1	1	2	2	1	2	4	2	∞	2	3	3	∞
J_2	1	1	2	2	1	1	4	∞	2	2	3	∞
J_3	2	1	1	2	4	2	1	∞	4	1	1	∞
J_4	1	3	2	1	1	4	2	∞	2	5	4	∞

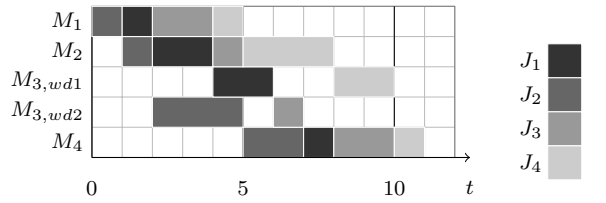

Fig. 2. An instance of HPFSISP, where two workers with disabilities must operate a dual-machine stage in a flow shop. Above: Processing times for regular worker and WWDs. Below: Gantt chart of an optimum solution of makespan 11.

A. Inserting a single worker into a flow shop

The situation encountered when we need to integrate workers with disabilities into a regular workforce can be seen in the upper part of Figure 1. In addition to the times that regular workers take to perform the operations, we have times for a worker with disabilities (WWD), which normally exceed the time of the regular workers. In the example, the times of the WWD were chosen randomly in the interval $[p, 2p]$, for a processing time of p of a regular worker. The WWD may also be unable to operate some of the machines. In the example, this is the case for machine M_4 represented by processing times of ∞ on this machine.

The problem of inserting a WWD into a flow shop (flow shop insertion and scheduling problem, FSISP) is defined as follows: we have to assign the WWD to a machine she is able to operate, and find a valid schedule of the jobs, such that the makespan is minimized. We call the variant restricted to permutation schedules the permutation flow shop insertion and scheduling problem (PFSISP).

B. Inserting two workers into a hybrid flow shop

When assigning a single, slow worker to a machine in a flow shop, she will likely be a bottleneck and increase the makespan. Therefore we also study a hybrid flow shop in which two WWDs are assigned to a single stage with two parallel machines. This allows to integrate more WWDs into the production line and has the potential to compensate for their increased execution times. We call this problem the Hybrid Flow Shop Insertion and Scheduling Problem (HFSISP), and its permutation variant the Hybrid Permutation Flow Shop Insertion and Scheduling Problem (HPFSISP). Using the notation of [13] the permutation variant is denoted by $FHm \mid (R2)^k, (1)^i_{i\in[m]\setminus\{k\}}, prmu \mid C_{\max}$.

An instance of these problems consists of the processing times for the regular workers, and two sets of processing times for WWDs. A solution is given by an assignment of the two workers to some stage, and a processing order of the jobs. In the permutation version, we require the jobs to obey the processing order on each of the parallel machines.

An example for the insertion of two WWDs is shown in Figure 2. Note that the parallel stage was able to compensate for the longer processing times of the WWDs and the schedule has makespan 11, as in the regular case.

III. Mathematical models

A. PFSISP

The model for the PFSISP is based on the best known model for the PFSSP. The main principle of the model is to define the starting times of job J_i, $i \in [n]$ at machine M_r, $r \in [m]$, by the sum of its predecessors on the first machine, plus the sum of its processing times on machines $1, \ldots, r-1$ and the waiting time on these machines before starting on the next machines. The comparison of these partial sums yields compact constraints for expressing the relative starting times for a given job permutation.

In the model we use subscripts $r \in [m]$ for machines, $i \in [n]$ for jobs, and $j \in [n]$ for sequence positions. We denote by A the set of machines the WWD can operate, by p_{ri} the processing time of job J_i on machine M_r of a regular worker, by d_{ri} the processing time of job J_i on machine M_r of the WWD, by C_{ri} the completion time of job J_i on machine M_r, by Y_{rj} the waiting time of the job in sequence position j after it finishes on machine M_r, and by TT_{rj} the processing time of the job in sequence position j on machine M_r after assigning the WWD. We further use binary variables Z_{ij} to indicate that job J_i is assigned to sequence position j, and X_r to indicate that the WWD is assigned to machine M_r.

$$\textbf{min.} \quad \sum_{p\in[n]} TT_{1p} + \sum_{q\in[2,m]} TT_{qn} + \sum_{q\in[m-1]} Y_{qn}, \quad \textbf{s.t.}, \tag{1}$$

$$\sum_{i\in[n]} Z_{ij} = 1, \qquad \forall i \in [m], \tag{2}$$

$$\sum_{j\in[n]} Z_{ij} = 1, \qquad \forall i \in [n], \tag{3}$$

$$TT_{1,j-1} - TT_{r,j-1} + \sum_{q\in[r-1]} TT_{qj} - TT_{q,j-1} + \sum_{q\in[r-1]} Y_{qj} - Y_{q,j-1} \geq 0, \qquad \forall r \in [2,m], j \in [2,n], \quad (4)$$

$$TT_{rj} = \sum_{i\in[n]} p_{ri}(1-X_r)Z_{ij} + d_{ri}X_rZ_{ij}, \qquad \forall r \in [m], j \in [n], \quad (5)$$

$$\sum_{r\in A} X_r = 1. \quad (6)$$

The objective function (1) is the sum of three components: the processing time of all jobs on the first machine, the processing time of the last job on all remaining machines, and the waiting times of the last job on all machines, except the last one. Constraints (2) and (3) model the assignment of jobs to positions: each job is assigned to only one sequence position and each sequence position has only one job assigned to it. Constraint (4) relates the starting times of the jobs at sequence positions $j-1$ and j according to the principle explained above. Constraint (5) defines the processing times of the jobs at each sequence position according to the assignment of the WWD to one of the machines she is able to operate. Note that these constraints are non-linear, but can be easily linearized using standard methods, by introducing n^2m auxiliary binary variables and n^2m additional constraints. Constraint (6) requires that the WWD is assigned to one of the machines she can operate.

B. HPFSISP

The formulation uses indices j and q for jobs, k for stages, l for machines and w for workers. The model uses dichotomous constraints for ordering the jobs on each machine. We denote by A the set of stages that the WWDs can operate, by p_{jk} the processing time of job J_j on stage k for a regular worker, by d_{jwk} the processing time of job J_j on stage k by the WWD w, by T_{jk} the processing time of job J_j on stage k after assigning the WWDs to some stage, and by C_{jk} by completion time of job J_j on stage k. We further use binary variables U_{jkl} to indicate that job J_j is assigned to machine l on stage k, P_{jq} to indicate that job J_j precedes job J_q, X_k to indicate that the WWDs are assigned to stage k, and W_{wl} to indicate that WWD w is assigned to machine l on the parallel stage. The constant Q can be set to $\sum_{j\in[n]}\sum_{k\in[m]} \max_{w\in[2]} d_{jkw}$.

$$\textbf{min.} \quad C_{\max}, \qquad \textbf{s.t.} \quad (7)$$

$$C_{\max} \geq C_{jm}, \qquad \forall j \in [n], \quad (8)$$

$$\sum_{l\in[2]} U_{jkl} = 1, \qquad \forall j \in [n], k \in [m], \quad (9)$$

$$U_{jk2} \leq X_k, \qquad \forall j \in [n], k \in [m], \quad (10)$$

$$C_{jk} - T_{jk} \geq C_{j,k-1}, \qquad \forall j \in [n], k \in [m], \quad (11)$$

$$Q(2 - U_{jkl} - U_{qkl} + P_{jq}) + C_{jk} - T_{jk} \geq C_{qk}, \qquad \forall j,q \in [n], k \in [m], l \in [2], \quad (12)$$

$$Q(3 - U_{jkl} - U_{qkl} - P_{jq}) + C_{qk} - T_{qk} \geq C_{jk}, \qquad \forall j,q \in [n], k \in [m], l \in [2], \quad (13)$$

$$T_{jk} = p_{jk}(1-X_r) + \sum_{l\in[2]} (d_{jkw}X_kW_{wl}), \quad \forall j \in [n], k \in [m], l,w \in [2], \quad (14)$$

$$\sum_{k\in A} X_k = 1, \quad (15)$$

$$\sum_{l\in[2]} W_{wl} = 1, \qquad \forall w \in [2], \quad (16)$$

$$\sum_{w\in[2]} W_{wl} = 1, \qquad \forall l \in [2], \quad (17)$$

$$C_{jk} \geq 0 \qquad \forall j \in [n], k \in [m]. \quad (18)$$

The objective function (7) is defined as the latest completion time in constraint (8). Constraint (9) requires that all jobs are assigned to only one of the machines at each stage. Constraint (10) guarantees that only the stage to which the workers with disabilities have been assigned can use a second, parallel machine. Constraint (11) ensures that each job can only start on a stage after it has finished on the previous one. Constraints (12) and (13) prevent any two jobs to be executed on the same machine simultaneously. The processing time of a job depends on the stage the workers with disabilities were assigned to, and is defined in constraint (14). Constraint (15) requires that the WWDs are assigned to one of the stages they can operate. Finally, constraints (16) and (17) require that each worker is assigned to exactly one of the parallel machines and each machine has only one worker assigned to it. Again, the model is not linear due to the term X_kW_{wl} in constraint (14), but can be linearized by introducing $4m$ auxiliary binary variables and $4m$ additional constraints.

IV. Heuristics for PFSISP and HPFSISP

To solve a worker insertion problem we must find the best stage for the WWDs and an optimal schedule of the operations. For a single worker, a simple solution is to solve a standard PFSSP for each possible insertion in one of the m stages. This makes it possible to use existing methods to solve each subproblem, but ignores that some stages may be better for inserting the worker than others, and thus should receive more search time. We propose a pooled strategy to solve this problem. The same approaches work when inserting a pair of workers, but we additionally have to solve the subproblem of scheduling the jobs on the stage with two parallel machines.

We have chosen to base our heuristic on an iterated greedy algorithm, a special form of an iterated local search [14]. Both methods were successful in finding good schedules for the permutation flow shop and related problems. For most of these problems they are, or are part of, the current best heuristics (see e.g. [15], [16]).

An iterated local search starts from an initial solution and applies a local search to find a local optimum. The local optimum is perturbed and the local search is applied again. The new local optimum often must satisfy an acceptance criterion to avoid perturbations that lead to much worse solutions. These steps are repeated until

Algorithm 1 Iterated greedy algorithm.

Input: A permutation schedule π.
Output: An improved permutation schedule π'.

function IGA(π)
  π := shift-localsearch(π)
  repeat
    remove d random jobs $j_1, \ldots, j_d$ from π to get π'
    for $i \in [d]$ **do**
      insert j_i into π' at the pos. of minimal $C_{\max}(\pi')$
    end for
    π' := shift-localsearch(π')
    if accept(π, π') **then**
      $\pi := \pi'$
    end if
  until some stopping criterion is satisfied
  return the best solution π^* found during the search
end function

some stopping criterion is satisfied. An iterated greedy algorithm (IGA) perturbs the solution using a greedy algorithm. A solution is *destroyed*, by removing randomly some of its elements, and the partial solution is then *reconstructed* to another solution by inserting the removed elements greedily. Besides the neighborhood, the amount of perturbation is the most important parameter of an iterated local search. If it is too small, the solution will stay at the current local optimum; if it is too large we obtain a randomized multi-start local search.

In a PFSSP, a solution is a permutation π of the jobs. A good initial solution can be found by the greedy constructive heuristic NEH or some of its more recent variants [17], which are the currently best known constructive heuristics for the PFSSP. NEH starts from an empty job sequence, processes the jobs in order or non-increasing total processing time $P_j = \sum_{i\in[m]} p_{ij}$, and inserts each job at the position which minimizes the makespan of the resulting partial sequence. Variants of NEH differ mainly in the way ties are broken among multiple positions [16].

Typical neighborhoods for local searches on job permutations are defined by *shifting* a job from its current position to another position, or by *swapping* the positions of two jobs. We use a shift neighborhood, which is the most common choice for the permutation flow shop, since the complexity of $O(n^2m)$ for finding the best insertion position of a job can be reduced to $O(nm)$ using an algorithm of [18]. The same optimization can be used to speed up the construction of an initial solution via NEH, and the perturbation step of the IGA, since both repeatedly insert jobs. If π is the current local minimum, a new local minimum π' is accepted if it satisfies a Metropolis criterion, i.e. with probability $P[\text{accept}(\pi,\pi')] = \min\{e^{-}\Delta(\pi,\pi')/T, 1\}$ for an increase of the objective function by $\Delta(\pi,\pi') = C_{\max}(\pi') - C_{\max}(\pi)$ and a temperature $T = \alpha\overline{p}/10$ where $\overline{p} = \sum_{j\in[n]}\sum_{i\in[m]} p_{ij}/nm$ is the average processing time of an operation, and a parameter α. The IGA for the PFSSP is shown in Algorithm 1.

Algorithm 2 A pooled IGA for PFSISP and HPFSISP.

Output: A solution (π, k) for the PFSISP or HPFSISP

$P := \{(\text{NEH}(k), k) \mid k \in [m]\}$ ▷ create the solution pool
while $|P| > 1$ **do**
  for all $(\pi, k) \in P$ **do**
    $(\pi, k) := (\text{IGA}(\pi, t), k)$
  end for
  $(\pi_0, k_0) := \text{argmax}_{(\pi,k)\in P} C_{\max}(\pi)$
  $P := P \setminus \{(\pi_0, k_0)\}$
end while
return the single solution (π, k) in the pool

A pooled IGA for inserting WWDs into flow shops

To find the best stage to insert the WWDs, we propose Algorithm 2, a pooled variant of an IGA. It initially creates a pool with m candidate solutions. Each candidate solution assigns the WWDs to one of the m stages, and applies the NEH heuristic to obtain an initial solution. The method then proceeds in m phases. In each phase, it applies the IGA to improve each candidate solution for a fixed time t and then discards the solution of the worst makespan in the pool. The total running time is therefore $\binom{m}{2}t$, and the kth best solution receives $(m+1-k)t$ of it. This ensures that solutions with a shorter makespan receive more time than those that get stuck early. This approach is preferable to more complex methods like [9], when only a few WWDs have to be inserted. In Algorithm 2, function NEH(k) returns an initial solution using the NEH algorithm, when assigning the WWDs to the kth stage, and function IGA(π, t) improves the current schedule π by applying the IGA for time t.

Solving the two-machine subproblem

A solution of the HPFSISP assigns two workers to a stage with two parallel machines, and must additionally solve the subproblem of finding an optimal schedule for this stage. This subproblem can be formulated as a head-body-tail problem on two unrelated machines. For a permutation π of the jobs, and an assignment of the WWDs to stage k, define heads $r_j = r_{k-1,j}$, and tails $q_j = q_{j,k+1}$, where r_{ij} is the earliest completion time of job j on stage i, defined by $r_{ij} = 0$, if $i = 0$ or $j = 0$, and

$$r_{ij} = \max\{r_{i,\pi(\pi^{-1}(j)-1)}, r_{i-1,j}\} + p_{ij}$$

otherwise, and q_{ij} is the shortest time from the start of job j on stage i to the completion of the last operation, defined by $q_{ij} = 0$ if $i = m+1$ or $j = n+1$, and

$$q_{ij} = \max\{q_{i,\pi(\pi^{-1}(j)+1)}, q_{i+1,j}\} + p_{ij},$$

otherwise. Then we have to find starting times S_j for the jobs $j \in [n]$ on the two machines, such that $S_i \geq r_j$, minimizing $C_{\max} = \max_j S_j + q_j$. Since we impose the order of the permutation flowshop π on all machines the problem reduces to finding an optimal assignment of the jobs to the parallel machines. This problem is NP-hard, since it generalizes $P2 \mid\mid C_{\max}$ [19], but can be solved by dynamic programming in time $O(n\overline{C}^2)$ for some upper

TABLE I

Sizes of the test instances.

Inst.	n	m	Inst.	n	m	Group	n	m
car1	11	5	car6	8	9	ta03	20	20
car2	13	4	car7	7	7	ta04	50	5
car3	12	5	car8	8	8	ta05	50	10
car4	14	4	ta01	20	5	ta06	50	20
car5	10	6	ta02	20	10			

bound $\overline{C}$ on the makespan. Let $C(j, t_1, t_2)$ be the minimal completion time when scheduling jobs $j, \ldots, n$ on the two parallel machines, starting not earlier than t_1 on the first, and not earlier than t_2 on the second machine. Then the optimal solution is given by $C(1, 0, 0)$, where

$$C(j, t_1, t_2) = \min\{ \max\{C_1(t_1, j) + q_j, C(j+1, C_1(t_1, j), t_2)\}, \max\{C_2(t_2, j) + q_j, C(j+1, t_1, C_2(t_2, j))\}\}$$

and $C_l(t, j) = \max\{t, r_j\} + p_{jl}$ is the completion time of job J_j when starting not earlier than t on parallel machine l, with a base case of $C(n+1, t_1, t_2) = 0$. We further have tested a heuristic, which processes the jobs in order, and greedily assigns each job to the parallel machine that results in the earliest completion time.

V. Computational Experiments

In this section we report the results of computational tests. We first analyze the performance of the models, an exact algorithm and the heuristics on small instances, that can be solved to optimality. In a second experiment we evaluate the heuristics on instances of practical sizes.

A. Test instances and experimental methodology

For the computational experiments we used nine small instances from [20] and 60 large instances from [21] with up to 50 jobs and 20 machines. Table I shows the number of jobs (n) and machines (m) of the instances. For each size there is a group of 10 Taillard instances.

We created instances for the worker inclusion problems based on these instances as follows. We assume that the processing times p_{ri} of a flow shop instance are those of a regular worker. To model a WWD, we modify the processing times in two ways. First, a fixed percentage of incompatibilities is introduced. An incompatibility models a worker who is unable to operate some machine (e.g. the WWD in Figure 1 cannot operate M_4). Second, the processing times are increased to reflect that a WWD usually needs more time to execute an operation. Based on experience with workers in SWDs, we generated instances with 0%, 10%, and 20% of incompatibilities per worker, and processing times chosen uniformly at random in $[p, 2p]$ or $[p, 5p]$, for a regular processing time p. With 3 levels of incompatibilities and 2 levels of operation time variation, we obtain a total of 408 test instances.

TABLE II

Results for Carlier instances when inserting one worker.

		CPLEX		LOMPEN		Heuristics		
Var.	Inc.	$\bar{t}$	Rd.	$\bar{t}$	Rd.	S	P	PL
2	0	26.7	7.4	0.1	7.4	7.4	7.4	7.4
2	10	17.8	7.9	0.1	7.9	7.9	7.9	7.9
2	20	14.5	9.2	0.1	9.2	9.2	9.3	9.3
5	0	55.7	75.8	0.0	75.8	75.8	75.8	75.8
5	10	46.7	75.8	0.0	75.8	75.8	75.8	75.8
5	20	11.3	77.7	0.0	77.7	77.7	77.7	77.7
Avg.		28.8	42.3	0.0	42.3	42.3	42.3	42.3

The mathematical models were solved using the commercial solver CPLEX 12.5 running with a single thread and a time limit of 1 h. Our heuristics were implemented in C++, and compiled with GNU C++ 4.7.3 with maximum optimization. The IGA sets $d = 4$ and $\alpha = 0.4$, identified as the best by [14], and stops after $3nm$ ms. All tests were done on a PC with an Intel Core i7 processor running at 2.8 GHz and 12 GB of main memory. We report the solution quality as the relative deviation $(C_{\max} - C^*_{\max})/C^*_{\max}$ from the best known makespan $C^*_{\max}$ of the corresponding flow shop instance, to be able to evaluate the impact of inserting WWDs. For experiments with the heuristics the relative deviations are averages over 5 replications with different seeds. All times are reported in seconds.

B. Experiments with the Carlier instances

On the Carlier instances, we conducted four experiments. We solved the mathematical models for PFSISP and HPFSISP, and for the PFSISP we also solved the problems exactly with the branch-and-bound algorithm LOMPEN [22], by solving an FSSP problem for each stage separately with a time limit of 2 h. We further evaluated the simple (S) and pooled (P) IGA heuristics. The pooled heuristic has been run with our a time limit of $3nm$ ms and a longer time limit of $3nm^2$ ms (PL). For the HPFSISP we also solved the scheduling problem on the parallel stage by dynamic programming (PD).

Table II reports the relative deviations when inserting one worker, and Table III when inserting two workers. For CPLEX and LOMPEN, we also report the solution time ($\bar{t}$). All instances inserting one worker could be solved to optimality. When inserting two workers, CPLEX was able to solve only about 80% of the instances in 1 h, and we further report the optimality gap (Gap).

The PFSISP is easy too solve on Carlier's instances. LOMPEN finds the optimum very quickly, and the heuristics also find the optimal solutions in at most 2 s. The HPFSISP is harder to solve, and CPLEX's average solution time increases by a factor of almost 40. The heuristics find good solutions with a makespan that is at most 0.4% longer in 1/500th of the time. We can also see that there are very small differences between the different heuristic strategies and time limits.

TABLE III

Results for Carlier instances when inserting two workers.

		CPLEX			Heuristics			
Var.	Inc.	Gap	$\bar{t}$	Rd.	S	P	PL	PD
2	0	6.9	1499.5	-4.2	-4.1	-4.0	-4.0	-3.6
2	10	6.4	1151.0	-2.2	-2.1	-2.1	-2.1	-2.1
2	20	5.1	1524.7	-0.6	-0.5	-0.5	-0.5	-0.4
5	0	4.1	875.8	3.6	4.6	4.3	4.3	4.6
5	10	3.7	899.0	5.0	5.8	5.5	5.6	5.5
5	20	3.7	788.2	5.4	6.1	5.9	5.8	5.7
Avg.		5.0	1123.0	1.2	1.6	1.5	1.5	1.6

TABLE IV

Results for Taillard instances with time variation $[p, 2p]$.

		LOMPEN		One worker			Two workers		
G	I	$\bar{t}$	Rd.	S	P	PL	S	P	PL
1	0	0.4	14.7	14.7	14.7	14.7	-3.4	-3.6	-3.7
2	0	3141.9	2.5	3.6	3.7	3.6	-0.3	-0.9	-1.3
3	0	54258.4	1.2	1.4	1.4	1.3	-0.0	-0.4	-0.9
4	0	6.9	27.4	27.4	27.4	27.4	-0.8	-1.3	-1.3
5	0	127.0	19.2	19.4	19.4	19.4	3.0	0.7	-0.4
6	0	21934.4	5.2	5.1	5.0	4.2	4.1	3.0	0.8
Avg.		13244.9	11.7	11.9	11.9	11.8	0.4	-0.4	-1.1
1	10	0.4	17.0	17.0	17.0	17.0	-3.5	-3.6	-3.7
2	10	3795.5	3.5	3.7	3.7	3.7	-0.3	-0.7	-1.2
3	10	48653.9	1.3	1.5	1.5	1.4	-0.1	-0.2	-0.9
4	10	6.2	28.4	28.4	28.4	28.4	-0.3	-0.9	-1.0
5	10	80.4	20.2	20.2	20.2	20.2	3.2	0.9	-0.3
6	10	19483.3	5.5	5.3	5.3	4.7	4.1	3.1	0.8
Avg.		12003.3	12.6	12.7	12.7	12.6	0.5	-0.2	-1.0
1	20	0.4	18.3	18.3	18.3	18.3	-1.7	-1.9	-2.0
2	20	1872.0	5.2	5.3	5.5	5.3	-0.1	-0.6	-0.9
3	20	44304.2	1.3	1.4	1.5	1.4	0.0	-0.2	-0.8
4	20	4.7	28.7	28.7	28.7	28.7	-0.0	-0.7	-0.7
5	20	76.4	20.5	20.5	20.6	20.6	3.2	1.3	0.2
6	20	16410.8	5.8	5.8	5.6	5.0	4.1	3.1	0.8
Avg.		10444.8	13.3	13.3	13.4	13.2	0.9	0.2	-0.6

We can observe that inserting a single worker introduces, as expected, a large overhead, in particular for a high time variation, whereas the makespan does not increase more than 1.6% when inserting two workers.

C. Experiments with the Taillard instances

We have repeated the experiments on Taillard's instances. We do not report results for CPLEX, which was unable to solve the larger instances, but report results for LOMPEN when inserting a single worker, which was able to solve all instances up to 10 machines, and about 70% of the instances with 20 machines in 2 h. We also do not report results for the heuristic variant computing an exact schedule on the parallel stage by dynamic programming, since it turned out to be too slow and thus found solutions of inferior quality.

Tables IV and V present the results for the Taillard instances with a time variation in $[p, 2p]$ and $[p, 5p]$, respectively. They report the average relative deviations for each group of Taillard's instances of the same size (G), and each percentage of incompatibilities (I) and additionally, when inserting one worker, the solution time of LOMPEN ($\bar{t}$).

TABLE V

Results for Taillard instances with time variation $[p, 5p]$

		LOMPEN		One worker			Two workers		
G	I	$\bar{t}$	Rd.	S	P	PL	S	P	PL
1	0	0.2	106.5	106.5	106.5	106.5	6.7	5.8	5.3
2	0	3.7	58.0	58.0	58.0	58.0	5.5	4.1	3.7
3	0	390.8	24.6	24.6	24.7	24.7	3.7	2.8	2.2
4	0	6.8	149.2	149.2	149.2	149.2	5.6	2.8	2.3
5	0	30.8	125.5	125.5	125.5	125.5	9.3	5.7	4.0
6	0	491.5	77.0	77.0	77.0	77.0	7.8	6.2	3.5
Avg.		154.0	90.1	90.1	90.2	90.2	6.4	4.6	3.5
1	10	0.2	108.3	108.3	108.4	108.4	6.6	5.8	5.4
2	10	3.6	58.0	58.0	58.0	58.0	5.5	4.4	3.6
3	10	389.5	26.6	26.6	26.6	26.6	3.6	2.7	2.1
4	10	5.8	154.7	154.7	154.7	154.7	5.2	3.0	2.4
5	10	27.5	125.5	125.5	125.6	125.5	9.6	6.6	4.3
6	10	476.9	77.8	77.8	77.8	77.8	8.0	6.4	3.4
Avg.		150.6	91.8	91.8	91.8	91.8	6.4	4.8	3.6
1	20	0.1	119.1	119.1	119.1	119.1	6.6	5.8	5.4
2	20	3.4	68.2	68.2	68.2	68.2	5.4	4.3	3.7
3	20	383.6	27.6	27.6	27.6	27.6	3.9	3.4	2.6
4	20	4.4	154.7	154.7	154.7	154.7	10.9	7.9	7.0
5	20	24.1	126.0	126.0	126.0	126.0	9.4	6.6	4.8
6	20	462.1	78.5	78.5	78.5	78.5	7.9	6.4	3.5
Avg.		146.3	95.7	95.7	95.7	95.7	7.3	5.7	4.5

We first look at the performance of the methods. Comparing the results of LOMPEN and the single worker case, we find that the heuristics again find very good solutions in a short time. The exact solver now has more difficulties, in particular for a time variation of $[p, 2p]$ and runs about 3 h whereas the heuristics run for at most 1 min. For a single worker the different strategies have only a small impact on solution quality. When inserting two workers, the choice of the heuristic strategy makes a difference: using a solution pool as well as a longer time limit improves the solution quality significantly. We also can observe that a large time variation makes the instances easier to solve, since the bottleneck machine dominates the solution.

We now turn to the practical value of the solutions. As before, inserting a single worker has a visible overhead of about 12% for a small time variation. A large time variation with a single worker is unpractical, since it leads to about 90% overhead. In contrast, the disabilities of two workers can be hidden completely for a small time variation and never exceed 5% for large time variations. In summary, the insertions of WWDs is feasible for moderate time variations, independent of the percentage of incompatibilities.

VI. Concluding remarks

We have studied two scheduling problems that aim to insert workers with disabilities into flow shops, and have

proposed mathematical models, heuristic solutions and a set of realistic test instances.

Our results show that our methods find close to optimal schedules for the inclusion of WWD into flow shops. From a practical point of view, when the processing time of the WWDs is about 50% slower in average, they can be included into a flow shop with a small overhead. When we assign two workers to a parallel stage with two machines we can even achieve a reduction in the makespan.

This suggests that companies can contribute to integrate persons with disabilities in their production systems with no or only moderate losses in productivity. We hope this can lower the prejudice and help to increase the participation of persons with disabilities in the market and in society.

References

[1] World Health Organization, *World report on disability*, 2011.

[2] C. Miralles, J. P. Garcia-Sabater, C. Andrés, and M. Cardos, "Advantages of assembly lines in Sheltered Work Centres for Disabled. A case study," *Int. J. Prod. Res.*, vol. 110, no. 2, pp. 187–197, 2007.

[3] C. Miralles, J. Marin-Garcia, G. Ferrus, and A. Costa, "Or/ms tools for integrating people with disabilities into employment. a study on valencia's sheltered work centres for disabled," *Int. Trans. Oper. Res.*, vol. 17, pp. 457–473, 2010.

[4] C. Blum and C. Miralles, "On solving the assembly line worker assignment and balancing problem via beam search," *Comput. Oper. Res.*, vol. 38, no. 2, pp. 328–339, 2011.

[5] F. Araújo, A. Costa, and C. Miralles, "Two extensions for the assembly line worker assignment, and balancing problem: parallel stations and collaborative approach," *Int. J. Prod. Econ.*, vol. 140, pp. 483–495, 2012.

[6] M. C. O. Moreira, M. Ritt, A. M. Costa, and A. A. Chaves, "Simple heuristics for the assembly line worker assignment and balancing problem," *J. Heuristics*, vol. 18, no. 3, pp. 505–524, 2012.

[7] M. Vila and J. Pereira, "A branch-and-bound algorithm for assembly line worker assignment and balancing problems," *Comput. Oper. Res.*, vol. 44, pp. 105–114, 2014.

[8] L. M. Borba and M. Ritt, "A heuristic and a branch-and-bound algorithm for the assembly line worker assignment and balancing problem," *Comput. Oper. Res.*, vol. 45, pp. 87–96, May 2014, online supplement: alwabp2.herokuapp.com/instances.

[9] A. J. Benavides, M. Ritt, and C. Miralles, "Flow shop scheduling with heterogeneous workers," *Eur. J. Oper. Res.*, vol. 237, no. 2, pp. 713–720, 2014.

[10] C. N. Potts, D. B. Shmoys, and D. P. Williamson, "Permutation vs. non-permutation flow shop schedules," *Oper. Res. Lett.*, vol. 10, no. 5, pp. 281–284, 1991.

[11] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, 1979.

[12] J. Gupta and E. Stafford, "A comprehensive review and evaluation of permutation flowshop heuristics," *Eur. J. Oper. Res.*, vol. 169, no. 3, pp. 699–711, 2006.

[13] R. Ruiz and J. A. Vázquez-Rodríguez, "The hybrid flow shop scheduling problem," *Eur. J. Oper. Res.*, vol. 205, no. 1, pp. 1–18, 2010.

[14] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *Eur. J. Oper. Res.*, vol. 177, no. 3, pp. 2033–2049, 2007.

[15] Q.-K. Pan and R. Ruiz, "Local search methods for the flowshop scheduling problem with flowtime minimization," *Eur. J. Oper. Res.*, vol. 222, pp. 31–43, 2012.

[16] V. Fernandez-Viagas and J. M. Framinan, "On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem," *Comput. Oper. Res.*, vol. 45, pp. 60–67, 2014.

[17] M. Nawaz, E. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

[18] E. Taillard, "Some efficient heuristic methods for the flow shop sequencing problem," *Eur. J. Oper. Res.*, vol. 47, no. 1, pp. 65–74, 1990.

[19] J. K. Lenstra, A. H. G. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Ann. Discrete Math.*, vol. 1, pp. 343–362, 1977.

[20] J. Carlier, "Ordonnancements a contraintes disjonctives," *R.A.I.R.O. Recherche operationelle/Operations Research*, vol. 12, no. 4, pp. 333–351, 1978.

[21] E. Taillard, "Benchmarks for basic scheduling problems," *Eur. J. Oper. Res.*, vol. 64, no. 2, pp. 278–285, 1993.

[22] R. Companys and M. Mateo, "Different behaviour of a double branch-and-bound algorithm on $fm \mid prmu \mid c_{\max}$ and $fm \mid block \mid c_{\max}$ problems," *Comput. Oper. Res.*, vol. 34, no. 4, pp. 938–953, 2007.