Scheduling Jobs on Unrelated Machines with Job Splitting and Setup Resource Constraints for Weaving in Textile Manufacturing

Ioannis Mourtos, Stavros Vatikiotis, Georgios Zois

Scheduling jobs on unrelated machines with job splitting and setup resource constraints for weaving in textile manufacturing[⋆]

Ioannis Mourtos[1], Stavros Vatikiotis[1], and Georgios Zois[1]

ELTRUN Research Lab, Department of Management Science and Technology, Athens University of Economics and Business, Athens 104 34, Greece
{mourtos, stvatikiotis, georzois}@aueb.gr

Abstract. This work considers the production scheduling of the weaving process in a real-life textile industry, where a set of jobs - linked to the production of a fabric type and accompanied by a quantity - arrive over time and have to be processed (woven) by a set of parallel unrelated machines (looms) with respect to their strict deadlines (delivery dates), under the goal of makespan minimization. A number of critical job and machine properties demonstrate the challenging nature of weaving scheduling, i.e., a) job splitting: each order's quantity is allowed to be split and processed on multiple machines simultaneously, b) sequence-dependent setup times: the setup time between any two orders j and k is different than setup time between jobs k and j on the same machine and c) setup resource constraints: the number of setups that can be performed simultaneously on different machines is restricted due to a limited number of setup workers. We propose a MILP formulation that captures the entire weaving process. To handle large real instances, while also speeding up an exact solver on smaller ones, we propose two heuristics that perform job-splitting and assignment of jobs to machines either greedily or by using a relaxed version of our MILP model, respectively. We evaluate the impact of our approach on real datasets under user-imposed time limits and resources (machines, workers) availability.

Keywords: Textile · Weaving Scheduling · Integer Programming · Heuristics

1 Introduction

Increasing productivity while reducing production costs has been essential in modern textile plants in terms of business sustainability. Scheduling algorithms [3] offer a viable and effective tool to improve productivity, by optimally allocating the available resources. A typical scheduling problem in textile considers a

[⋆] This research has been supported by the EU through the FACTLOG Horizon 2020 project, grant number 869951.

set of articles/orders to be woven by a set of looms with respect to their delivery dates. Each order is linked to the production of a specific fabric type and is accompanied by a positive quantity, while the looms are unrelated, meaning that each loom operates on different speeds for different orders. The aim is to find a schedule with the minimum makespan, i.e., the time that the last executed order is finished.

Two properties that make weaving scheduling a challenging problem are job splitting (a job can be split in different machines) and sequence-dependent setup times (per pairs of jobs and per machine). In practice, the latter is justified by the fact that different fabric types require different warp chains for processing, thus imposing machine setup times (to replace the warp chain) from a few hours to a few days [14]. Both properties have been studied extensively under abstract models of various machine environments and optimisation criteria [1,13,10,8,9,4] and tackled through exact methods, approximation algorithms and metaheuristics. The weaving scheduling problem has also been well-studied and admits exact polynomial time algorithms for special cases where setup times are independent and job splitting is relaxed to preemption [14], as well as MILP models and efficient metaheuristics for the general case [5,6,15,11].

Our work is focused on the weaving scheduling of PIACENZA, a textile enterprise in north Italy that manufactures woolen fabrics for luxury clothing brands. Its production environment is a parallel weaving environment composed of multiple type of looms, operating at different speeds. Weaving scheduling in PIACENZA adopts all the above-described job and machine properties, plus setup resource constraints. Specifically, the number of setups that can be performed simultaneously on different machines is restricted due to a limited number of setup workers and daily setup time is also bounded. We should note that the seminal work of [14] signifies the addition of setup resource constraints to the standard weaving scheduling as a severe challenge.

According to our knowledge, the most relevant previous work appears in [8,9]. In [8], the authors proposed near optimal heuristics for a simplified model with identical machines, job splitting, multiple setup resources and fixed (independent) setup times per job, under the makespan minimisation objective. [9] proposes a Benders Decomposition approach and heuristics for the general case of unrelated machines, sequence-dependent setup times and multiple setup resources, again under the makespan minimisation objective. However, none of these works combine all the complex properties needed for PIACENZA's case. Interestingly, [8] referred to a case combining job-splitting, sequence-dependent setup times, unrelated machines and setup resource constraints as an open research direction.

Our contribution. In Section 2 we propose a formal definition of our scheduling problem, address its computational complexity and propose a mixed integer linear programming (MILP) formulation that captures the elaborate structure of the weaving process. To handle large real instances, we propose in Section 3 two combinatorial heuristics that differ on the way they perform job splitting and assignment to machines. We experiment with several weekly instances on both MILP (using a standard solver) and heuristics to establish the computational efficiency of our approach in Section 4. As we note, although typically the trade-off between delivery dates, available machines and setup resources allows the scheduler to deliver each job on time, due to the COVID-19 pandemic a large number of jobs arrive late on the weaving department, while others become more tight in terms of deadline. To improve resource management while avoiding a further increase of tardy jobs, we propose in Section 5 a strategy that dedicates an appropriate number of machines to samples (i.e., jobs with small quantity and tight deadlines) while allocating the rest to regular jobs (i.e., jobs with large quantity and loose deadlines).

2 Mathematical Modeling

To present our mixed integer linear program (MILP), we employ the notation of Tables 1 and 2.

Model Parameters	
J	The set of jobs (orders)
M	The set of machines (looms)
s_m	The fixed speed (in strokes/min) of machine $m \in M$
q_i	The quantity (in meters) of job $i \in J$
u_i	The number of strokes/meter for the fabric type of job i
$p_{i,m}$	The processing time of $i \in J$ on $m \in M$, $p_{i,m} = q_i \cdot u_i / s_m$
$S_{i,j,m}$	The setup time of $j \in J$ succeeding job $i \in J$ on $m \in M$
$\bar{S}_i$	The setup time of jobs processed first on each machine (1h)
L_i	A lower limit on the quantity of part of job $i \in J$ allocated to any machine (50 meters)
d_i	The deadline of job $i \in J$, i.e., a strict delivery date for i
$T_{\max}$	An upper bound on the makespan of an optimal schedule, e.g., $T_{\max} = \sum_{i \in J} \max_{m \in M}(p_{i,m} + \max_{j \in J} S_{i,j,m})$.
$\mathcal{T}$	A set of equal-length intervals $[\tau_{i-1}, \tau_i),\ 1 \le i \le T$, where $\tau_0 = 0$ and $\tau_T = T_{\max}$
$l_{i,j,m}$	The number of intervals needed to setup job j after job i on machine m
l_τ	The length of every interval $t \in \mathcal{T}$ (2h, which is the least common multiple over all setup times)
$\mathcal{D}$	A partition of $\mathcal{T}$ into subsets of consecutive time intervals q with total length equal to a working day
u_q	The allowed setup time per working day $q \in \mathcal{D}$ (50h)
R	A setup resource constraint to indicate that, at each time interval, at most R machines can be set up in parallel

Table 1: Model Parameters

(MILP) : C_{max}

s.t. :

$$\sum_{i,j \in J, i \neq j} X_{i,j,m,t} \leq 1, \qquad \forall t \in \mathcal{T}, m \in M \quad (1)$$

$$\sum_{i \in J} X'_{i,m,t} \leq 1, \qquad \forall t \in \mathcal{T}, m \in M \quad (2)$$

$$\sum_{t \in \mathcal{T}} X_{i,j,m,t} \leq 1, \qquad \forall i, j \in J, i \neq j, m \in M \quad (3)$$

$$\sum_{j \in J, t \in \mathcal{T}} X_{0,j,m,t} \leq 1, \qquad \forall m \in M \quad (4)$$

$$\sum_{m \in M} Q_{i,m} = q_i, \qquad \forall i \in J \quad (5)$$

$$L_i \cdot Y_{i,m} \leq Q_{i,m} \leq q_i \cdot Y_{i,m}, \qquad \forall i \in J, m \in M \quad (6)$$

$$Y_{i,m} = \sum_{t \in \mathcal{T}, j \in J, j \neq i} X_{i,j,m,t}, \qquad \forall i \in J, m \in M \quad (7)$$

$$Y_{j,m} = \sum_{t \in \mathcal{T}, i \in J, i \neq j} X_{i,j,m,t}, \qquad \forall j \in J, m \in M \quad (8)$$

$$\sum_{t \in \mathcal{T}} X_{i,j,m,t} + \sum_{t \in \mathcal{T}} X_{j,i,m,t} \leq 1 \qquad \forall i, j \in J, i, j \neq 0, i \neq j, m \in M \quad (9)$$

$$\sum_{i,j \in J, i \neq j, t \in \mathcal{T}} X_{i,j,m,t} = \sum_{i \in J} Y_{i,m} - 1, \qquad \forall m \in M \quad (10)$$

$$X_{i,j,m,t} + \sum_{\substack{i' \in J, i' \neq i, \\ t' \in \mathcal{T}, t' \leq t}} X_{j,i',m,t'} \leq 1, \qquad \forall i \in J, j \in J, i \neq j, m \in M, t \in \mathcal{T} \quad (11)$$

$$X_{i,j,m,t} \cdot l_{i,j,m} \leq \sum_{t'=t}^{t+l_{i,j,m}-1} X'_{j,m,t'}$$
$$\forall i, j \in J, i \neq j, m \in M, t \in \mathcal{T} \setminus \{T - r | 1 \leq r \leq l_{i,j,m}\} \quad (12)$$

$$\sum_{t \in \mathcal{T}} X'_{j,m,t} \leq \sum_{i \in J, i \neq j, t \in \mathcal{T}} l_{i,j,m} \cdot X_{i,j,m,t}, \qquad \forall j \in J, m \in M \quad (13)$$

$$\sum_{i \in J, m \in M} X'_{i,m,t} \leq R \qquad \forall t \in \mathcal{T} \quad (14)$$

$$\sum_{i \in J, m \in M, t \in q} l_\tau \cdot X'_{i,m,t} \leq u_q \qquad \forall q \in \mathcal{D} \quad (15)$$

$$C_{j,m} - C_{i,m} + V(1 - \sum_{t \in \mathcal{T}} X_{i,j,m,t}) \geq Q_{j,m} \cdot \frac{u_j}{s_m} + S_{i,j,m} \cdot \sum_{t \in \mathcal{T}} X_{i,j,m,t},$$
$$\forall i, j \in J, j \neq i, m \in M \quad (16)$$

$$C_{j,m} \geq \sum_{i \in J, i \neq j, t \in T} X_{i,j,m,t}(\tau_{t-1} + S_{i,j,m}) + Q_{j,m} \cdot \frac{u_j}{s_m}, \qquad \forall j \in J, m \in M \quad (17)$$

$$\bar{S}_i \cdot Y_{i,m} + Q_{i,m} \cdot \frac{u_i}{s_m} \leq C_{i,m} \leq C_{\max} \qquad \forall i \in J, m \in M \quad (18)$$

$$\sum_{j \in J, t \in \mathcal{T}, t > \lceil \frac{R}{M} \rceil} X_{0,j,m,t} = 0 \qquad \forall m \in M \quad (19)$$

$$Y_{i,m}, X'_{i,m,t}, X_{i,j,m,t} \in \{0,1\}, C_{i,m}, Q_{i,m} \in \mathbb{R}^+, \qquad \forall i, j \in J, m \in M, t \in \mathcal{T}$$

The overall goal is to minimise the makespan of the schedule, denoted as $C_{\max}$. Since setup times are strictly positive, it is easy to prove that each machine processes at most one part of each split job. We refer to the above problem as the *Weaving Scheduling* problem, which is *NP-hard* even if machines are identical, job setup times are fixed (and independent) and $R = 1$ [7].

Variables	
$X_{i,j,m,t}$	1 if $j \in J$ is the successor of $i \in J$ on machine $m \in M$, which is set up right after i at time $t \in \mathcal{T}$ and there are no other jobs processed between them on m, 0 otherwise
$X'_{i,m,t}$	1 if $i \in J$ is under setup on machine $m \in M$ at $t \in \mathcal{T}$, 0 otherwise
$Y_{i,m}$	1 if $i \in J$ is assigned on machine $m \in M$, 0 otherwise
$Q_{i,m} \in \mathbb{R}^+$	The quantity of job $i \in J$ to be processed by $m \in M$
$C_{i,m} \in \mathbb{R}^+$	The completion time of the part of job $i \in J$ processed on $m \in M$
$C_{max} \in \mathbb{R}^+$	The makespan of the schedule

Table 2: Decision Variables

(MILP) is partly inspired by formulations on special cases [8,13], extending them to capture the elaborate structure of *Weaving Scheduling*. More specifically, Constraints (1)-(4), (7)-(11), (19), are used to ensure the feasibility of job assignment, respecting that each machine processes at most one single part of each split job. Constraints (5)-(6) allow for job splitting wrt to the quantity limits. Constraints (12), (13), (16) ensure that the setup of each job part precedes its execution on the corresponding machine and calculate its completion time. Constraints (14), (15) are setup resource constraints, and Constraints (17), (18) provide tight lower bounds.

3 Combinatorial Heuristics

Using an exact commercial solver (Gurobi 9.1) on (MILP), we can solve many daily instances (i.e., ones with orders arriving at the same date) in a few minutes either optimally or by a small gap. Hence (MILP) could be used to support short-term goals like scheduling jobs in a daily manner. However, to fully support the business needs of a weaving enterprise, including mid and long-term goals, it is important to efficiently tackle larger real instances. In this direction, we propose two combinatorial heuristics, GH1 and GH2, which differ in the way they handle job splitting and assignment of each part of a job to a machine, while handling the sequence-dependent setup times and setup resources in the same way. Table 3 summarizes the notation used in the present and the following sections.

GH1, performs an iterative exact splitting and assignment of jobs (parts) to machines using a MILP formulation (which is a subproblem of *Weaving Scheduling* where setup resource constraints are not taken into account) that minimises makespan subject to Constraints (5), (6) (to ensure that quantity limits are satisfied), (20) that calculates a lower bound on the time needed to process the assigned part of each job on each machine and (21) that limits the number of

Notation	
GH1	The first greedy heuristic
max_assgn	Upper limit on job assignments in each iteration of GH1
GH2	The second greedy heuristic
LPT	Longest Processing Time first rule used in GH2
λ	A positive constant chosen on the assignment step of GH2
ld_m	The load of $m \in M$ on the assignment step of GH2
aTSP	The Asymmetric Traveling Salesman Problem
LB	A lower bound derived by the solution of the MILP in GH1
% Gap	The percentage gap of GH1 or GH2 wrt LB: $\frac{\{\text{GH1,GH2}\}-LB}{LB} \cdot 100$
Tardiness	For each job j in a schedule it is equal to $\{\max_{m\in M} C_{j,m} - d_j, 0\}$
Tardy job	A job $j \in J$ with positive tardiness, i.e., $\max_{m\in M} C_{j,m} > d_j$

Table 3: Algorithms and experiment parameters and abbreviations

possible job assignments to max_assgn. GH1 starts by setting the maximum possible value of max_assgn $= |J| \times |M|$ and after each iteration decreases it by 1, in order to exploit all possible exact solutions (of increased or decreased job splitting potential) choosing the best among them. It terminates when the number of jobs exceeds the possible assignments i.e., max_assgn $= |J| - 1$, as there is no possibility to assign all jobs.

$$\sum_{i\in J}(Q_{i,m}\frac{u_i}{s_m} + Y_{i,m} \cdot \min_{j\in J} S_{i,j,m}) \leq C_{max} \qquad \forall m \in M \qquad (20)$$

$$\sum_{i\in J}\sum_{m\in M} Y_{i,m} \leq \max\text{\_assgn} \qquad \forall m \in M \qquad (21)$$

On the other hand, GH2 performs a greedy job splitting dividing job quantities into parts based on the lower bound L_i: For each job i with $q_i \geq 2L_i$ we create $\alpha = \lceil \frac{q_i}{L_i} \rceil$ job parts of quantity equal to $\frac{q_i}{\alpha}$. Then, the job parts are ordered according to the LPT rule, in order to prevent the resulted schedule from unbalanced machine loads (i.e., when a job with large processing time is scheduled last). Then assignment process is similar to the one proposed in [2] for makespan minimisation on unrelated processors: For the LPT order of job parts, it assigns each part i to the machine $k = \arg\min_{m\in M}\{\lambda^{ld_m + S_{j,i,m} + Q_{i,m}\frac{u_i}{s_m}} - \lambda^{ld_m}\}$, where j is the last job executed on m before i.

Both GH1 and GH2 are then following the next two stages. Stage A: For each machine, the assigned job parts are scheduled optimally by reducing the problem to aTSP, where nodes correspond to jobs' parts and the distance between nodes to sequence-dependent setup time plus processing time of the corresponding job part; the exact approach of [12] is proved quite efficient for our instances. Stage B: For each machine in decreasing order of load and each available group of workers, we compute the earliest time that a job part can start its setup, respecting the order of job parts from Stage A. Note that, in Stage B, by starting from the most loaded machine, we significantly reduce the effect of idle intervals between consecutive job executions on the final makespan. Moreover, in the case of GH2, we do not violate the assumption that each machine processes at most one single part of each split job, as the setup time between parts of the same job is equal to zero, and thus in the aTSP solution they will be consequently ordered.

Summarising, GH1 performs an exhaustive job splitting and assignment supported by an exact solver, while GH2 computes a fast greedy assignment of all possible job parts to machines.

4 Computational experiments

The experiments are performed on 27 weekly instances, from 01/2020 - 07/2020. The number of jobs per instance ranges from 7-69, the available groups of workers and number of machines per week are $R = 3$ and 12 respectively, while setup times receive values from the set $\{2h, 4h, 6h\}$. The experiments ran on a 64-bit Windows PC (Intel i5, 2.5GHz CPU speed, 8GB RAM) using Python 3.7.2 for GH1, GH2 and GUROBI 9.1 (Python API for (MILP) and MILP of GH1).

We tested (MILP) on the above dataset, with a 2-hour limit, on 4, 6, 8 and 10 machines and it was able to solve optimally one weekly instance (7 orders) on 10 and 8 machines in 10 sec and 25 sec respectively, while the other two instances were solved with Gaps 8.62 % after 262 sec for 6 machines and 5.14% after 1735 sec for 4 machines. The difficulty of (MILP) to deal with job splitting property lies on the fact that the time horizon (thus, the number of time intervals and the number of variables) increases exponentially as the quantity of the job increases. Interestingly, the above results refer to the solution of Gurobi when using as upper bound the best among GH1 and GH2 solutions (normalizing processing times and setup times as multiples of l_τ), otherwise we could only handle some daily instances. So we proceed by applying GH1 and GH2 to solve our weekly instances. To better evaluate the performance of GH1 and GH2 we divide our dataset into five subsets of increasing number of jobs, each consisting of 5-6 weekly instances and we test each subset for different number of available machines (4,6,8,10,12).

	% Gaps of GH1					Mean	Mean	% Gaps of GH2					Mean	Mean
# Orders	4	6	8	10	12	Gap (%)	t(s)	4	6	8	10	12	Gap (%)	t(s)
$[7, 26]$	3.2	4.92	6.99	10.45	10.39	7.19	38.7	13.03	22.46	30.21	38.43	42.15	29.25	3.9
$[35, 40]$	2.2	3.21	4.2	5.67	5.99	4.25	34.9	11.07	16.94	22.91	36.8	29.82	23.51	25.3
$[43, 45]$	2.14	2.77	4.17	5.59	7.05	4.34	38.6	10.87	14.88	23.78	35.5	38.63	24.73	8.7
[46.51]	1.97	2.86	3.7	4.82	6.09	3.89	33.8	11.16	15.95	23.02	33.86	30.72	22.94	6.5
$[57, 69]$	1.76	2.54	3.37	4.39	5.62	3.54	36.1	12.98	17.96	22.7	31.22	24.05	21.78	15.9
Mean Gap	2.28	3.3	4.55	6.29	7.1	4.71	-	11.84	17.75	24.68	35.23	33.32	24.57	-
Mean t(s)	6.3	35.4	33.5	46.5	59.8	-	35	26.9	7	5.3	4	14.6	-	11.6

Table 4: Results over all weekly instances on 4, 6, 8, 10 and 12 machines

As we show in Table 4, GH1 outperforms GH2, achieving results of 4.2 times smaller gap, but being 3 times slower on average, over different numbers of available machines. Notably, GH1 achieves almost optimal solutions of Gap less than 7.1% (4.7% on average) for all instances, in less than a minute (35 sec on average). Note that instances with a few orders on many machines seem to demonstrate larger Gaps, compared to smaller number of machines mainly due to the total setup time constraint and the limited number of groups of workers. Additionally, running times may seem inconsistent regarding the size of the instances, but this is justified due to the small number of instances of each subset. As a result, instances that are time-consuming within a subset have a huge impact on the average running time.

5 Enhancements

It is important to note that, due to the COVID-19 situation, 22.13% of orders were tardy. Even though the solutions in Section 4 achieve small gaps, they cause a significant increase on the number of tardy jobs which therefore rise to 27.4 % of the total orders (an increase of 24 % compared to the ones initially tardy).

Moreover, observing that small jobs (unsplittable with $q_i < 100$ meters) have tight deadlines, while larger ones have looser, it appeared reasonable to dedicate a set of machines to small jobs and the rest to the large ones. To this direction, we perform a comparison of GH1 and GH2 on small and large jobs separately, to decide which is the best choice in every case. We divide each weekly instance to small and large jobs and, as before, we divide our dataset into subsets of increasing number of (either small or large) jobs. Subsets with small jobs consist of 4-6 weekly instances each, while subsets of large jobs of 5-7; note that on the latter we have excluded two instances, since they included only 1 and 2 jobs respectively. The size of small job instances ranges from 6 to 41, while for large from 5 to 34.

# Orders	% Gaps of GH1				% Gaps of GH2			
	4	6	8	10	4	6	8	10
$[6, 12]$	3.78	8.53	5.36	4.37	27.11	26.67	20.13	15.36
$[14, 22]$	6.64	9.55	16.69	19.84	22.66	41.52	44.42	52.07
$[25, 29]$	8.26	20.53	48.24	58.43	16.35	32.03	55.16	68.08
[30.31]	6.69	25.8	60.75	77.98	26.49	53.8	64.05	102.75
$[33, 41]$	6.41	16.33	45.5	65.08	15.13	23.86	51.83	56.02
Mean Gap	6.54	16.35	36.6	46.94	20.95	35.56	48.49	60.45
Mean t(s)	9	2.7	5.7	30.6	1.1	1.5	2	2.4

# Orders	% Gaps on GH1				% Gaps on GH2			
	4	6	8	10	4	6	8	10
$[5, 12]$	2.96	4.23	6.04	7.82	17.45	38.01	30.82	64.14
$[13, 16]$	1.66	2.46	3.4	4.71	10.14	20.66	24.61	46.55
$[18, 19]$	1.58	2.29	3.46	4.19	8.88	13.43	21.61	29.36
[20.34]	1.02	1.53	2.27	3.21	7.89	14.34	19.7	25.27
Mean Gap	1.78	2.59	3.73	4.93	11.01	21.61	24.12	41.38
Mean t(s)	9.9	358.1	208.6	300.1	125.8	51.7	13.8	11.5

Table 5: Results on small (left) and large (right) job instances for 4, 6, 8, 10 machines

We tested (MILP) on small jobs, using a simplified version (where in constraints (16)-(18) we substituted $Q_{i,m} \cdot \frac{u_i}{s_m}$ by $Y_{i,m} \cdot p_{i,m}$ while Constraints (5)-(6) were removed) on 4, 6, 8 and 10 machines, for 8 out of 27 weekly instances (from 6 to 18 orders). Notably the exact solver was able to solve optimally 20 instances in 98.03 sec on average, 10 instances were solved with mean Gap 7.36% and for 2 it was not able to obtain a solution under 1-hour limit. However, since the solutions obtained were of similar Gap with the ones of GH1, we do not present them in more detail. Table 5 presents the comparison between GH1 and GH2 on small and large job instances, respectively. GH1 achieves solutions of better quality, with 26.6% and 4.8% Gap for small and large jobs respectively, however GH2 is much faster (4 times on large and almost 6 times on small jobs). Interestingly, for small jobs the difference on their gap is significantly decreasing (from 410% on large jobs to 55%). Note that Gap values on small jobs instances are quite large, but this is due to the strict daily total setup time constraint.

Since GH1 performs better on both small and large job instances, we run it once to schedule first all small jobs to an appropriate number of machines, and re-run it consequently to schedule the large jobs on the remaining machines or (if possible) after the small jobs on their dedicated machines. More precisely, we run GH1 for each weekly instance, for 12 candidate numbers of dedicated machines ($|M| \in \{1, 2, \ldots, 12\}$) on small jobs. The aim of this approach is to examine the effect of dedicated machines on three optimisation criteria: makespan, number of tardy jobs and total tardiness.

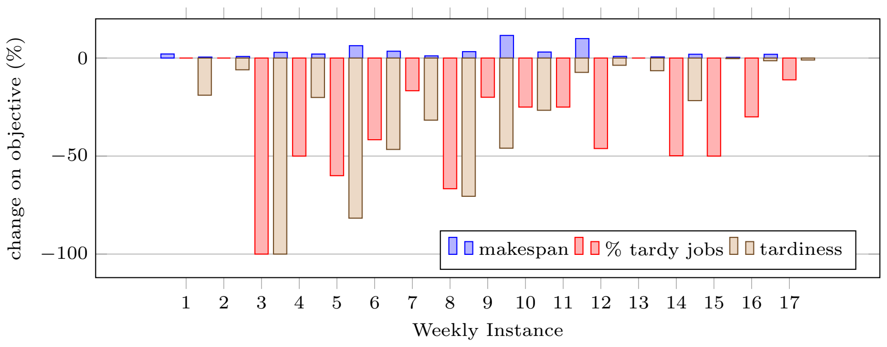

Fig. 1: Best policies to balance makespan, number of tardy jobs and total tardiness, over weekly instances.

We consider as baseline the makespan, number of tardy jobs and total tardiness over all weekly instances computed by GH1 in Section 4 and highlight the smallest average change on each criterion over the same instances, over all runs under different number of dedicated machines: For makespan, the smallest average increase is 1.55%, while for the same instances tardiness and the number of tardy jobs decrease by 16.68% and 10% respectively. For the number of tardy jobs, the largest average decrease is 16%, while for the same instances the makespan increases by 4.07 %, and tardiness decreases by 19.1%. For total tardiness, the largest average decrease is 22.62 %, while for the same instances makespan increases by 6.35% and number of tardy jobs decreases by 12.12%.

Fig. 1 presents a proposed policy for weekly instances, in order to achieve better trade offs between makespan increase and number of tardy jobs, tardiness decrease. We conclude that dedicating machines on small jobs positively affects 17 our of 27 instances (in Fig. 1), trading a small increase on makespan for large reductions on the number of tardy jobs and total tardiness. Notably all improvements occurred when the number of dedicated machines ranges from 2-7, while in 76% of the instances the range is from 2-4. It is also encouraging that on 15 of those 17 instances there were various alternative policies that could be chosen demonstrating also positive effects.

Additional experimentation, on both real and random or modified literature instances, could yield more insights. Although already competitive within a quite challenging setting, our optimisation approach could be further strengthened by examining tighter formulations in a combination with a Benders-like decomposition, to accomplish provably near-optimal solutions on even larger instances.

References

1. Allahverdi, A., Ng, C-T., Cheng, T. E. and Kovalyov, Y. A survey of scheduling problems with setup times or costs. *EJOR*, 187: 985-1032, 2008.
2. Aspnes, Y., Azar, Y. Fiat, A., Plotkin, S., and Waarts, O. On-line Routing of Virtual Circuits with Applications to Load Balancing and Machine Scheduling. *JACM* 44(3):486–504, 1997.
3. Brucker, P. Scheduling Algorithms. *Springer*, 1999.
4. Correa, J., Verdugo, V., Verschae, J. Splitting versus setup trade-offs for scheduling to minimize weighted completion time. *ORL*, 44: 469-473, 2016.
5. Eroglu, D. Y. and Ozmutlu, H. C. Solution method for a large-scale loom scheduling problem with machine eligibility and splitting property. *TJTI*, 108(12): 2154-2165, 2017.
6. Eroglu, D. Y., Ozmutlu, H. C. and Ozmutlu, S. Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent set-up times. *IJPR*, 52(19):5841-5856, 2014.
7. Letsios, D., Bradley, J. T., Suraj, G., Misener, R. and Page, N. Approximate and robust bounded job start scheduling for Royal Mail delivery offices. *JOS*, 1-22, 2021.
8. Lee, J-H., Hoon Jang, H. and Kim, H-J. Iterative job splitting algorithms for parallel machine scheduling with job splitting and setup resource constraints. *JORS*, 2020.
9. Peyro, L. F Models and an exact method for the Unrelated Parallel Machine scheduling problem with setups and resources. *ESWA*, 2020.
10. Peyro, L.F., Ruiz, R. and Perea, F. Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *COR*, 81: 173-182, 2019.
11. Pimentel, C., Alvelos, F., Duarte, A. and Carvalho, J. Exact and heuristic approaches for lot splitting and scheduling on identical parallel machine. *IJMTM*, 22(1): 39-57, 2011.
12. Roberti, R., and Toth P. Models and algorithms for the Asymmetric Traveling Salesman Problem: an experimental comparison. *EJTL*, 1:113–133, 2012.
13. Rosales, O.A., Bello, F. A. and Alvarez, A. Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *IJAMT*, 76:1705–1718, 2015.
14. Serafini, P. Scheduling jobs on several machines with job splitting property. *INFORMS J. Comp.*, 44:531-659 1996.
15. Wang, J-B. and Wang, J-J. Research on scheduling with job-dependent learning effect and convex resource-dependent processing times. *IJPR*, 53 (19): 5826-5836, 2015.