Minimizing total completion time in the two-machine no-idle no-wait flow shop problem

Federico Della Croce[1,2], Andrea Grosso[3], and Fabio Salassa[1]

[1] DIGEP, Politecnico di Torino, Torino, Italy,
{federico.dellacroce, fabio.salassa}@polito.it
[2] CNR, IEIIT, Torino, Italy
[3] D.I., Universitá degli Studi di Torino, Torino, Italy,
grosso@di.unito.it

1 Introduction

In flow shop scheduling, on the one hand, it is often required to compute so-called no-idle schedules, namely schedules where machines process the jobs continuously without inserted idle time. This occurs, for instance, when machines represent very expensive equipments and the production cost is mainly related to the actual time consumption. On the other hand, it is often required to obtain so-called no-wait schedules, namely schedules where jobs cannot be idle between the completion of an operation and the start of the following one. This latter requirement occurs, for instance, in metal-processing industries, where delays between operations interfere with the technological process, We consider here the contemporaneous presence of these two requirements by approaching the two-machine no-idle/no-wait shop scheduling problem with sum of completion times as performance criterion. Using the standard three-field notation, we deal with the $F2|no-idle, no-wait|\sum C_j$ problem. With respect to the relevant literature on flow shop, it is well known that problem $F2||C_{\max}$ is solvable in $O(nlogn)$ time [7] by first arranging the jobs with $p_{1,j} \leq p_{2,j}$ in non-decreasing order of $p_{1,j}$, followed by the remaining jobs arranged in non-increasing order of $p_{2,j}$, where $p_{i,j}$ denotes the processing time of job J_j on machine M_i. On the other hand, problem $F2||\sum C_j$ is NP-hard in the strong sense [5]. As mentioned in [1] and [10], problems $F2|no-idle|C_{\max}$ and $F2|no-wait|C_{\max}$ can also be solved in $O(nlogn)$ time. Instead, problems $F3||C_{\max}$, $F3|no-idle|C_{\max}$ and $F3|no-wait|C_{\max}$ were all shown to be NP-hard in the strong sense [5, 2, 11]. Recently, Billaut et al. [3] showed that problem $F2|no-idle, no-wait|\sum C_j$ is solvable in linear time, but problems $J2|no-idle, no-wait|\sum C_j$ and $O2|no-idle, no-wait|\sum C_j$ are NP-hard in the strong sense. In [1], it is reported that both problems $F2|no-idle|\sum C_j$ and $F2|no-wait|\sum C_j$ are NP-hard in the strong sense by exploiting the fact that the NP-hardness proof of problem $F2||\sum C_j$ in [5] was provided by constructing a flow shop instance that happened to be both no-idle and no-wait. Thus, also problem $F2|no-idle, no-wait|\sum C_j$ is NP-hard in the strong sense.

The paper proceeds as follows. In Section 2, the problem is introduced, an ILP formulation based on positional completion times variables is provided and the

no-idle no-wait requirement is expressed in a form of valid inequalities on the ILP model. Further properties determining dominant schedules in the exploration of the solutions space are also provided. Section 3 presents a matheuristic approach where a very large size neighborhood combining the above mentioned properties to the ILP formulation of the problem is proposed. Computational results are provided in Section 4.

2 Problem description and related properties

In the $F2|no-idle, no-wait|\sum C_j$ problem, a set of n jobs is available at time zero. Each job j must be processed non-preemptively on two continuously available machines M_1, M_2 with known integer processing times $p_{1,j}, p_{2,j} > 0$, respectively. Each machine is subject to the so-called no-idle constraint, namely, it processes continuously one job at a time, and operations of each job cannot overlap. Each job is also subject to the so-called no-wait constraint, namely, it cannot be idle between the completion of the first operation and the start of the second operation. All jobs are processed first on machine M_1 and next on machine M_2 and, given the no-wait constraint, the jobs sequences on the two machines must be identical. Consider Table 1 and Figure 1 which provide an illustrative example of a feasible no-idle, no-wait schedule for a 7-job problem.

i	J_1	J_2	J_3	J_4	J_5	J_6	J_7
$p_{1,i}$	4	3	4	2	3	5	6
$p_{2,i}$	3	4	2	3	5	6	4

Table 1. Processing times for a 2-machine flow shop

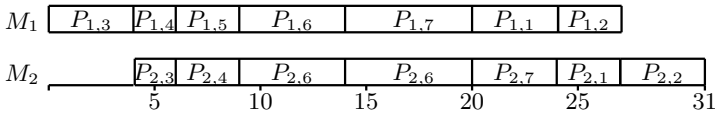

Fig. 1. A no-idle no-wait schedule for a 2-machine flow shop

We emphasize that the $no-idle, no-wait$ requirement is very strong. Indeed, for any feasible sequence $S = ([1], [2], \ldots, [n])$ consecutive jobs are forced to share common processing times in such a way that the following equalities hold.

$$p_{2,[j]} = p_{1,[j+1]} \qquad \forall j \in 1, \ldots, n-1. \tag{1}$$

Besides, the no-idle constraint on the machines imply that also the following equalities hold.

$$C_{i,[j]} + p_{i,[j+1]} = C_{i,[j+1]} \qquad \forall i = 1,2 \quad \forall j \in 1,\dots,n-1. \tag{2}$$

As a consequence, we have

$$\sum_{j=1}^{n} C_{2,[j]} = \sum_{j=1}^{n} C_{1,[j]} + \underbrace{\sum_{j=1}^{n} p_{2,j}}_{\text{constant}} \tag{3}$$

where condition (3) allows to treat the problem objective referring only to a single-machine function $f(S) = \sum_{j=1}^{n} C_{1[j]}$. Some useful properties, as well as neighborhood definitions, emerge when a sequence $S = ([1],[2],\dots,[n])$ is split in *blocks* of consecutive jobs $S = (B_0, B_1, \dots, B_t)$. For a block $B = ([k],[k+1],\dots,[k+m])$ in such a sequence we define an *entering* processing time $p_1(B) = p_{1,[k]}$, a *leaving* processing time $p_2(B) = p_{2,[k+m]}$ and a block processing time $p(B) = \sum_{j=k}^{k+m} p_{1[j]}$. Two blocks B, B' appearing in a feasible sequence $S = (\alpha, B, \pi, B', \omega)$ are then *swap-compatible* if $p_1(B) = p_1(B')$ and $p_2(B) = p_2(B')$. Then also the new sequence $S' = (\alpha, B', \pi, B, \omega)$ is feasible. If two swap-compatible blocks appear consecutively (i.e. $\pi = \emptyset$) then $p_1(B) = p_2(B) = p_1(B') = p_2(B')$. The following property holds.

Property 1 *Given a sequence $S = (\alpha, B, B', \omega)$ where two swap-compatible blocks appear consecutively, S is optimal iff*

$$\frac{p(B)}{|B|} \leq \frac{p(B')}{|B'|}.$$

Proof. Consider the two block sequences $S = (\alpha, B, B', \omega)$ and $S' = (\alpha, B', B, \omega)$. Note that $f(S') - f(S) = -|B'|p(B) + |B|p(B')$, hence $f(S') - f(S) > 0$ iff $\frac{p(B)}{|B|} < \frac{p(B')}{|B'|}$.

The latter property extends the well-known WSPT rule to block sequencing in the $2|\text{no-idle, no-wait}|\sum_j C_j$ problem. Consider now a processing time $\bar{p}$ and a sequence S; split S into a block sequence $S[\bar{p}] = (B_0, B_1, B_2, \dots, B_t, B_{t+1})$ with B_0 possibly empty and $p_1(B_1) = p_1(B_2) = \dots = p_1(B_t) = \bar{p}$. Directly from Property 1, we derive the following corollary.

Corollary 1 (block-WSPT property) *In any optimal sequence S, for all $\bar{p}$:*

$$S = S[\bar{p}] \implies \frac{p(B_1)}{|B_1|} \leq \frac{p(B_2)}{|B_2|} \leq \dots \leq \frac{p(B_t)}{|B_t|}$$

2.1 ILP formulation

We stand on the model with positional completion times variables (see [9]), since it allows (compared to other models) to better exploit the no-wait constraint.

Let C_{ki} be variables representing the completion times of the job in position i processed by machine $k = 1, 2$ and x_{ij} 0/1 decision variables, where $i, j \in \{1, \ldots, n\}$. A variable x_{ij} is equal to 1 if job i is in position j of the sequence, zero otherwise. We point out that in many flow–shop problems with regular performance criterion, solutions are characterized by a compact schedule on the first machine (i.e. there is no idle time on the first machine). In the present problem this is a requisite, thus the MIP formulation can be slightly adapted from standard 2-machine flow-shop to include this requisite depicted in (2). The problem can be then formulated as follows.

$$\min \sum_{j=1}^{n} C_{2j} \tag{4}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j = 1, \ldots, n \tag{5}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1, \ldots, n \tag{6}$$

$$C_{11} = \sum_{i=1}^{n} p_{1i} x_{i1} \tag{7}$$

$$C_{2j} = C_{1j} + \sum_{i=1}^{n} p_{2i} x_{ij} \qquad \forall j = 1, \ldots, n \tag{8}$$

$$C_{ij} = C_{i,j-1} + \sum_{i=1}^{n} p_{ij} x_{ij} \qquad \forall i = 1, 2 \quad \forall j = 2, \ldots, n \tag{9}$$

$$x_{ij} \in \{0, 1\} \tag{10}$$

where constraints (5)–(6) state that a job is chosen for each position in the sequence and each job is processed exactly once. Constraint (7) sets the completion time of the first job on M_1. Constraints (8) force for each job the start of the second operation on M_2 exactly after its preceding operation on M_1 has completed (no wait requisite). Constraints (9) impose that on each machine the schedule must be compact (no idle time as in (2)). Finally, (10) sets variables definition. As pointed out in (1), the processing times on the two machines in each position $(2, \ldots, n-1)$ must match. In other words, whenever a job with processing time p on machine M_2 is processed in position $j-1$, then the following job in position j is required to have processing time p on machine M_1. Thus, the following set of constraints can be added to tighten the above model.

$$\sum_{i:\ p_{1i}=p} x_{ij} = \sum_{i:\ p_{2i}=p} x_{i(j-1)} \qquad \forall j = 2, \ldots, n-1 \quad \forall p \tag{11}$$

Notice that this set of constraints can be easily introduced in the MIP model due the choice of positional variables, since they offer a straightforward implementation of (1). Preliminary testing indicated that the addition of the set of

constraints (11) dramatically increases the performances of any solver applied to the ILP formulation of our problem. However, only instances with up to 70 jobs are solved to optimality within reasonable CPU time. Correspondingly, the use of a matheuristic procedure for handling medium-large size instances of the problem appears very appropriate.

3 Heuristic Approach

We propose a three-step approach that first generates a starting feasible solution, then applies a neighborhood search approach exploiting the problem properties and finally executes a matheuristic step to further improve the solution.

3.1 Starting solution generation

Searching for an eulerian tour in a directed graph is a useful tool for generating feasible sequences for $F2|\text{no-idle, no-wait}|\sum_j C_j$. We represent the problem instance as a directed multigraph $G(V, A)$ as follows.

$$V = \{p_{ij} : i = 1, 2, j \in 1, 2, \ldots, n\}$$
$$A = \{(p_{1j}, p_{2j}) : j = 1, 2, \ldots, n\}.$$

Each processing time in the problem is associated to a node in G, while each job j with processing times p_{1j}, p_{2j} is associated with an arc directed from node p_{1j} to node p_{2j} — note that multiple arcs can be present. A feasible solution is represented by an eulerian tour of G; actually each eulerian tour represents $|V|$ solutions, since any job (arc) can be chosen as starting point for the sequence. Figure 2 gives an idea of the structure of G, referring to the jobs in Table 1.

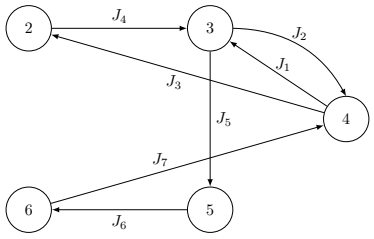

Fig. 2. Jobs of Table 1 represented as directed graph.

Sequences like $(J_5, J_6, J_7, J_8, J_9, J_1, J_2, J_3, J_4)$, $(J_6, J_7, J_8, J_9, J_1, J_2, J_3, J_4, J_5)$, $(J_7, J_8, J_9, J_1, J_2, J_3, J_4, J_5, J_6)$ can be generated from the same eulerian tour. To reach an initial feasible solution we run on G a randomized version of Hierholzer's algorithm:

– Choose randomly a starting vertex $v \in V$;
– Follow a trail of arcs from v until returning to v. The obtained tour is a closed tour, but may not cover all the nodes and arcs of the initial graph.
– As long as there exists a vertex u that belongs to the current tour but has adjacent edges not in the tour, start another trail from u, following unused arcs until returning to u and join the new tour to the previous one.

The trails are built by randomly selecting the next arc among those available. The feasible sequence starts with the job corresponding to the first selected arc.

3.2 Neighborhood exploration

Here we develop a neighborhood search procedure for $F2|\text{no-idle,no-wait}|\sum C_j$. Each time a current sequence S is available, we enforce the block-WSPT ordering property. The current sequence is decomposed into the consecutive swap-compatible blocks $S[p] = (B_0, B_1, \ldots, B_t)$ with entering time p. The blocks-WSPT sequencing is enforced by sorting such blocks in nondecreasing order of $p(B)/|B|$. Also, a post-optimization phase is performed, processing the "straddling" jobs in $B_0 \cup B_t$. The jobs in $B_0 \cup B_t$ are rotated, until they are optimally partitioned between the head and the tail of the sequence. The WSPT enforcement is repeated for every processing time p, and until no further improvement is found. Each WSPT enforcement requires $O(n \log n)$ time, given that the blocks can be identified in $O(n)$ time. The neighborhood structure considered in our local search algorithm is based on swap-compatible blocks. Given a current sequence S, we consider all possible pairs of processing times $p, \bar{p}$ and identify nonadjacent swap-compatible blocks in $S = (\alpha, B, \pi, B', \omega)$ with $p_1(B) = p_1(B') = p$, $p_2(B) = p_2(B') = \bar{p}$ and compute $S' = (\alpha, B', \pi, B, \omega)$ for all pairs of consecutive swap-compatible blocks B, B' — i.e. no other compatible block is contained in π. We chose to operate on consecutive blocks in order to save time. The sequence is decomposed in blocks in linear time. A linear number of consecutive blocks has to be considered for the swap operations, and the objective variation for each swap operation can be evaluated in constant time. Since we consider every pair of times $p, \bar{p}$ this neighborhood is searched in $O(n^3)$ time. The best neighbor S' (if $f(S') < f(S)$) is kept as current solution and the block-WSPT ordering is enforced again. The search is repeated until a local minimum is found, i.e. when no improvement is found while trying all possible swaps of swap-compatible blocks. Notice that, for any randomly detected initial solution provided by the randomized version of Hierholzer's algorithm, the application of phase 2 generates a different local minimum. Hence, it is worthy to consider a multistart approach that iteratively determines a different initial solution and then generates by means of phase 2 the corresponding local minimum.

3.3 Matheuristic step

In addition, a matheuristic step is applied taking advantage of the positional completion times variables ILP formulation. This step is applied along the lines

presented in [4] for problem $F2|no-idle, no-wait|\sum C_j$. Consider a working sequence $\bar{S}$ corresponding to a valid configuration $\bar{x} = (\bar{x}_{ij} : i, j = 1, \ldots, n)$ of the x_{ij} variables. The matheuristic neighborhood $\mathcal{N}(\bar{S}, r, h)$ can be defined by choosing a position r in the sequence and a window size parameter h. Let denote by $\bar{S}(r; h) = \{[r], [r+1], \ldots, [r+h-1]\}$ the index set of the jobs located in the consecutive positions $r, \ldots, r+h-1$ of sequence $\bar{S}$. This corresponds in the ILP formulation to add the following constraint:

$$x_{ij} = \bar{x}_{ij} \qquad i \notin \bar{S}(r; h),\ j \notin \{r, \ldots, r+h-1\}. \tag{S1}$$

The choice of the best solution in the neighborhood $\mathcal{N}(\bar{S}, r, h)$ is accomplished by keeping the sequence unchanged outside the jobs window and optimizing the sequence within the window. The resulting minimization program can then be solved by means of a MIP solver. The additional constraints (S1) state that in the new solution all jobs that do not belong to the window are fixed in the position they have in the current solution, while the window gets reoptimized. If no improved solution is found a new job-window is selected to be optimized until all possible $O(n)$ windows have been selected. The search is stopped once local optimality (no window reoptimization offers any improved solution).

4 Computational Results

4.1 Instance generation

The instances were generated in the following way:

- processing times on machine *one* were randomly extracted in $[1..max_{ptime}]$,
- $max_{ptime} = \alpha N_{jobs}$ where $\alpha \in \{0.1, 0.2, 0.3\}$,
- processing times on machine M_2 were obtained via a random permutation of the processing times of machine M_1.

For each jobs cardinality N_{jobs}, 30 instances were tested, namely 10 for each value of parameter α. For each instance the time limit was set to 600 seconds. The tests were executed on a i5 @ 2.30 GHz equipped with 8GB RAM. Table 2 provides the results reached by the proposed approach applying either phases 1,2 only (starting solution generation plus neighborhood search) or the complete heuristic with the additional matheuristic step. 5 independent runs were performed for each instance. In the matheuristic step the ILP solver was CPLEX 12.7. After a preliminary parameters calibration, the window size parameter h of the matheuristic was set to 35 jobs, while the number of repetitions of the multistart approach combining phases 1 and 2 was set to 1000. Every row provides the average over ten instances with same value of N_{jobs} and α. Columns 1,2 report the instance size and the value of max_{ptime} respectively. Columns 3,4 provide the minimal (over 5 independent runs) and average percentage error, respectively, of the neighborhood search given by phases 1,2 with respect the lower bound provided by the continuous relaxation of the enhanced ILP model (4–11). Column 5 depicts the average CPU time of the neighborhood search.

Columns 6,7,8 provide the same entries of columns 3,4,5 when the matheuristic step is added to phases 1, 2.

N_{jobs}	max_{ptime}	NS_{BEST}	NS_{AVG}	CPU_{NS}	MH_{BEST}	NH_{AVG}	CPU_{MH}
100	10	0.124	0.157	2.787	0.100	0.110	22.446
100	20	0.532	0.709	3.420	0.440	0.499	30.019
100	30	1.742	1.885	3.910	1.635	1.721	25.520
200	20	0.173	0.293	18.323	0.034	0.105	88.921
200	40	0.520	0.883	22.484	0.242	0.484	91.556
200	60	0.960	1.432	25.227	0.696	1.025	63.591
300	30	0,281	0.411	55.952	0.105	0.210	189.700
300	60	0.680	0.987	73.057	0.494	0.710	153.380
300	90	0.900	1.325	82.417	0.794	1.156	114.580

Table 2. Multistart NS vs Matheuristic

The results indicate that phases 1,2 already get very limited percentage errors from the continuous bound and that the contribution of the matheuristic step significantly improves the results at the expense of a reasonable CPU time effort with the largest size instances requiring approximately 3 minutes.

References

1. I. Adiri and D. Pohoryles. Flowshop / no-idle or no-wait scheduling to minimize the sum of completion times. *Naval Research Logistics*, 29, 495–504, 1982.
2. P. Baptiste and K.H. Lee. A branch and bound algorithm for the $F|no-idle|C_{\max}$. *Proceedings of the International Conference on Industrial Engineering and Production Management*, 1, 429–438, 1997.
3. J.C. Billaut, F. Della Croce, F. Salassa and V. T'kindt. When shop scheduling meets dominoes, eulerian and hamiltonian paths. *MAPSP17* Proceedings, Seeon-Seebruck, Germany, 20–22, 2017 (see also https://arxiv.org/abs/1707.02849).
4. F. Della Croce, A. Grosso and F. Salassa. A matheuristic approach for the two-machine total completion time flow shop problem. *Annals of Operations Research*, 213, 67-78, 2014.
5. M.R. Garey, D.S. Johnson and R. Sethi. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1, 117–129, 1976.
6. P.C. Gilmore, R.E. Gomory. Sequencing a one state variable machine: A solvable case of the traveling salesman problem. *Operations Research* 12, 655–679, 1964.
7. S.M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1, 61–68, 1954.
8. P.J. Kalczynski and J. Kamburowski. On no-wait and no-idle flow shops with makespan criterion. *European Journal of Operational Research*, 178, 677-685, 2007.
9. Lasserre J.B., Queyranne M. Generic scheduling polyhedral and a new mixed integer formulation for single machine scheduling. *in Proceedings of the IPCO Conference*, 136–149, 1992.
10. S.S. Reddi and C.V. Ramamoorthy. On the flowshop sequencing problem with no-wait in process. *Operational Research Quarterly*, 23, 323–331, 1972.
11. H. Röck. The three machine no-wait flowshop problem is NP-complete. *Journal of the Association for Computing Machinery*, 31, 336–345, 1984.