Branch-and-bound-and-memorize for the blocking permutation flowshop problem

Optimization Letters (2025) 19:1979–1996
https://doi.org/10.1007/s11590-025-02200-w

ORIGINAL PAPER

Branch-and-bound-and-memorize for the blocking permutation flowshop problem

Roberto Bargetto[1] · **Fabio Salassa**[1]

Received: 4 August 2024 / Accepted: 27 March 2025 / Published online: 12 May 2025

Abstract
Flow scheduling problems are among the most studied problems in the optimization literature and different technological constraints characterize the many variants of the general problem. This paper deals with the flowshop scheduling problem where there is no infinite buffer between subsequent machines: a job may lay on the current machine without leaving until the buffer has sufficient room. Such a situation is called blocking. In this paper, a branch-and-bound-based algorithm is proposed to optimally solve instances of the problem. The proposed method relies on a fast upper bounding procedure for early node pruning and passive node memorization that further speeds up the problem resolution. The proposed approach is compared with previous results in the literature and assessed as outperforming.

Keywords Flowshop scheduling · Blocking · Branch-and-bound · Memorization

1 Introduction

In the permutation flow shop scheduling environment, a set of jobs $N = 1, 2, ..., n$ needs to be scheduled on m machines, and each job $i \in N$ is made of m operations to be executed in the same order. Each job operation requires running continuously, i.e., without interruptions, on the relative machine, and each machine can process at most one job at a time. For each job, the operation subsequent to the current one cannot begin if the current operation is not completed.

The completion time C_i of a job $i \in N$ in any schedule S is defined as the completion time of its last operation (i.e., the operation on the last machine). The total completion time measures the sum of the completion time of each job on the last machine or, in other words, the time a job takes to completely pass through the system. The

✉ Roberto Bargetto
roberto.bargetto@polito.it

Fabio Salassa
fabio.salassa@polito.it

[1] DIGEP, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy

family of permutation flow shop scheduling problems includes a large number of possible variants of the classical problem that has been presented. Variants can concern various aspects represented by different technological characteristics of the environment itself. In particular, in this paper, the possibility that the buffer between successive machines is not infinite is taken into account. A large majority of papers on this family of problems consider, in fact, the general situation in which jobs can wait in a buffer between the end of an operation on its specific machine and the start of the next operation on the related following machine. In real-world situations, however, it is not always possible to have infinite buffers available between subsequent machines and it may happen that a job must wait on the current machine without leaving until the buffer has room enough to contain the job. This particular situation is called *blocking*. This situation differs from the *no-wait* case because, in this latter condition, there is no buffer between two subsequent machines, and the jobs must start the next operation immediately after finishing the operation on the previous machine. Following the three-field notation $\alpha|\beta|\gamma$ (see [1]), this paper deals with the $F_m|block|\sum C_j$ scheduling problem. The relevant literature includes exact, as well as heuristic approaches, aiming at finding optimal or approximate solutions respectively. The problem is known to be *NP-Hard* [2], and *APX-Hard* [1], and heuristic and metaheuristic approaches have been thus developed to solve problem large instances. A seminal work in this context is the one of [3] which deals with a problem considering only the 2-machine case whilst more recent literature deals with the more general problem with m machines. Concerning exact approaches, many works have been developed to identify tight lower bounds to be embedded in tree-based exact searches for the specific problem in an effort to optimally solve small-sized as well as medium-sized instances. The most recent results on exact approaches for the considered problem are the ones of [4] and [5]. A very recent survey dealing with the permutation blocking flow shop problem, its variants, and cataloging the relative solution approaches can be found in [6].

In the present paper, a branch-and-bound-based algorithm is proposed to optimally solve instances of the problem and the approach is compared with the results previously obtained in the literature. The proposed method relies on a fast upper bounding procedure, which is not present in previous work of the literature, and speeds up the search for high-quality solutions. Concerning lower bounds, we analyzed the ones found in the literature and we propose a counterexample to the one proposed in [5]. Moreover, we exploit *pasive node memorization* [7] (more details in Sect. 3.1) as a procedure to further improve tree-node pruning in the branch-and-bound.

The paper proceeds as follows. Section 2 provides the problem description as well as details of the proposed solution approach. Section 4 discusses in detail the outcome of the numerical experiments we ran for comparing the proposed approach and the most recent results of the literature [4, 5].

2 Problem setting

In the blocking permutation flowshop problem, the set of jobs N is to be scheduled for the given set of machines M. Each job $j \in N$ comprises $m = |M|$ ordered operations, each to be performed on a given machine $i \in M$, and the order of the operations is the same for all jobs. A processing time p_{ij} that characterizes the operation of machine $i \in M$ and job $j \in N$ is also known for all jobs. No storage buffer is assumed between successive machines in the order of operations, i.e., a job can get stuck into the machine of its current operation if the machine for the next operation is busy.

As a scheduling performance measure, we consider the total completion time, i.e., the sum of the completion time on the last machine of all jobs. This measure is also called *total flow-time* and is related to the average flow-time of jobs. Solving the problem is thus computing a permutation (an ordering) of the jobs that minimizes the total completion time.

Herein, we report an integer linear programming (ILP) formulation of the problem $F_m|block|\sum C_j$, specifically, the formulation based on the departure time of the jobs, i.e., the time jobs leave machines. Let x_{jk} be a binary variable that takes value 1 if the job j is scheduled as the k-th job; 0 otherwise. Let S_k be a continuous variable defining the starting time of the k-th job on the first machine, and D_{ki} a continuous variable defining the departure time of the k-th job leaving the machine $i \in M$. Let also $n = |N|$ be the number of feasible positions for a job in the flowshop schedule. In a feasible schedule, each job holds one, and only one, position from 1 to n. The ILP formulation reads:

$$\min \sum_{k=1}^{n} D_{km} \tag{1}$$

subject to

$$\sum_{k=1}^{n} x_{jk} = 1, \qquad \forall\, j \in N; \tag{2}$$

$$\sum_{j \in N} x_{jk} = 1, \qquad \forall\, k = 1, ..., n; \tag{3}$$

$$S_0 = 0, \tag{4}$$

$$S_k = D_{k-1,1}, \qquad \forall\, k = 2, ..., n; \tag{5}$$

$$D_{ki} \geq D_{k-1,i} + \sum_{j \in N} p_{ij} x_{jk}, \qquad \forall\, k = 2, ..., n,\ i = 1, ..., m; \tag{6}$$

$$D_{ki} \geq D_{k-1,i+1}, \qquad \forall\, k = 2, ..., n,\ i = 1, ..., m-1; \tag{7}$$

$$S_k \geq 0, \qquad \forall\, k = 1, ..., n; \tag{8}$$

$$D_{ki} \geq 0, \qquad \forall\, k = 1, ..., n, i = 1, ..., m; \tag{9}$$

$$x_{jk} \in \{0, 1\} \qquad \forall\, j \in N,\ k = 1, ..., n. \tag{10}$$

The objective function (1) minimizes the total completion time of the jobs. Constraints (2) ensure that each job is scheduled for a position in the schedule and constraints (3) ensure that only one job is scheduled for each position. Constraint (4) sets to 0 the starting time on the first machine for the job in the first position of the schedule. Constraints (5) enforce, for the first machine, that the starting time of the job in position k is equal to the departure time of the job in position $k-1$. Constraints (6) ensure that jobs do not overlap on every machine. Constraints (7) ensure the departure time of a job to be subsequent to the departure time of the previous job leaving the next machine, i.e., the job cannot leave the machine if the next machine is busy, except the case the machine is the last one. With (8), (9) and (10), the domain of problem variables is defined.

2.1 Problem lower bounds

In this section, we present, in general terms, the lower bounds that we consider for our solution algorithm. First, we present the most recent lower bound available in the literature, i.e., that of [5], then, the three lower bounds of [4].

The lower bound of [5] is based on the computation of an underestimated total completion time of each machine (i.e., a lower bound for each machine) and keeping as the problem lower bound the greatest one. The computation of the lower bound of a single machine relies on the relaxation of the blocking constraint and the optimality of the shortest processing time (SPT) rule for the total completion time of the single machine problem [8]. The general idea of the lower bound computation of [5] is similar to that of the simpler lower bound described in [4], and published earlier. Below, we formally describe the bound of [5], through formulas as they appear in the paper.

Given σ a partial sequence of s jobs, the problem lower bound is computed as

$$LB = TFT(\sigma) + \max_{1 \leq k \leq m} (LB_k) \tag{11}$$

where $TFT(\sigma)$ is the total flow time of σ (note that, as job release times are all equal to 0, the total flow time coincides, in this case, with the total completion time). LB_k is the problem lower bound for the single machine k, namely,

$$LB_k = \sum_{t=s+1}^{n} \left(ED_{tk} + \sum_{r=k+1}^{m} p_{rt} \right). \tag{12}$$

In the equation above, ED_{tk} is the lower bound for the departure from machine k of the job in position $t > s$. Given $p_{k,[j]}$ the j-th shortest processing time of jobs not in σ, in [5], ED_{tk} is defined as

$$ED_{tk} = \max\left(D_{s,k-1} + \sum_{j=s+1}^{t} p_{k-1,[j]}; D_{sk} + \sum_{j=s+1}^{t-1} p_{k,[j]}; D_{s,k+1} + \sum_{j=s+1}^{t-2} p_{k+1,[j]}\right) + p_{k,[t]}. \tag{13}$$

See [5] for the details of the formulas.

In [5], a toy instance with five jobs and three machines presents an illustrative example of the lower bound computation. In the example, a homework-like development of computations for the partial sequence ($\sigma = \{1\}$) is reported for a better understanding. Being (1, 2, 4, 3, 5) the optimal solution of the toy instance with a total completion time of 155, the lower bound of the partial sequence $\sigma = \{1\}$ must be not greater than 155 to be valid. However, the computation of the lower bound for $\sigma = \{1\}$ in [5], developed coherently with the paper formulas, results in a value of 159.

To prove our claim, in Fig. 1, we report the Gantt chart of the optimal solution $\pi^* = (1, 2, 4, 3, 5)$ of the toy instance. We also report the toy instance data, i.e., the processing times of the five jobs, in Table 1. The table reports, by rows, the processing time of each job for each machine. Based on what is depicted in Fig. 1, one can easily obtain the completion times of each job (i.e., 17, 19, 32, 40, and 47) and therefore compute the sum of completion times equal to 155. This conflicts with the lower bound calculation of 159 as it is in [5].

According to us, the problem of the bound computation is in Eq. (13) (Eq. (8) in [5]). The equation computes the maximum among three lower bound values:

1. The departure time of the i-th job from the $(k-1)$-th machine,
2. The departure time of the $(t-1)$-th job from the k-th machine,
3. The departure time of the $(t-2)$-th job from the $(k+1)$-th machine.

In [5], authors state that not all three equation terms apply for every machine k, i.e., for $k = 1$ and $k = m$, the value of terms related to machines $k-1$ and $k+1$ are null respectively. However, nothing is said about the job position, and, according to us, the third term of the max function (i.e: $D_{s,k+1} + \sum_{j=s+1}^{t-2} p_{k+1,[j]}$) must be null if $t < 3$. If the position $t-2$ does not exist for the given value of t, the max function cannot include the corresponding term. In this paper, we consequently rewrite Eq. (13) as

Table 1 toy instance data of [5]; processing time of jobs

Job	1	2	3	4	5
M1	5	6	10	5	10
M2	4	6	8	7	4
M3	8	2	5	8	6

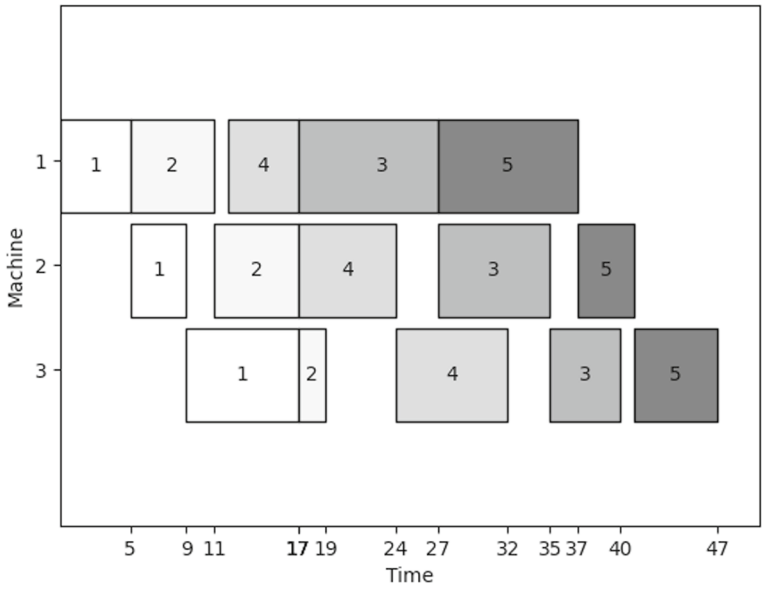

Fig. 1 Gantt chart of the toy instance optimal solution

$$ED_{tk} = \begin{cases} \text{Equation (13) applies,} & \text{if } t \geq 3, \\ \max\left(D_{s,k-1} + \sum_{j=s+1}^{t} p_{k-1,[j]}; D_{sk} + \sum_{j=s+1}^{t-1} p_{k,[j]}\right) + p_{k,[t]}, & \text{otherwise.} \end{cases} \tag{14}$$

We denote ILB0 the lower bound computed with (14) instead of (13). In Appendix A, we report the illustrative lower bound computation of [5] (with highlighted the values that we consider as wrong) and its correct computation through (14).

In [4], the authors propose three lower bounds, namely ILB, ILB1, and ILB2. The simpler lower bound ILB relies on the optimality of the SPT rule for the single machine problem, ILB1 refines ILB considering as much as possible the blocking constraint between subsequent machines, and ILB2 further improves considering the relationship among the departure time from a given machine of the i-th job and the departure time of the job from all the preceding machines. ILB2 is computationally more expensive than ILB1 and the latter is preferable according to the experiments in [4]. However, ILB2 is (on average) tighter. Below, we report formulas of ILB1, the lower bound defined in [4] that we use in this paper.

Let S be the set of scheduled jobs and $q_{j,k,l}$ the sum of processing times of the single job $j \in N$ from machine $k \in M$ to machine $l \in M$ (i.e., an underestimation of the remaining total processing time of job $j \in J$). Given σ a partial problem solution and $TCT(\sigma)$ the related total completion time, the lower bound of each machine $k \in M$ for σ is computed through

$$LB_k(\sigma) = TCT(\sigma) + \sum_{t=s+1}^{n} ED_{tk} + \sum_{j \in N \setminus S} q_{j,k+1,m} \; \forall \; k = 1, ..., m-1, \tag{15}$$

and

$$LB_m(\sigma) = TCT(\sigma) + \sum_{t=s+1}^{n} ED_{tm}. \tag{16}$$

The overall lower bound for σ is computed as

$$LB(\sigma) = \max_{1 \le i \le m} \{ LB_i(\sigma) \}. \tag{17}$$

For the last machine m, we have

$$ED_{tm}(\sigma) = D_{sm}(\sigma) + \sum_{r=1}^{t-s} p_{m,[r]}^{\{N \setminus S\}}, \; t = s+1, ..., n, \tag{18}$$

where $p_{m,[r]}^{\{N \setminus S\}}$ is the r-th smallest processing time of the unscheduled jobs $N \setminus S$ on machine m. For the other machine, it is required the computation of

$$USQ_{t,v} = \begin{cases} \{D_{s,t}(\sigma) - D_{v-1,}\}, & t = s+1, \\ \{D_{s,t}(\sigma) - D_{v-1,}\} \cup \{p_{v,[r]}^{\{N \setminus S\}} | 1 \le r \le t-s-1\}, & s+2 \le t \le n, \end{cases} \tag{19}$$

for $v = 2, ..., m$ and $t = s+1, ..., n$, and $Q_{t,v} = (Q_{t,v}(1), ..., Q(t-s))$ as the non-decreasing order of elements in $USQ_{t,v}$ for every for $v = 2, ..., m$ and $t = s+1, ..., n$. Then,

$$ED_{tk}(\sigma) = D_{sk}(\sigma) + \sum_{r=1}^{t-s} \max(p_{m,[r]}^{\{N \setminus S\}}, Q_{k+1,t}(r)), \; k = 1, ..., m-1, t = s+1, ..., n. \tag{20}$$

To obtain the final values of ED_{tk} for computing ILB1 through Eq. (15), we have to compute the values

$$ED'_{tk}(\sigma) = \max(ED'_{k+1,t-1}(\sigma), ED_{tk}(\sigma)) \; t = s+2, ..., n. \tag{21}$$

For a more detailed description of the ILB1 formulas, see [4].

3 A branch-and-bound-and-memorize algorithm

In this section, we describe the B&B &M algorithm for solving the $F_m|block|\sum C_j$ problem. B&B &M that we devised. The algorithm relies on memorization and upper bounding techniques for the early pruning of nodes related to dominated partial solutions. The problem lower bound ILB1 proposed in [4] is exploited for implementing a *best-first* B&B &M rule. To speed up the B&B &M problem resolution, we compute an initial feasible solution with the NEK-MK heuristics of [4].

3.1 Memorization

In [7], memorization techniques are shown as an effective way to speed up the resolution of sequencing problems through branching algorithms. The memorization of the not-dominated partial solution enables the algorithm to avoid the generation of large parts of the search tree. The algorithm discards the further branching of any partial solution that the memorization detects as dominated.

Our B&B &M uses the *passive node memorization* described in [7] for the Branch & Memorize paradigm. Accordingly, whenever a node is evaluated, the corresponding partial solution (i.e., a permutation of a subset of jobs) is tested for dominance against the best already found and memorized permutation of the same subset of jobs. Thus, if the currently evaluated partial solution is dominated, the related node is pruned and if the partial solution is not dominated, it is *memorized*. We highlight that in [7] only single-machine scheduling problems are considered for applying the studied memorization paradigms with computational results. In this paper, passive node memorization is applied to the resolution of $F_m|block|\sum C_j$.

The dominance rule for partial solutions we apply in our memorization scheme is the same applied in [4] to discard dominated partial solutions within a procedure that does not involve memorization. For the sake of completeness, we report the fundamental propositions of the dominance rule below.

Given σ_S and σ'_S two permutations of the same subset of jobs $S \subset N$, $C_i(\sigma_S)$ and $C_i(\sigma'_S)$ the completion time of machine $i \in M$ for σ_S and σ'_S respectively, given also $TCT(\sigma_S) = \sum_{j\in S} C_m(\sigma_S)$ and $TCT(\sigma'_S) = \sum_{j\in S} C_m(\sigma'_S)$ the total completion time of the last machine m for σ_S and σ'_S respectively, it holds as follows.

Proposition 1 σ'_S dominates σ_S if

i.) $TCT(\sigma'_S) \leq TCT(\sigma_S)$ and
ii.) $C_i(\sigma'_S) \leq C_i(\sigma_S)\ \forall\, i \in M$.

Proof See the proof of Lemma 1 in [4]. □

Proposition 1 can be restated in simpler terms as

Proposition 2 σ'_S dominates σ_S if
$TCT(\sigma_S) - TCT(\sigma'_S) \geq (|N| - |S|) \cdot \max_{i\in M} \left\{ C_i(\sigma'_S) - C_i(\sigma_S) \right\}$

Proof See the proof of Lemma 2 in [4]. □

Given $N \setminus S$ the complement of S, i.e., the set of all unscheduled jobs in N, and $\sigma^*_{N\setminus S}$ the best permutation of $N \setminus S$, according to Proposition 1, the best schedule $\sigma_S\sigma^*_{N\setminus S}$ cannot be strictly better than $\sigma'_S\sigma^*_{N\setminus S}$. The search subtree rooted at σ_S is thus not explored because a better or equivalent solution is in the subtree rooted at σ'_S.

In more general terms, Proposition 1 defines a dominated partial solution as a permutation of a subset of jobs that, if completed with any permutation of the remaining jobs, does not provide a better solution than the best one obtained by completing the dominating partial solution. Thus, when searching for the optimal solution, any partial solution that is dominated by another one permuting the same subset of jobs must be discarded, i.e., the corresponding node of the search tree must be pruned. An illustrative example of memorization node pruning follows.

We consider an example instance with three jobs and three machines to show how passive node memorization impacts the search tree. Job 1 and 2 are those in Table 1. Job 3 has processing times for machines M1, M2, and M3, respectively 5, 8 and 5. Figure 2 shows the search tree of the considered instance as it is if the branching rule is breath-first/rightmost-first (forward branching), no lower bound is applied, and nodes are pruned exclusively applying the passive node memorization paradigm. In Fig. 2, each node is labeled with a letter and reports the job scheduled for the corresponding position, e.g., the path from the root node to leaf *o* corresponds to the ordered schedule of jobs (1, 2, 3). The dashed nodes are those that are never opened and the gray nodes correspond to dominated (partial) solutions. The search tree includes six partial solutions: (1, 2), (1, 3), (2, 1), (2, 3), (3, 1), and (3, 2).

According to the branching rule, the first partial solution that the algorithm computes is (3, 2). The algorithm memorizes the partial solution (3, 2) without any test as that is the first solution it finds for the job set $\{2, 3\}$. The algorithm memorizes, for the job set, the total completion time and the completion time on the three machines of the last job of the sequence (i.e., job 2). These four values are 11, 19, 21, and 39, respectively. The next partial solution the algorithm finds is (3, 1). As there is no partial solution yet memorized for the job set $\{1, 3\}$, the algorithm memorizes for the job set the total completion time and the latest completion time of each machine. Then the algorithm evaluates the node *f* and finds a new partial solution for the job set $\{2, 3\}$, i.e., the partial solution (2, 3) with total completion time 39 and

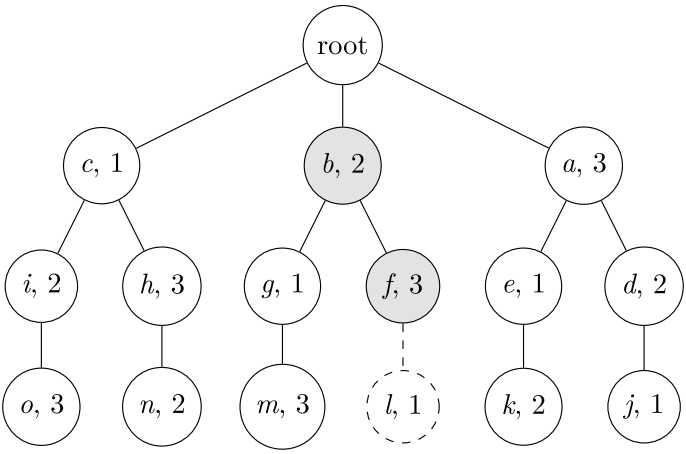

Fig. 2 Search tree example of passive node memorization with three jobs

latest completion times of the three machines respectively 11, 20, and 25. According to Proposition 1, the partial solution (3, 2) dominates the partial solution (2, 3) and, given the values previously memorized by the algorithm for the partial solution (3, 2), no child node is generated for the node *f*, i.e., node *l* is pruned. The exploration of the search tree proceeds according to the lexicographic order of the node labels. There is no partial solution dominated other than (2, 3), and the optimal solution corresponds to the leaf *n* with a completion time of 66.

For the problem $F_m|block|\sum C_j$, computing the objective value of a solution σ_S has a worst-case complexity that is linear in the number of tasks of all jobs, i.e., $O(mn)$. The complexity of memorization techniques depends on the implementation and, for efficient implementations, the operations of insertion and lookup have average and worst-case complexity respectively constant and linear in the number of memorized elements. For the $F_m|block|\sum C_j$ problem, this is the number of not-dominated partial permutations. The worst-case for the number of memorized elements is $O(n!)$, which is much higher than $O(mn)$. The worst-case complexity of searching a memorized permutation is thus $O(n!)$, equal to the worst-case number of permutations to evaluate for solving $F_m|block|\sum C_j$. However, given the constant average complexity of insertion and lookup operations, using passive node memorization is computationally cheap in practice and can hardly worsen the practical complexity of a B&B algorithm. The constant average complexity of insertion and lookup operations is enabled by hashing techniques, which efficiently implement the data structure (commonly referred to as a table or map) to store not-dominated permutations for each visited job set. In such a table, job sets serve as the keys, while permutation data are the indexed values.

3.2 Upper bounding

For general B&B algorithms, the availability of high-quality problem solutions usually enables the aggressive pruning of B&B nodes. Embedding in the B&B search the computation of high-quality heuristic solutions provides thus a strategy for leveraging the standard rule of node pruning, i.e.: nodes with a lower bound worse than, or equal to, the current best integer solution are pruned. In this paper, we stick to this approach and devise a heuristic algorithm for computing complete problem solutions through the iterations of the B&B &M.

Our B&B &M computes an initial problem solution—an upper bound (UB)—by means of the NEK-MK heuristics of [4]. In general terms, this heuristic initially computes the total processing time of each job (i.e., the sum of the processing times of all machines) and sorts the jobs according to a non-decreasing order of total processing time. Then, through the iterated operations of job selection and job reinsertion, a complete problem solution is computed. A pair-wise local search to possibly improve the obtained solution is finally performed. The worst-case complexity of the NEH-MK heuristics is $O(mn^4)$; see [4] for details.

Searching for the optimal solution, our B&B &M computes a complete problem solution for each active node (i.e., not pruned or dominated) and updates the problem UB any time a better problem solution is found. The algorithm can thus update

the problem UB more frequently than only when evaluates a leaf of the search tree. We highlight that any time the B&B &M improves the current problem UB, the search space reduces. The active nodes with a related LB that is worse or equal to the current problem UB will be pruned when evaluated. The devised heuristic algorithm operates as described below.

Given the partial solution $\bar{\sigma}_S$ related to the current B&B &M active node and σ^*_N the current best solution, the algorithm operates as follows.

Step 1. Retrieve from memory the last memorized permutation of jobs S, let σ'_S be this permutation;

Step 2. Remove jobs S from σ^*_N to obtain the permutation $\sigma^*_{N\setminus S}$ for the jobs $N \setminus S$;

Step 3. Perform a pair-wised local search of improvement on $\sigma^*_{N\setminus S}$ to obtain $\sigma'_{N\setminus S}$;

Step 4. Compute the current node UB' as the objective function value for permutation $\sigma'_S\sigma'_{N\setminus S}$.

As for any B&B that solves a minimization problem, anytime the algorithm finds an UB' better than the current best objective function value, the current best solution is updated to $\sigma'_S\sigma'_{N\setminus S}$; i.e., the current problem UB is set equal to UB'.

The steps of the upper bounding procedure with higher worst-case complexity are Step 1. (memorized permutation lookup) and Step 3. (pair-wise local search); these are $O(n!)$ and $O(n^2)$ respectively. The procedure has thus worst-case complexity $O(n!)$, and a much lower average complexity because: (*i*) the memory lookup has constant average complexity and (*ii*) the pair-wise local search concerns only the partial permutation $\sigma^*_{N\setminus S}$ and the number of pair-wise comparisons decreases with the length of σ'_S. The average complexity is thus much lower than n^2. The calculation of an upper bound at each node is convenient because it enables the algorithm to find high-quality problem solutions very soon and prune open nodes earlier.

As highlighted in Sect. 3.1, constant-time lookup operations are achievable due to the efficient implementation of hash tables, which map not-dominated permutations to their corresponding job sets.

3.3 Branching scheme

In this section, we describe how the algorithm explores the solution space. Each search tree node corresponds to a permutation σ_S of a given subset S of jobs, i.e., a partial problem solution. The algorithm builds the permutations of jobs through forward branching. Given σ_S the permutation of the currently evaluated node, the algorithm branches selecting a job $j \in \{ N \setminus S \}$ for the next position $|S| + 1$ and creates a new node for each job in $N \setminus S$. All newly created partial solutions are straightforwardly tested for dominance against the memorized permutations and eventually discarded if dominated. For each node not dominated, the algorithm computes the

lower bound $LB(\sigma_{S\cup\{j\}})$ and prunes the node if not better than the current best solution. If a node is not pruned, the algorithm computes an upper bound $UB(\sigma_{S\cup\{j\}})$ as described in Sect. 3.2, updates eventually the best problem solution, and adds finally the node to the queue. The algorithm processes the queue of nodes applying the *best-first* rule, i.e., the nodes with the lowest lower bound are evaluated first. The choice of the best-first rule is based on preliminary numerical tests demonstrating that it performs better than depth-first and breadth-first rules.

4 Computational results and discussion

In this section, we report the comparison through numerical experiments of our B&B &M with the B&B of [4] and the B&B using the lower bound of [5] corrected by us. For our experiments, we consider the best version of the B&B proposed in [4], i.e., the algorithm that uses lower bound ILB1 and includes dominance for node pruning denoted as Procedure 2. In [5], not many details are provided about the implemented B&B scheme and a standard depth-first branching scheme is described. For a fair comparison, the lower bound of [5] with our correction is evaluated using it as a lower bound procedure alternative to ILB1 in the same B&B scheme of [4] that applies the node dominance Procedure 2. We implemented the three algorithms in C++ and ran all the experiments on a desktop PC equipped with processor *Intel(R) Core(TM) i5-8500 3.00GHz*, 16GB of RAM, and running Ubuntu 20.04 LTS. All the experiments are run as single-thread processes.

We compare the algorithms on a set of instances we generated using Taillard's procedure [9] and the smaller Taillard's instances [9], i.e., those with 20 jobs and 5, 10, or 20, machines. For the instances that we generated, the number of jobs n is in $\{18, 20, 22, 24, 26, 28, 30\}$ and the number of machine m in $\{2, 3, 4, 5, 7, 10\}$. Job processing times are drawn from the integer uniform distribution $U(1, 99)$. For each pair of values (n, m), we generated 10 random instances. We generate a total of 420 instances. All the instances used in this paper are publicly available. A link to download the dataset is available at the end of the paper in the data availability statement.

In Table 2, we report the computational results in terms of number of instances solved to optimality (column *Opt.*), seconds of computation time (column *Time*), and optimality gap (column *Gap*). Each set of instances is identified through the number of jobs (column $|J|$) and the number of machines (column $|M|$). Results for Taillard's instances are denoted with a capital "T" preceding the number of jobs in column $|J|$. the B&B algorithms solved many of the smaller instances by evaluating all the generated nodes, so the gap of the optimal solution with the best bound is not 0 albeit all the instances of the set are solved to optimality; this data provides, however, an evaluation of the lower bound quality.

The numerical outcomes reported in Table 2 show clearly that the algorithm we propose outperforms the other two compared. According to Table 2, the B&B algorithm using the corrected lower bound of [5] solves to optimality 164 instances over 450 (the instances that we generated plus the 30 Taillard's instances), the best B&B of [4] 182 instances, and our B&B &M 312 instances. Results for Taillard's instances are in line with that of the instances that we generated.

Table 2 Comparison of the three B&B algorithms

$\|J\|$	$\|M\|$	ILB0			Moslehi et al. [4]			B&B with Memorization		
		Opt	Time [s]	Gap [%]	Opt	Time [s]	Gap [%]	Opt	Time [s]	Gap [%]
18	2	10	2.1	10.1	10	0.7	9.1	10	0.2	1.4
18	3	10	12.4	17.7	10	8.2	15.9	10	1.5	4.9
18	4	10	28.3	16.9	10	17.0	15.1	10	3.4	4.8
18	5	10	124.9	23.4	10	101.6	21.5	10	11.7	6.2
18	7	10	252.0	24.3	10	227.3	23.3	10	25.2	8.7
18	10	9	1553.1	24.4	9	1407.4	23.3	10	107.2	11.1
20	2	10	54.1	13.5	10	13.3	10.0	10	1.9	1.5
20	3	9	435.3	16.6	9	407.3	15.6	10	15.7	3.4
20	4	10	491.8	19.4	10	219.1	17.3	10	17.8	0.5
20	5	9	1459.2	18.5	9	1004.5	17.3	10	71.0	9.4
T20	5	8	1628.3	21.6	8	1283.2	20.8	10	94.7	12.5
20	7	2	3493.6	23.5	4	3171.4	22.1	10	176.9	11.2
20	10	1	3447.7	23.2	1	3423.1	22.1	10	637.6	12.2
T20	10	2	3349.6	26.5	2	3291.4	25.3	10	535.3	15.2
T20	20	0	3600.0	23.8	0	3600.0	23.2	1	3389.4	19.0
22	2	10	427.3	11.5	10	135.2	9.6	10	7.6	9.3
22	3	9	1233.4	18.5	10	632.9	16.0	10	37.2	12.0
22	4	7	2064.4	20.5	9	1217.1	18.3	10	83.1	15.1
22	5	2	3109.5	19.2	3	2947.5	18.0	10	245.8	12.0
22	7	0	3600.0	22.8	1	3575.0	21.7	9	1631.9	16.9
22	10	0	3600.0	28.4	0	3600.0	27.3	4	2829.1	20.3
24	2	6	1748.2	13.8	9	1039.8	11.5	10	30.2	10.7
24	3	3	3048.8	20.3	3	2782.4	18.7	10	321.5	15.4
24	4	0	3600.0	25.1	0	3600.0	22.4	10	684.3	17.9
24	5	0	3600.0	26.2	0	3600.0	25.0	7	2265.0	21.2
24	7	0	3600.0	27.7	0	3600.0	26.5	4	3180.7	20.3
24	10	0	3600.0	26.0	0	3600.0	25.1	0	3600.0	19.0
26	2	2	3162.2	12.7	5	2559.7	10.7	10	69.9	9.9
26	3	1	3287.1	19.1	3	2928.6	16.8	10	632.4	13.8
26	4	0	3600.0	24.8	0	3600.0	23.4	6	2498.3	17.2
26	5	0	3600.0	25.3	0	3600.0	23.6	1	3482.6	18.6
26	7	0	3600.0	30.7	0	3600.0	29.3	0	3600.0	21.2
26	10	0	3600.0	28.2	0	3600.0	27.5	0	3600.0	21.6
28	2	2	3036.2	14.5	4	2690.2	11.6	10	464.1	9.8
28	3	1	3369.5	20.4	2	3402.7	18.3	5	2180.4	14.9
28	4	0	3600.0	23.5	0	3600.0	21.7	3	3004.9	17.1
28	5	0	3600.0	28.6	0	3600.0	27.7	0	3600.0	20.9
28	7	0	3600.0	31.4	0	3600.0	30.2	0	3600.0	23.7
28	10	0	3600.0	30.0	0	3600.0	29.4	0	3600.0	21.4
30	2	1	3448.4	16.8	1	3310.0	13.7	8	1734.8	11.1
30	3	0	3600.0	22.5	0	3600.0	20.3	3	3206.1	15.8

Table 2 (continued)

|J|	|M|	ILB0			Moslehi et al. [4]			B&B with Memorization		
		Opt	Time [s]	Gap [%]	Opt	Time [s]	Gap [%]	Opt	Time [s]	Gap [%]
30	4	0	3600.0	28.6	0	3600.0	26.7	1	3577.6	19.5
30	5	0	3600.0	28.0	0	3600.0	26.6	0	3600.0	20.9
30	7	0	3600.0	32.6	0	3600.0	31.4	0	3600.0	24.7
30	10	0	3600.0	33.0	0	3600.0	32.3	0	3600.0	25.9

Our B&B &M is at least one order of magnitude faster than the other two if there are few machines (2 or 3 machines). The difference between the computation times grows to two orders of magnitude if there are more than three machines. Moreover, the optimality gap that the B&B &M delivers is significantly smaller than that of the other two algorithms. As the B&B &M uses ILB1 of [4], such an improvement is due to memorization and branching scheme. The applied memorization techniques and branching scheme speed up the problem resolution to a great extent.

In Table 3, we report, for the three algorithms and each Taillard's instance with 20 jobs: the problem lower bound available when the optimization is over (column *LB*), the best upper bound (column *UB*), and the number of created nodes (column *Nodes*). We recall the time limit is 3600 s (1 h). In the table, the objective function value of instances solved to optimality is in bold, and italic font is used for best-known solutions (though not proven optimal).

According to Table 3, if no memorization is applied, no upper bounding procedure is used and the branching rule is depth-first, no matter what lower bound procedure is used (that of [4] or that of [5] corrected by us), the instances solved to optimality are the same. Comparing the lower bounds and the number of created nodes, a few considerations can be added to the discussion. The lower bound of [4] appears preferable to that of [5], as fewer nodes are created in the depth-first B&B scheme of [4]. Table 3 shows that by applying memorization and best-first branching scheme, much fewer nodes are created.

We finally report a comparison of memorization, upper bounding, and lower bounding, for what concerns node pruning and time efficiency. For this purpose, we define R_{nodes} as the ratio of nodes pruned by memorization over those pruned by the lower bounding procedure. We limit the analysis to Taillard's instances with 20 jobs.

Table 4 reports, for each number of machines, the average values of the lower bounding procedure total time, the upper bounding procedure total time, the memorization total time, and the value of R_{nodes}. Computation times are reported as percentage values of the total algorithm time. Note that the three time percentages do not sum to 100%, the remaining time is spent managing the search tree (creating/pruning nodes, etc.) and computing the initial solution.

Table 4 highlights that, for the hardest instances of the considered set (i.e., those with 20 machines), memorization prunes on average up to more than five times the nodes that the upper bounding procedure prunes; values of R_{nodes}. Moreover, memorization takes a time that is much lower than that of the lower bounding procedure.

Table 3 Comparison of the three algorithms over Taillard's instances with 20 jobs and 5, 10, or 20, machines

$\|M\|$	I	ILB0			Moslehi et al. [4]			B&B with Memorization		
		LB	UB	Nodes	LB	UB	Nodes	LB	UB	Nodes
5	0	12667	**14953**	102922133	12694	**14953**	50644110	14953	**14953**	2983425
5	1	12613	16568	546817258	12933	**16343**	402489048	12987	**16343**	30345863
5	2	10938	14628	514265200	10979	14628	371515934	11201	**14297**	14363732
5	3	13255	**16483**	114353277	13405	**16483**	67733507	16483	**16483**	6049853
5	4	12650	**14212**	52161624	12650	**14212**	36977968	14212	**14212**	2470477
5	5	11276	**14624**	31710046	11276	**14624**	14560383	11713	**14624**	1711073
5	6	10307	**14936**	160059247	10783	**14936**	38355383	11978	**14936**	4456713
5	7	12287	**15193**	145190043	12344	**15193**	78915537	15193	**15193**	8081375
5	8	11874	**15544**	266963426	11908	**15544**	147393317	12151	**15544**	13597728
5	9	10909	**14392**	353781980	10920	**14392**	137749275	11359	**14392**	10767913
10	0	17154	23133	278281741	17562	23133	209275094	22358	**22358**	63726657
10	1	18261	24956	257433938	18982	25037	192291202	19769	**23881**	70509264
10	2	16613	**20873**	234411543	16670	**20873**	173623712	16944	**20873**	15578520
10	3	14286	20807	253996876	14490	20807	187691931	19916	**19916**	70892265
10	4	13506	21021	247058701	13959	21021	181470377	14927	**20196**	21417035
10	5	15151	20239	238007029	15151	**20126**	175849307	15438	**20126**	21923813
10	6	15357	**19471**	150571765	15536	**19471**	98780284	15609	**19471**	9290614
10	7	16342	21644	231126921	16342	21644	168400757	16840	**21330**	16741987
10	8	14829	**21585**	154158255	15197	**21585**	97943229	17309	**21585**	9789760
10	9	17880	23087	246040534	18106	23087	175549975	21203	**22582**	21051773
20	0	28700	35889	134385508	28722	35889	94986757	28926	*34831*	74493418
20	1	24128	34061	117393727	24332	34042	84807865	26217	*33136*	68229878
20	2	26833	*34973*	130197068	26842	*34973*	93130510	27541	35025	55642993
20	3	25835	33760	123959207	26204	33760	92249779	27004	**33006**	35869380
20	4	28898	35928	146884859	29600	35932	104927627	29742	*35464*	57251569
20	5	26065	34525	126499382	26065	34525	90345593	27226	*33881*	58898957
20	6	26558	34769	143897764	26558	34769	109199364	27652	*34112*	63546879
20	7	26001	33570	122494363	26333	33570	88911747	27240	*33459*	69135438
20	8	25980	36538	133855039	26260	36538	95618895	28156	*35015*	63408645
20	9	26506	34413	113634822	26765	34413	83466237	26820	*33288*	67167444

Table 4 Taillard's instances with 20 jobs and 5, 10, or 20, machines

$\|M\|$	LB time [%]	UB time [%]	Mem. time [%]	R_{nodes}
5	56.5	17.3	7.9	1.3
10	70.6	10.3	7.7	1.9
20	74.0	9.2	11.9	5.5

On average, a fifth of the time of the lower bounding procedure. Lastly, Table 4 shows that the upper bounding procedure takes, on average, much less time than the lower bounding procedure. The procedure of Sect. 3.2 is thus efficient despite the theoretical worst-case complexity.

In conclusion, we can assess that for solving the $F_m|block|\sum C_j$ scheduling problem, using memorization and applying the upper bounding procedure at each node, improves dramatically the performance of the B&B algorithm, especially when the number of machines is large. Moreover, results show that the best-first branching rule is preferable to depth-first.

Appendix A Correction of the example from Nagano et al. [5]

In this appendix, we report the homework-like development, as it is in [5], that applies equations Eq. (8), Eq. (7) and Eq. (8) of [5], to the toy instance given the partial sequence $\sigma = \{1\}$. The computation results we consider as wrong are in bold characters. The toy instance data are those in Table 1 of the paper main text.

In [5], the lower bound of *Machine 1* is computed as

$$ED_{2,1} = \max(5, \mathbf{9}) + 5 = \mathbf{14}, \tag{A1}$$

$$ED_{3,1} = \max(10, 9) + 6 = 16, \tag{A2}$$

$$ED_{4,1} = \max(16, 13) + 10 = 26, \tag{A3}$$

$$ED_{5,1} = \max(26, 19) + 10 = 36, \tag{A4}$$

$$LB_1 = \left(\sum_{t=s+1}^{n} ED_{tk} + \sum_{r=2}^{m} p_{tr} \right) = \mathbf{92} + 46 = \mathbf{138}. \tag{A5}$$

Equations (A1)–(A4) apply Eq. (7) of [5]. Equation (A5) applies Eq. (8) of [5], and computes the lower bound for the first machine. The lower bound of *Machine 2* is computed as

$$ED_{2,2} = \max(10, 9, \mathbf{17}) + 4 = \mathbf{21}, \tag{A6}$$

$$ED_{3,2} = \max(16, 13, 17) + 6 = 23, \tag{A7}$$

$$ED_{4,2} = \max(26, 19, 19) + 7 = 33, \tag{A8}$$

$$ED_{5,2} = \max(36, 26, 24) + 8 = 44, \tag{A9}$$

$$LB_2 = \left(\sum_{t=s+1}^{n} ED_{tk} + \sum_{r=3}^{m} p_{tr} \right) = \mathbf{121} + 21 = \mathbf{142}. \tag{A10}$$

Equations (A6)–(A9) apply Eq. (7) of [5]. Equation (A10) applies Eq. (8) of [5], and computes the lower bound for the second machine. The lower bound of *Machine 3* is computed as

$$ED_{2,3} = \max(13, 17) + 2 = 19, \tag{A11}$$

$$ED_{3,3} = \max(19, 19) + 5 = 24, \tag{A12}$$

$$ED_{4,3} = \max(26, 24) + 6 = 32, \tag{A13}$$

$$ED_{5,3} = \max(30, 34) + 8 = 42, \tag{A14}$$

$$LB_3 = \sum_{t=s+1}^{n} ED_{tk} = 117. \tag{A15}$$

Equations (A11)–(A14) apply Eq. (7) of [5]. Equation (A15) applies Eq. (8) of [5], and computes the lower bound for the third machine. The overall lower bound is computed in [5] applying Eq. (6) of that paper as

$$LB = TFT(\sigma) + \max_{1 \le k \le m} (LB_k) = 17 + \max(138, \mathbf{142}, 117) = \mathbf{159}. \tag{A16}$$

Below, we provide our correction of the lower bound computation using formulas we provide in the paper main.

We consider the reported computation as wrong since the optimal solution of the toy instance starts with the partial sequence $\{1\}$ and its optimal objective function value is 155. The optimal sequence of the toy instance is $\pi^* = (1, 2, 4, 3, 5)$. According to us, the problem is in the computation of *Machine 1* and *Machine 2* lower bounds, more precisely in formulas (A1) and (A6). In (A1), the third term of the function max equals value 9, i.e., the departure time from the second machine of job 0, but the job 0 does not exist. The third term in the max function must be thus null and $ED_{2,1} = \max(5) + 5 = 10$ consequently. This is obtained applying (14) (the correct formulation we provide in this paper) instead of (13). In turn, $LB_1 = 88 + 46 = 134$. In (A6), the third term of the function max equals value 17, i.e., the departure time from the third machine of job 0, but the job 0 does not exist. The third term in the max function must be thus null and $ED_{2,2} = \max(10, 9) + 4 = 14$ consequently. This is applying (14) instead of (13). In turn, $LB_2 = 114 + 21 = 135$. The correct problem lower bound is therefore $LB = 17 + \max(134, 135, 117) = 152$.

Funding Open access funding provided by Politecnico di Torino within the CRUI-CARE Agreement.

Data availability All the flowshop instances used in this study are available at https://doi.org/10.6084/m9.figshare.26485930.

References

1. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.H.G.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discret. Math. **5**, 287–326 (1979)
2. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Math. Op. Res. **1**(2), 117–129 (1976)
3. Reddi, S., Ramamoorthy, C.: On the flow-shop sequencing problem with no wait in process. J. Op. Res. Soc. **23**, 323–331 (1972)
4. Moslehi, G., Khorasanian, D.: Optimizing blocking flow shop scheduling problem with total completion time criterion. Comput. Op. Res. **40**(7), 1874–1883 (2013)
5. Nagano, M.S., Robazzi, J.V.S., Tomazella, C.P.: An improved lower bound for the blocking permutation flow shop with total completion time criterion. Comput. Ind. Eng. **146**, 106511 (2020)
6. Miyata, H.H., Nagano, M.S.: The blocking flow shop scheduling problem: a comprehensive and conceptual review. Exp. Syst. Appl. **137**, 130–156 (2019)
7. Shang, L., t'Kindt, V., Della Croce, F.: Branch & memorize exact algorithms for sequencing problems: efficient embedding of memorization into search trees. Comput. Op. Res. **128**, 105171 (2021)
8. Pinedo, M.L.: Scheduling. Springer, New York (2012)
9. Taillard, E.: Benchmarks for basic scheduling problems. Eur. J. Op. Res. **64**(2), 278–285 (1993)