CEJOR (2019) 27:15–38
https://doi.org/10.1007/s10100-017-0488-5

ORIGINAL PAPER

Improved bounded dynamic programming algorithm for solving the blocking flow shop problem

Ansis Ozolins[1]

Published online: 14 September 2017

Abstract In this paper, the blocking flow shop problem is considered. An exact algorithm for solving the blocking flow shop problem is developed by means of the bounded dynamic programming approach. The proposed algorithm is tested on several well-known benchmark instances. Computational results show that the presented algorithm outperforms all the state-of-the-art exact algorithms known to the author. Additionally, the optimality is proven for 26 previously open instances. Furthermore, we provide new lower bounds for several benchmark problem sets of Taillard requiring a relatively short CPU time.

Keywords Scheduling · Blocking flow shop · Bounded dynamic programming · Exact method

1 Introduction

The permutation flow shop problem is a special case of the flow shop problem where n jobs have to be processed on m machines. The objective is to minimize the time at which all jobs are completed. If there does not exist any storage space between machines, then the problem is called the blocking flow shop problem denoted by $Fm|block|C_{\max}$. This problem will be considered in the present paper. For the two machine case, Reddi and Ramamoorthy (1972) presented a polynomial algorithm which gives an exact solution. However, for $m > 2$, the $Fm|block|C_{\max}|$ problem is strongly NP-hard as it is shown by Hall and Sriskandarajah (1996) using a result from Papadimitriou and Kanellakis (1980).

✉ Ansis Ozolins
ansis.ozolins1989@gmail.com

[1] University of Latvia, Zellu Street 25, Riga LV-1002, Latvia

The majority of research is focussed on developing the heuristics. We will highlight only the most important papers. Grabowski and Pempera (2007) developed a tabu search method for the $Fm|block|C_{\max}$ problem. However, there were not done further investigations of the tabu search approach. Wang et al. (2010) proposed a novel hybrid discrete differential evolution algorithm. Ribas et al. (2011) presented an iterated greedy (IG) algorithm. Lin and Ying (2013) developed a revised artificial immune system algorithm (AIS) that is a stochastic computational technique. This technique combines the features of the artificial immune system with the annealing process of simulated annealing algorithms. Pan et al. (2013) developed a memetic algorithm. By means of this algorithm, 75 out of 120 best-known solutions for Taillard benchmark instances were improved. Zhang et al. (2015) proposed an algorithm that combines the variable neighborhood and simulated annealing approach. Tasgetiren et al. (2015) developed an extremely effective populated local search algorithm. The idea is to learn some unknown parameters through a differential evolution based on the work of Tasgetiren et al. (2013). Ultimately, 90 out of 120 best-known solutions for Taillard benchmark instances were improved showing the effectiveness of the proposed algorithm. Recently, Tasgetiren et al. (2017) presented highly efficient iterated greedy algorithms. Also, speed-up methods are used to obtain a better performance of the algorithms. Computational results show that the iterated greedy algorithms proposed by Tasgetiren et al. (2017) outperform all the state-of-the-art heuristics currently reported in the literature.

Several articles stress the necessary of developing exact procedures for the $Fm|block|C_{\max}$ problem, e.g. Lin and Ying (2013). However, only a few exact methods have been given for the $Fm|block|C_{\max}$ problem. The simplest way is to enumerate all permutations. However, this method is not practical even for problems of small size. Ronconi (2005) proposed an exact branch and bound (B&B) algorithm. This algorithm provides lower and upper bounds for several sets of benchmark instances. Companys and Mateo (2007) proposed the LOMPEN algorithm. This algorithm is based on the double branch and bound method. The idea is to apply simultaneously B&B algorithms to direct and reverse instances and then to create links and data exchanges between both processes. Companys and Ribas (2011) proved the optimality for only one Taillard 20×10 type benchmark instance using the LOMPEN algorithm. Bautista et al. (2012) proposed the bounded dynamic programming (BDP) algorithm. This method combines features of dynamic programming with features of B&B. The BDP algorithm is proposed as a heuristic. However, this algorithm can easily be interpreted as the exact algorithm.

The dynamic programming approach has also been developed for the job shop scheduling problem. This problem is another variant of the shop scheduling. Gromicho et al. (2012) [a corrigendum on this paper by van Hoorn et al. (2016)] proposed the DP algorithm with a complexity proven to be exponentially lower than exhaustive enumeration. Recently, van Hoorn (2016) provided the BDP algorithm that extends the previous version of the DP algorithm for the job shop.

The objective of the present paper is to develop the exact procedure to solve the $Fm|block|C_{\max}$ problem. Main contributions of this paper can be summarized as follows. We improve the BDP approach. Computational results confirm that our version of the BDP algorithm significantly outperforms the base version proposed by

Table 1 Basic notations

$\mathcal{J} = \{1, \ldots, n\}$	Set of jobs
$\mathcal{M} = \{1, \ldots, m\}$	Set of machines
$i, j \in \mathcal{J}$	Job indices
$k \in \mathcal{M}$	Machine index
$p_{i,k}$	Processing time associated with machine k and job i
$C_{\max}$	Maximum completion time
$t = \lvert S \rvert$	Cardinality of set S, i.e. number of scheduled jobs
$\overline{S} = \mathcal{J} \setminus S$	Set of unscheduled jobs
$\overline{n} = n - t$	Number of unscheduled jobs
$\Pi(S)$	Set of all job permutations associated with S
$\pi \in \Pi(S)$	Job permutation associated with $S \subset \mathcal{J}$
$e_k(\pi)$	Departure time of the last scheduled job on machine k
$r_k(\pi)$	The earliest possible starting time of unscheduled job in machine k
$\ell_i(k_1, k_2)$	Time lag between machines k_1 and k_2 associated with job i
$\pi^1 \prec \pi^2$	$\pi^1 \in \Pi(S)$ dominates $\pi^2 \in \Pi(S)$
$\pi^1 \equiv \pi^2$	$\pi^1 \in \Pi(S)$ is equivalent to $\pi^2 \in \Pi(S)$
$\pi \oplus i$	Expansion of $\pi \in \Pi(S)$ adding job i
$Z(S) \subset \Pi(S)$	Specific set of job permutations associated with S
$\lvert Z \rvert_t$	Number of job permutations stored at t
$\lvert Z \rvert$	Total number of job permutations
$\lvert Z_{\max} \rvert$	Maximum number of job permutations stored at one stage
$LB(\pi)$	Lower bound of optimal makespan of π

Bautista et al. (2012). Furthermore, computational results show that our version of the BDP algorithm outperforms all the afore mentioned state-of-the-art exact algorithms. Finally, we show that the presented BDP algorithm can be successfully applied for lower bound calculations.

This paper is organized as follows. Section 2 presents a problem description and basic notations. In Sect. 3, we describe lower bounds that are used in the present work. Section 4 proposes our version of BDP. Computational results are given in Sect. 5. Finally, concluding remarks and some possible directions for future research are described in Sect. 6.

2 Basic notations

The basic notations used in this paper are summarized in Table 1.

In the permutation flow shop problem, n jobs have to be processed in m machines in the same order, from machine 1 to machine m. However, this order is not known. The processing time associated with a specific machine $k \in \mathcal{M} = \{1, \ldots, m\}$ and a specific job $i \in \mathcal{J} = \{1, \ldots, n\}$ is denoted by $p_{i,k}$. These times are non-negative.

The objective is to find a permutation π of n jobs such that the maximum completion time (the time when all jobs are completed) is minimal. However, if there does not exist any storage capacity amongst machines, a job cannot leave the machine while the next machine is not free. In this case, the permutation flow shop problem becomes the blocking flow shop problem which is denoted by $Fm|block|C_{\max}$.

Let S be a set of scheduled jobs and let $\pi \in \Pi(S)$ be the job permutation associated with S. Here $\Pi(S)$ stands for the set of all job permutations associated with S. Let

$$e(\pi) = (e_1(\pi), \ldots, e_m(\pi))$$

be the vector consisting of departure times $e_k(\pi)$ on machine $k \in \mathcal{M}$ and let

$$r(\pi) = (r_1(\pi), \ldots, r_m(\pi))$$

be the vector consisting of the heads $r_k(\pi)$ for each machine k. These heads can be interpreted as lower bounds for the starting time of any job in machine k. By defining the head $r_k(\pi)$ we take into account that no job can be processed simultaneously by two or more machines. Define time lags $\ell_i(k_1, k_2)$ between machines k_1 and k_2 $(k_1 < k_2)$ for each job i in the following way:

$$\ell_i(k_1, k_2) = \sum_{k=k_1}^{k_2-1} p_{i,k}.$$

The head $r_k(\pi)$ is defined as

$$r_k(\pi) = \min_{i \in \overline{S}} r_{i,k}(\pi), \tag{1}$$

where $\overline{S} = \mathcal{J} \setminus S$ and

$$r_{i,k}(\pi) = \max\{e_k(\pi), r_{i,1}(\pi) + \ell_i(1, k), \ldots, r_{i,k-1}(\pi) + \ell_i(k-1, k)\}. \tag{2}$$

From 2 it follows that $e_k(\pi) \leq r_k(\pi)$ for all $k \in \mathcal{M}$, $S \subset \mathcal{J}$, and $\pi \in \Pi(S)$.

We study an example in order to explain the concepts of heads. In this example we have five jobs and five machines. The sequence $\pi = (4, 5)$ is depicted in Fig. 1. Thus, $\overline{S} = \{1, 2, 3\}$. Processing times of remaining jobs are given in Table 2.

Figure 1 shows that

$$e_k(\pi) < r_k(\pi)$$

for $k \in \{2, 3, 4\}$. The heads $r_{i,k}$ for $i \in \{1, 2, 3\}$ and $k \in \mathcal{M}$ are given in Table 2. These heads are calculated using (2). Values in bold correspond to those $r_{j,k}$ that are minimal among all $r_{i,k}$ with $i \in \overline{S}$.

We define dominance '$\prec$' and equivalence '$\equiv$' relations between job permutations $\pi^1, \pi^2 \in \Pi(S)$ in the following way:

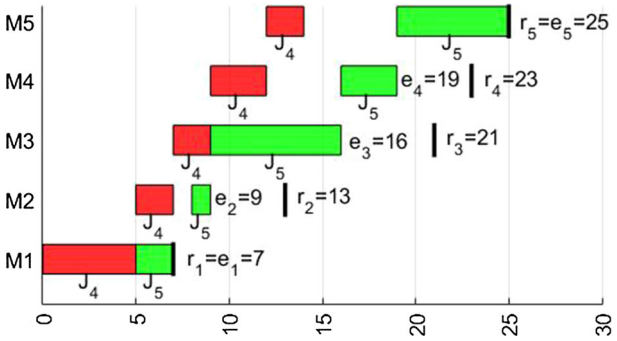

Fig. 1 Example of the departure times and heads

Table 2 Processing, departure times, and heads of an example with $\pi = (4, 5)$

k	1	2	3	4	5
$p_{1,k}$	6	10	6	1	2
$p_{2,k}$	9	5	2	1	6
$p_{3,k}$	7	9	7	4	1
$e_k(\pi)$	7	9	16	19	25
$r_{1,k}(\pi)$	7	**13**	23	29	30
$r_{2,k}(\pi)$	7	16	**21**	**23**	**25**
$r_{3,k}(\pi)$	7	14	23	30	34
$r_k(\pi)$	7	13	21	23	25

$$\pi^1 \equiv \pi^2 \iff r_k(\pi^1) = r_k(\pi^2) \text{ for all } k \in \mathcal{M};$$
$$\pi^1 \prec \pi^2 \iff \pi^1 \not\equiv \pi^2 \text{ and } r_k(\pi^1) \leq r_k(\pi^2) \text{ for all } k \in \mathcal{M}. \tag{3}$$

In (3), Bautista et al. (2012) used $e(\pi)$ instead of $r(\pi)$ to define the dominance and equivalence. However, our version of the DP can also discard additional job permutations. It is shown by the following theorem as well as by the counterexample illustrated in Fig. 2:

Theorem 1 *Let $S \subset \mathcal{J}$, $S \neq \mathcal{J}$, and let $\pi^1, \pi^2 \in \Pi(S)$. If $e_k(\pi^1) \leq e_k(\pi^2)$ for all $k \in \mathcal{M}$, then $\pi^1 \prec \pi^2$ or $\pi^1 \equiv \pi^2$.*

Proof We will prove that

$$r_{i,k}(\pi^1) \leq r_{i,k}(\pi^2) \tag{4}$$

for all $i \in \overline{S}$ and $k \in \mathcal{M}$. Then the statement

$$\left[r_k(\pi^1) \leq r_k(\pi^2) \text{ for all } k \in \mathcal{M}\right] \Longrightarrow \left[\pi^1 \prec \pi^2 \text{ or } \pi^1 \equiv \pi^2\right].$$

follows from (1)

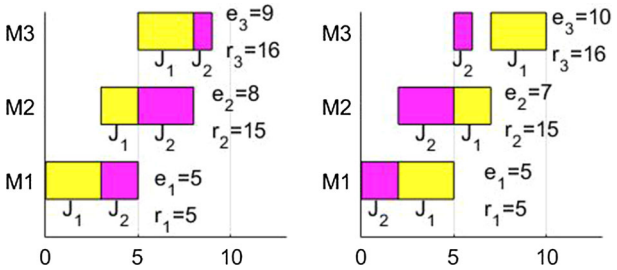

Fig. 2 On the *left side* $\pi^1 = (1, 2)$. On the *right side* $\pi^2 = (2, 1)$

Table 3 Processing, departure times, and heads of an example with $\pi^1 = (1, 2)$ and $\pi^2 = (2, 1)$

k	1	2	3
$p_{1,k}$	3	2	3
$p_{2,k}$	2	3	1
$p_{3,k}$	10	1	1
$e_k(\pi^1)$	5	8	9
$e_k(\pi^2)$	5	7	10
$r_k(\pi^1)$	5	15	16
$r_k(\pi^2)$	5	15	16

Fix $i \in \overline{S}$ and set $k = 1$. Then $r_{i,k}(\pi^1) = e(\pi^1)$ and $r_{i,k}(\pi^2) = e(\pi^2)$ from which it follows that (4) holds for $k = 1$.

Assume that (4) holds for all $k \in \{1, \ldots, h\}$ where $h \in \{1, \ldots, m-1\}$. Then we prove that (4) holds also for $k = h + 1$. Using the assumptions formulated in the theorem we have

$$
\begin{aligned}
&e_k(\pi^1) \le e_k(\pi^2),\\
&r_{i,1}(\pi^1) \le r_{i,1}(\pi^2) \Rightarrow r_{i,1}(\pi^1) + \ell_i(1,k) \le r_{i,1}(\pi^2) + \ell_i(1,k),\\
&\quad \ldots\\
&r_{i,k-1}(\pi^1) \le r_{i,k-1}(\pi^2) \Rightarrow r_{i,k-1}(\pi^1) + \ell_i(k-1,k) \le r_{i,k-1}(\pi^2) + \ell_i(k-1,k)
\end{aligned}
$$

from which it follows that (4) holds. Since i is fixed, then by recursion it can be obtained that (4) is true for all $i \in \overline{S}$ and $k \in \mathcal{M}$. □

Now a simple example will be given. This example shows that $\pi^1 \equiv \pi^2$, whereas the statement

$$e_k(\pi^1) \le e_k(\pi^2), \quad k \in \mathcal{M}$$

is not true.

Processing times are given in Table 3. Select $S = \{1, 2\}$, $\pi^1 = (1, 2)$, $\pi^2 = (2, 1)$. Then from Table 3 it can be seen that $\pi^1 \equiv \pi^2$. On the other hand,

$$e_2(\pi^1) > e_2(\pi^2),$$
$$e_3(\pi^1) < e_3(\pi^2).$$

3 Lower bounds for bounded dynamic programming algorithm

The performance of the BDP algorithm is strongly dependent on the quality of lower bounds. In this section, we will briefly describe lower bounds used in the current work. These LB are based on the optimal makespan of one or two machines.

Let $S \subset \mathcal{J}$ ($S \neq \mathcal{J}$) and $\pi \in \Pi(S)$ be given. The lower bound $LB^{(0)}$ was used by Bautista et al. (2012). Here,

$$LB_k^{(0)}(\pi) = e_k(\pi) + \sum_{i \in \overline{S}} p_{i,k} + \min_{i \in \overline{S}} q_{i,k}, \tag{5}$$

where the tails $q_{i,k}$ are obtained as follows:

$$q_{i,k} = \sum_{k_1=k+1}^{m} p_{i,k_1}.$$

The first lower bound used in the current work is a slight modification of $LB_k^{(0)}$ where $e_k(\pi)$ in (5) is replaced by $r_k(\pi)$, i.e.

$$LB_k^{(1)}(\pi) = r_k(\pi) + \sum_{i \in \overline{S}} p_{i,k} + \min_{i \in \overline{S}} q_{i,k}.$$

Hence, a valid lower bound for π is

$$LB^{(h)}(\pi) = \max_{1 \leq k \leq m} LB_k^{(h)}(\pi), \quad h \in \{0, 1\}. \tag{6}$$

The lower bound $LB^{(0)}$ will be used only for an illustrative example. The lower bound $LB^{(1)}(\pi)$ can be computed in $O(m \cdot \overline{n})$ time.

The second lower bound used in the present work is based on two machines. Denote by $C^{k_1,k_2}(\pi)$ the optimal makespan of the two machine scheduling problem with time lags $F2|\ell_i(k_1, k_2)|C_{\max}$. This makespan is computed for the machine pair (k_1, k_2) where $k_1 < k_2$. Lenstra et al. (1977) showed that C^{k_1,k_2} can be obtained by applying Johnson rule to $F2||C_{\max}$ with processing times $(\ell_i(k_1, k_2), p_{i,k_2} + \ell_i(k_1, k_2) - p_{i,k_1})$. In this case the artificial job 0 is added with the following characteristics:

$$\begin{aligned} p_{0,k_1} &= 0, \\ p_{0,k_2} &= r_{k_2}(\pi) - r_{k_1}(\pi), \\ \ell_0(k_1, k_2) &= \ell_1. \end{aligned} \tag{7}$$

Then

$$LB^{(2)}_{k_1,k_2}(\pi) = r_{k_1}(\pi) + C^{k_1,k_2}(\pi) + \min_{i\in\overline{S}} q_{i,k_2}, \tag{8}$$

where $1 \leq k_1 < k_2 \leq m$. The resulting lower bound is

$$LB^{(2)}(\pi) = \min_{1\leq k_1<k_2\leq m} LB^{(2)}_{k_1,k_2}(\pi). \tag{9}$$

Computing $LB^{(2)}_{k_1,k_2}(\pi)$ in (8) requires the complexity $O(\overline{n}\log\overline{n})$. However, the complexity can be reduced by pre-calculating the order of tails at the beginning of the algorithm. Therefore, applying Johnson rule to $F2||C_{\max}$ will require only $O(\overline{n})$. Thus, the computation of $LB^{(2)}$ in (9) leads to the complexity $O(m^2\cdot\overline{n})$

Consider now the machine pair $(k, k+1)$ where $1 \leq k < m$. Gilmore and Gomory (1964) algorithm can be applied for the $F2|block|C_{\max}$ case since this problem can be reduced to the travelling salesman problem (TSP) with $\overline{n}+1$ nodes. In this case the following parameters for the artificial job 0 are used:

$$p_{0,k} = 0, \tag{10}$$

$$p_{0,k+1} = \min\left\{r_{k+1}(\pi) - r_k(\pi), \min_{i\in\overline{S}} p_{i,k+1}\right\}. \tag{11}$$

Thus,

$$LB^{(3)}_k(\pi) = r_k(\pi) + C^k(\pi) + \min_{i\in\overline{S}} q_{i,k+1}, \quad 1 \leq k < m, \tag{12}$$

where $C^k(\pi)$ is obtained using the Gilmore and Gomory algorithm to the $F2|block|C_{\max}$ including the artificial job 0. Finally,

$$LB^{(3)}(\pi) = \min_{1\leq k<m} LB^{(3)}_k(\pi). \tag{13}$$

The Gilmore and Gomory algorithm requires sorting $\overline{n}$ jobs that cannot be initialized at the beginning of the algorithm. Therefore, the complexity is $O(\overline{n}\cdot\log\overline{n})$ in order to calculate $LB^{(3)}_k(\pi)$ (see (12)). Finally, looping through all machines requires the overall complexity $O(m\cdot\overline{n}\log\overline{n}))$ to calculate the lower bound $LB^{(3)}(\pi)$ in (13).

Consider now the example that illustrates the calculation of lower bounds. Benchmark instance car1 with processing times depicted in Table 4 is chosen and the job permutation $\pi = (8, 5, 4, 2, 3, 6)$ is taken. Table 5 shows that $LB^{(1)}_5(\pi) > LB^{(0)}_5(\pi)$ since $r_5(\pi)$ is greater than $e_5(\pi)$. This leads to the greater overall lower bound $LB^{(1)}(\pi) > LB^{(0)}(\pi)$. Further, $LB^{(2)}_{k_1,k_2}(\pi) > LB^{(1)}(\pi)$ for several machine pairs (k_1, k_2) as can be seen in Tables 5 and 6. Therefore, the second lower bound

$$LB^{(2)}(\pi) = LB^{(2)}_{4,5}(\pi) = 7395 > LB^{(1)}(\pi).$$

Table 4 Processing times for benchmark instance car1

Machines	Jobs										
	1	2	3	4	5	6	7	8	9	10	11
1	375	632	12	460	528	796	532	14	257	896	532
2	12	452	876	542	101	245	230	124	527	896	302
3	142	758	124	523	789	632	543	214	753	214	501
4	245	278	534	120	124	375	896	543	210	258	765
5	412	398	765	499	999	123	452	785	463	259	988

Table 5 Parameters of the job sequence $\pi = (8, 5, 4, 2, 3, 6)$

k	1	2	3	4	5
$e_k(\pi)$	2962	3207	3839	4477	4600
$r_k(\pi)$	2962	3219	3839	4477	4722
$\sum_{i\in\overline{S}} p_{i,k}$	2592	1967	2153	2374	2574
$\min_{i\in\overline{S}} q_{i,k}$	811	731	517	259	0
$LB_k^{(0)}(\pi)$	6365	5905	6509	7110	**7174**
$LB_k^{(1)}(\pi)$	6365	5917	6509	7110	**7296**
$C^{k,k+1}(\pi)$	2604	2773	3012	2918	
$LB_{k,k+1}^{(2)}(\pi)$	6297	6509	7110	**7395**	
$C^k(\pi)$	2968	2552	2827	3121	
$LB_k^{(3)}(\pi)$	6661	6288	6925	**7598**	

Bold values indicate the maximum value in a row

Table 6 Parameters of the job sequence $\pi = (8, 5, 4, 2, 3, 6)$

(k_1, k_2)	(1, 3)	(1, 4)	(1, 5)	(2, 4)	(2, 5)	(3, 5)
$C^{k_1,k_2}(\pi)$	3438	3889	4431	3632	4077	3525
$LB_{k_1,k_2}^{(2)}(\pi)$	6917	7110	**7393**	7110	7296	7364

Bold value indicates the maximum value in a row

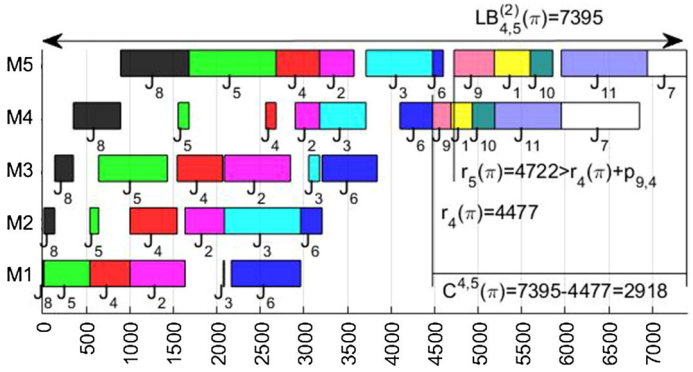

Fig. 3 Calculation of the lower bound $LB_{4,5}^{(2)}(\pi)$

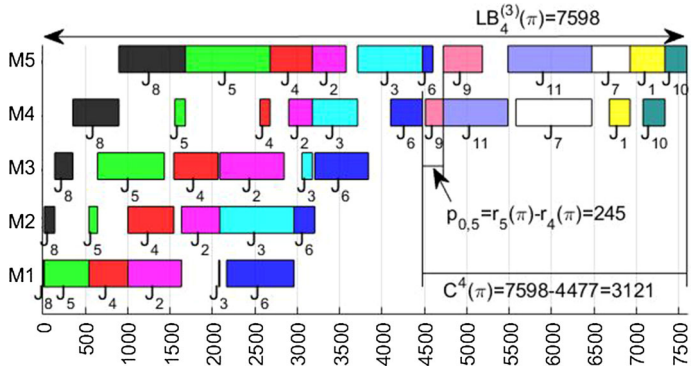

Fig. 4 Calculation of the lower bound $LB_4^{(3)}(\pi)$

Table 7 Lower bounds of the given example

h	0	1	2	3
$LB^{(h)}(\pi)$	7174	7296	7395	7598

Finally, the third lower bound

$$LB^{(3)}(\pi) = LB_4^{(3)}(\pi) = 7598$$

is greater than $LB^{(2)}(\pi)$ because the Gilmore and Gomory TSP algorithm for machine $k = 4$ gives a better makespan than the Johnson algorithm, which ignores the blocking constraint.

Figures 3 and 4 illustrate how the lower bounds $LB^{(2)}(\pi)$ and $LB^{(3)}(\pi)$ are obtained. In Fig. 3 it is shown that job $j = 9$ on machine $k = 5$ is delayed (i.e. starts later than the completion time of job on machine 4) because of the inequality $r_5(\pi) - r_4(\pi) > p_{9,4}$. This case demonstrates the usefulness of artificial job 0 with parameters defined in (7). The Gilmore and Gomory algorithm gives different job order than that of the Johnson algorithm because the latter ignores the blocking constraint. In addition, the use of artificial job 0 with parameters defined in (10) strengthens the lower bound $LB^{(3)}(\pi)$, see Fig. 4 for the explanation. Summary of the lower bounds is given in Table 7. This example shows that the values of all four lower bounds may differ.

4 Bounded dynamic programming algorithm

Let $\Pi(S)$ be a set consisting of all job permutations associated with S and let $Z(S)$ be a subset of $\Pi(S)$. We formulate the following generalized Bellman equation for the $Fm|block|C_{\max}$ problem:

$$Z(\{i\}) = \{(i)\}, \quad i \in \mathcal{J}; \tag{14}$$

$$Z(S) = \text{Min}\left\{\bigcup_{\pi \in Z(S\setminus\{i\}):\, i \in S} \pi \oplus i\right\}, \qquad S \subset \mathcal{J}:\ 1 < |S| < n; \tag{15}$$

$$Z(\mathcal{J}) = \underset{\pi \in Z(\mathcal{J}\setminus\{i\}:\, i \in \mathcal{J})}{\arg\min} e_m(\pi \oplus i). \tag{16}$$

Here, $\text{Min}\, A \subset A$ stands for the set of all minimal elements according to '$\prec$' and '$\equiv$'. It means that for all $\pi \in \text{Min}\, A$ there does not exist another element that dominates π or is equivalent to π. The solution of the blocking flow shop problem is $Z(\mathcal{J})$. Note that for the solution $Z(\mathcal{J})$ of system (14)–(16) the property $|Z(\mathcal{J})| = 1$ holds. Thus, $e_m(\pi)$ $(\pi \in Z(\mathcal{J}))$ is the optimal makespan of the given $Fm|block|C_{\max}$ problem. The state space of the dynamic programming algorithm can be reduced by discarding those job permutations π for which $LB(\pi) \geq UB$. In addition, if $|Z|_t > H$ (i.e. the number of the job permutations with a size t is greater than a fixed window width H), then a sub-problem can be solved. Let $Z_1(S)$ be another subset of $\Pi(S)$. Then the following Bellman equation is formulated for Z_1:

$$Z_1(S) = \begin{cases} Z(S), & S \in T \\ \emptyset, & \text{otherwise} \end{cases}, \qquad S \subset \mathcal{J}:\ |S| = t_0; \tag{17}$$

$$Z_1(S) = \text{Min}\left\{\bigcup_{\pi \in Z_1(S\setminus\{i\}):\, i \in S} \pi \oplus i\right\}, \qquad S \subset \mathcal{J}:\ t_0 < |S| < n; \tag{18}$$

$$Z_1(\mathcal{J}) = \underset{\pi \in Z_1(\mathcal{J}\setminus\{i\}:\, i \in \mathcal{J})}{\arg\min} e_m(\pi \oplus i), \tag{19}$$

where $t_0 = t-1$ and $T \subset \{S \subset \mathcal{J}:\ |S| = t_0\}$. Note that if $t_0 = 1$ and $T = \{\{i\} | i \in \mathcal{J}\}$, then the Bellman equation (14)–(16) is equivalent to the Bellman equation (17)–(19) for a sub-problem.

Algorithm 2 presents the dynamic programming algorithm based on the Bellman equation. This algorithm creates $Z(\mathcal{J})$. If it is necessary, then T is also created.

Now we briefly describe Algorithm 2, which solves the $Fm|block|C_{\max}$ problem to optimality. Initially, n job permutations, which consist of a single job, are constructed. Then a loop through stages $t = 1$ to $n-2$ has been done (line 1.2). For each stage t, we loop through all subsets S of size t (line 1.3) and for each chosen subset S, we loop through all job permutations $\pi \in Z(S)$ (line 1.4). These job permutations are expanded with job i, which is not scheduled. If there does not exist another job permutation that dominates $\pi \oplus i$ or is equivalent to $\pi \oplus i$, then the lower bound of $\pi \oplus i$ is estimated. Finally, if

$$\max\left\{LB^{(1)}(\pi \oplus i), LB^{(2)}(\pi \oplus i), LB^{(3)}(\pi) \oplus i\right\} < UB,$$

then $\pi \oplus i$ is included in $Z(S \cup \{i\})$. In addition, all job permutations that are dominated by $\pi \oplus i$ are removed from $Z(S \cup \{i\})$.

Algorithm 1: Pseudocode of the procedure BDP.

```
1.1  Procedure BDP(t_0, UB, Z)
1.2  for t = t_0 to n − 2 do
1.3      forall the S ⊂ J : |S| = t do
1.4          forall the π ∈ Z(S) do
1.5              Z(S) = Z(S) \ {π}
1.6              forall the i ∈ S̄ do
1.7                  if π^1 ≠ π ⊕ i and π ⊕ i ≢ π^1 for all π^1 ∈ Z(S ∪ {i}) then
1.8                      if LB(π ⊕ i) < UB then
1.9                          D = {π^2 ∈ Z(S ∪ {i}) | π ⊕ i ≺ π^2}
1.10                         Z(S ∪ {i}) = Z(S ∪ {i}) ∪ {π ⊕ i} \ D
1.11         if termination criterion is met then
1.12             Z_1 = Z
1.13             t_1 = t
1.14             BDP(t_1, UB, Z_1)
1.15             Z(S) = ∅ for all S ⊂ J with |S| = t
1.16 if ∃i ∈ J : Z(J \ {i}) ≠ ∅ then
1.17     Z(J) is created from (16)
1.18     Select π ∈ Z(J)
1.19     UB = e_m(π)
```

Algorithm 2: The BDP algorithm that solves the $Fm|block|C_{\max}$ problem to optimality.

```
2.1  Step 1: initialization
2.2  t_0 = 1
2.3  UB = UB_0
2.4  Z({i}) = {(i)}, i ∈ J
2.5  Z(S) = ∅, where S ⊂ J and |S| > 1
2.6  Step 2: BDP(t_0, UB, Z) - the main part of the BDP algorithm.
2.7  Step 3: Opt = UB - the optimal makespan is obtained.
```

If the size of $Z(S \cup \{i\})$ becomes too large, then the memory can be exceeded. A typical termination criterion can be $|Z|_{t+1} > H$, i.e. the number of job permutations that are stored in new stage $t+1$ becomes greater than a fixed constant. In general, H depends on m, n, and the computer system requirements. The procedure BDP is recursively executed (line 1.14) and we go to the next stage by setting $t = t+1$ since there does not exist $S \subset \mathcal{J}$ with $|S| = t$. When the *for* loop is ended (line 1.2), the optimal makespan $Opt = e_m(\pi)$ ($\pi \in Z(\mathcal{J})$) is found. However, if $Z(\mathcal{J})$ is empty, then the optimal makespan is equal to UB. Generally speaking, UB_0 can be replaced by the general bound B_0. If the final UB is not less than B_0, then the BDP algorithm will prove that B_0 is the lower bound of the given blocking flow shop instance.

The major changes of the BDP against the algorithm given by Bautista et al. (2012) are summarized below.

1. The given algorithm is proposed as an exact algorithm instead of heuristics.

Table 8 Processing times for benchmark instance car7

Machines	Jobs						
	1	2	3	4	5	6	7
1	692	581	475	23	158	796	542
2	310	582	475	196	325	874	205
3	832	14	785	696	530	214	578
4	630	214	578	214	785	236	963
5	258	147	852	586	325	896	325
6	147	753	2	356	565	898	800
7	255	806	699	877	412	302	120

2. Bautista et al. (2012) used only the lower bound $LB^{(0)}$. In this work tighter lower bounds $LB^{(1)}$, $LB^{(2)}$, and $LB^{(3)}$ are used. These lower bounds have been described in Sect. 3.
3. The head $r(\pi)$ (see equation (1)) instead of the vector of the departure times $e(\pi)$ is used. It can be expected that the total number of job permutations $|Z|$ will be reduced in this case. In addition, the lower bounds $LB^{(h)}$, $h \in \{1, 2, 3\}$, can be strengthened if $r_k(\pi) > e_k(\pi)$ for at least one machine $k \in \mathcal{M}$.
4. The value $|Z|_{t+1}$ can grow too large. Therefore the procedure BDP is recursive and the termination criterion is included to avoid memory overreaching (lines 1.11–1.14). Therefore, the given algorithm is theoretically able to solve any instance without running out of the given amount of memory. Note that when the sup-problem (line 1.14) has been solved, then the upper bound UB can be improved.
5. We have formulated the Bellman equations (14)–(16) and (17)–(19) for the $Fm|block|C_{\max}$ problem that has not been done before.

Now we inspect an example to illustrate Algorithm 2. Bechmark instance car7 with processing times given in Table 8 is chosen.

Initially, we set $UB = 6888$ and $H = 13$. The DP algorithm returns the optimal schedule $\pi_{Opt} = (4, 5, 2, 6, 7, 3, 1)$ with

$$e(\pi_{Opt}) = (3752, 4062, 5172, 5802, 6060, 6207, 6788)\,.$$

Hence, the optimal makespan is $Opt = e_7(\pi_{Opt}) = 6788$. Table 9 shows that the value $|Z|_3$ exceeds the window width $H = 13$. Therefore, the sub-problem has to be solved with the initial condition

$$Z_1(S) = \begin{cases} \{(4,6)\}, & S = \{4,6\}; \\ \{(4,7)\}, & S = \{4,7\}; \\ \{(5,1)\}, & S = \{1,5\}; \\ \{(5,3)\}, & S = \{3,5\}; \\ \emptyset, & \text{otherwise.} \end{cases}$$

Table 9 Example of the proposed DP algorithm

t	$\pi \in Z(S)$ with $\lvert S\rvert = t$	$\lvert Z\rvert_t$
1	(1),(2),(3),(4),(5),(6),(7)	7
2	(2,5),(4,1),(4,2),(4,3),(4,5),(4,6),(4,7),(5,1),(5,2),(5,3),(5,4)	11
3	(4,2,1),(4,2,3),(4,2,5),(4,2,7),(4,1,2),(4,3,2),(4,3,5),(4,5,1),(4,5,2),(4,5,3), (4,5,6),(4,5,7),(5,4,2)	13*
2	(4,6),(4,7),(5,1),(5,3)	
3	(4,6,5),(4,7,1),(4,7,3),(4,7,5),(5,1,4),(5,3,2)	6
4	(4,7,3,2)	1
4	(4,2,5,3),(4,5,2,1),(4,5,2,6),(4,2,7,6),(4,5,1,2),(4,3,5,2),(4,5,3,2),(4,5,6,3)	8
5	(4,5,3,2,6),(4,5,2,6,3),(4,5,2,6,7),(4,5,1,2,6)	4
6	(4,5,2,6,7,3)	1
7	(4,5,2,6,7,3,1)	1
$Opt = 6788$, $\lvert Z\rvert = 52$, $\lvert Z_{\max}\rvert = 13$		

*The window width is exceeded

In this case the set T used in equation (17)) is equal to $\{\{4, 6\}, \{4, 7\}, \{1, 5\}, \{3, 5\}\}$. For the sub-problem $|Z_1|_5 = 0$ since the lower bound is greater than UB for all extensions of $\pi = (4, 7, 3, 2)$.

An unanswered question remains about the effective calculation of r. Let π be a given job permutation with $|\pi| < n$. The vector of departure times $e(\pi \oplus i)$ can be recursively obtained from $e(\pi)$ using the following recurrence equation:

$$e_k(\pi \oplus i) = \max\{e_{k-1}(\pi \oplus i) + p_{i,k}, e_{k+1}(\pi)\}, \qquad k \in \mathcal{M} \tag{20}$$

with the artificial departure times

$$e_0(\pi \oplus i) = e_1(\pi),$$
$$e_{m+1}(\pi) = 0.$$

An index i in (20) denotes the job given in Algorithm 1. Clearly, the computation of e requires the complexity $O(m)$ in the worst case. However, the calculation of r is more expensive requiring the worst case complexity $O(nm^2)$. Since $r_k \geq e_k$ for all $k \in \mathcal{M}$, we can efficiently obtain r in practice using Algorithm 3.

The effectiveness of Algorithm 3 lies in the principle that the use of (1) can be avoided as well as the further search through the set $\mathcal{J}$ if job j satisfying $r_{j,k}(\pi) = e_k(\pi)$ (line 3.6) is found.

We continue to inspect the previously described example (see Tables 4 and 5). Table 10 demonstrates the application of Algorithm 3. For example, if jobs are selected in increasing order of the job numbers, then it is sufficient to calculate $r_{1,3}(\pi)$ and $r_{1,4}(\pi)$ in order to obtain $r_3(\pi)$ and $r_4(\pi)$ respectively. However, when $k = 2$ or $k = 5$, it is necessary to calculate $r_{i,k}$ for all $i \in \{1, 7, 9, 10, 11\}$ since $r_k(\pi) > e_k(\pi)$ for $k \in \{2, 5\}$.

Algorithm 3: Algorithm that builds $r(\pi)$.

Input : $S \subset \mathcal{J}, i \in \overline{S}, \pi \in \Pi(S)$
Output: $r(\pi)$
3.1 get $e(\pi)$ from equation (20)
3.2 $r_1(\pi) = e_1(\pi)$
3.3 **for** $k = 2$ **to** m **do**
3.4 **forall the** $j \in \mathcal{J} \setminus S$ **do**
3.5 get $r_{j,k}(\pi)$ from equation (2)
3.6 **if** $r_{j,k}(\pi) = e_k(\pi)$ **then**
3.7 $r_k(\pi) = e_k(\pi)$
3.8 break
3.9 **else**
3.10 get $r_k(\pi)$ from equation (1)

Table 10 The calculation of $r(\pi)$ for the job sequence $\pi = (8, 5, 4, 2, 3, 6)$

k	1	2	3	4	5
$e_k(\pi)$	2962	3207	3839	4477	4600
$r_{1,k}(\pi)$		3337	**3839**	**4477**	4722
$r_{7,k}(\pi)$		3494	**3839**	**4477**	5373
$r_{9,k}(\pi)$		3219	**3839**	4592	4802
$r_{10,k}(\pi)$		3858	4754	4968	5226
$r_{11,k}(\pi)$		3494	**3839**	**4477**	5242
$r_k(\pi)$	2962	3219	3839	4477	4722

Value in **bold**: $e_k(\pi) = r_{i,k}(\pi)$

5 Computational results

We implement the BDP algorithm in C++ programming environment and compiled it with Microsoft Visual Studio. Windows 64 bit operating system with 8 GB RAM memory and 2.8 GHz CPU was used. Several sets of benchmark instances are used: Taillard (1993), Reeves (1995), Heller (1960). The number of jobs is 20 or 50. The number of machines goes from 5 to 20. The effectiveness of the proposed algorithm will be analysed in terms of CPU time, $|Z_{\max}|$, and $|Z|$ where

$$|Z| = \sum_{S \subset \mathcal{J}} |Z(S)|$$

is the total number of job permutations and

$$|Z_{\max}| = \max_{t=1,\ldots,n} |Z|_t \tag{21}$$

is the maximum number of job permutations stored at one stage. The value $|Z|_t$ in (21) stands for the number of job permutations stored at t, i.e.

$$|Z|_t = \sum_{S \subset \mathcal{J} \,:\, |S|=t} |Z(S)|.$$

The termination criterion (line 1.11 in Algorithm 1) depends on the problem and is set taking into account the system limits. In practice, the limit of $|Z_{\max}|$ is taken as $4.05 \cdot 10^7$ or $4.1 \cdot 10^7$ for 20×20 type benchmark instances. The sub-problem should be solved when the termination criterion is met (line 1.14 in Algorithm 1).

Table 11 reports the results of the improved BDP algorithm with $UB_0 = BKS$ in Algorithm 2. An asterisk in Table 11 refers to the case when it is previously proven that the best-known solution (BKS) is equal to the optimal makespan, see Companys and Ribas (2011). All remaining BKS are, in general, not optimal.

Table 11 shows that the optimality is proven for all inspected benchmark instances with $n = 20$. Unfortunately, our CPU times cannot be compared with those that are obtained by Companys and Ribas (2011) since the authors did not report the corresponding CPU times. Furthermore, the optimality is proven for 13 out of 14 instances of type 20×10 and for all 20×15 and 20×20 type instances that has not been done before. All 20×10 type instances were solved within a CPU time less than 30 min except for the instance reC07 that required the CPU time 50 min. 20×15 type instances were solved in less than two hours. It is significantly harder to prove the optimality for 20×20 type instances as shows the CPU times that varies from 2-3 hours up to 35 hours. However, our improved BDP algorithm still cannot solve benchmark instances with at least $n = 30$ jobs in a reasonable time limit.

Note that the termination criterion is met for five instances of type 20×20. However, lines 1.11–1.14 in Algorithm 1 allow us to avoid from the memory overreaching. For instances with m equal to 10 or 5, the proposed BDP algorithm solves all instances being far away from the termination criterion as it is shown under column '$|Z_{\max}|$' in Table 11.

Now we apply the BDP algorithm to calculate the lower bounds. We use the following strategy. We start with an initial bound B_0 that is placed in Algorithm 2 by replacing UB_0. Then we run the BDP algorithm. If the used bound is not lowered after the execution of the algorithm, then it is clear that the lower bound cannot be less than B_0. Therefore, the current bound is increased by 10, and we rerun the BDP algorithm. The run is interrupted if there exists t with $|Z|_t > H$ or the optimal makespan is found. The value $\sum |Z|$ in Table 12 stands for the sum of all $|Z|$ that is obtained from all previous bounds. The value LB_{root} denotes the combination of one- and two-machine based lower bound $\max\{LB^{(1)}, LB^{(2)}, LB^{(3)}\}$ (see (6), (9), and (13) in Sect. 3) that is obtained at the root of the BDP algorithm, i.e. π is empty and $S = \emptyset$. The window width H is set as 1000, 10000, or 100000, respectively. For instances of type 20×5 or 20×10 we also inspect the case when H is not taken into account thus solving the $Fm|block|C_{\max}$ problem to optimality. In this case, we do not use BKS values thus obtaining the optimal makespan independently of other algorithms.

Table 12 shows that all Taillard 20×5 type instances can be solved to optimality in minutes. 20×10 type instances were solved in one hour on average. We do not try to solve 20×20 type instances to optimality since even proving the optimality takes a significant time amount.

Table 11 The optimality proof offered by BDP

$n \times m$	Instance	BKS	$UB = BKS$		
			$\lvert Z_{\max}\rvert$	$\lvert Z\rvert$	CPU
20 × 5	reC01	1418*	702719	3280651	38
	reC02	1253*	508792	2323426	30
	reC03	1417*	2951273	14151542	152
	ta01	1374*	1050007	4980611	54
	ta02	1408*	326814	1506333	17
	ta03	1280*	1420394	6315020	70
	ta04	1448*	456418	2051498	25
	ta05	1341*	786769	3616124	40
	ta06	1363*	533216	2521293	31
	ta07	1381*	746821	3585683	48
	ta08	1379*	884623	4139667	42
	ta09	1373*	301330	1435959	17
	ta10	1283*	1241855	5866129	68
20 × 10	hel2	150	1790813	8077460	264
	reC07	1728	15900913	65529580	3196
	reC09	1709	8354760	33604196	1427
	reC11	1588	4066335	18604927	654
	ta11	1698*	611583	2707728	103
	ta12	1833	3980354	17171958	711
	ta13	1659	7200089	31060780	831
	ta14	1535	2824706	12430097	445
	ta15	1617	4816045	18452561	618
	ta16	1590	10793445	47261705	1529
	ta17	1622	1947583	8654402	246
	ta18	1731	8372334	39278303	1122
	ta19	1747	2638924	11763636	327
	ta20	1782	7343976	32682770	955
20 × 15	reC13	2104	18945092	78256828	7039
	reC15	2075	3921118	17466701	1327
	reC17	2082	13213470	55633121	4934
20 × 20	ta21	2436	40500000	589909534	109475
	ta22	2234	16645279	64467948	14538
	ta23	2479	40500000	383231517	83010
	ta24	2348	11920903	47186700	8145
	ta25	2435	40500000	423705792	100036
	ta26	2383	40500000	451425366	128244
	ta27	2390	27964915	114673847	21037
	ta28	2328	41000000	255657708	57675
	ta29	2363	17259020	63789377	15291
	ta30	2323	20228314	89514396	9513

Table 12 Lower bounds obtained using different window widths

$n \times m$	Inst.	BKS	LB_{root}	$H = 1000$		$H = 10000$		$H = 100000$		$H = \infty$	
				LB	CPU	LB	CPU	LB	CPU	CPU	$\sum \lvert Z \rvert$
20×5	ta01	1374	1278	1278	0	1298	1.2	1318	10	216	3458515
	ta02	1408	1355	1375	0.1	1375	0.4	1395	7.7	57	888044
	ta03	1280	1073	1173	0.2	1183	0.9	1213	10.6	305	4896312
	ta04	1448	1283	1353	0.1	1383	1.2	1413	12.2	95	1461705
	ta05	1341	1217	1237	0.1	1257	1	1287	12.7	182	2815569
	ta06	1363	1215	1275	0.1	1295	1.2	1325	13.6	104	1448545
	ta07	1381	1226	1316	0.1	1326	0.9	1346	11.4	157	2067694
	ta08	1379	1170	1270	0.1	1290	0.9	1320	10.9	164	2676269
	ta09	1373	1208	1308	0.2	1318	0.8	1348	12.1	59	853887
	ta10	1283	1108	1188	0.1	1208	1.4	1228	10.9	273	4072470
	Average	1363	1213	1277	0.1	1293	1.0	1319	11.2	161	2463901
20×10	ta11	1698	1494	1604	0.2	1634	3.4	1664	36.5	353	8491354
	ta12	1833	1552	1672	0.3	1702	2.9	1742	35.5	3109	73015600
	ta13	1659	1407	1497	0.3	1527	2.2	1567	32.6	4752	127590394
	ta14	1535	1337	1407	0.3	1437	4.2	1467	43.4	1713	36675177
	ta15	1617	1344	1474	0.3	1504	3.6	1534	33.9	3262	69722267
	ta16	1590	1313	1423	0.2	1453	2.7	1483	26.3	6908	173784318
	ta17	1622	1416	1486	0.3	1516	2.7	1556	34.9	1042	28031346
	ta18	1731	1379	1509	0.3	1549	3.4	1589	34.3	8163	216173456

Table 12 continued

$n \times m$	Inst.	BKS	LB_{root}	$H = 1000$		$H = 10000$		$H = 100000$		$H = \infty$	
				LB	CPU	LB	CPU	LB	CPU	CPU	$\sum \lvert Z \rvert$
	ta19	1747	1533	1643	0.2	1663	2	1693	25.5	1340	36477567
	ta20	1782	1453	1573	0.3	1613	3.2	1653	29.9	5017	128328755
	Average	1681	1423	1529	0.3	1560	3.0	1595	33.3	3566	89829023

$n \times m$	Inst.	BKS	LB_{root}	$H = 1000$		$H = 10000$		$H = 100000$	
				LB	CPU	LB	CPU	LB	CPU
20×20	ta21	2436	1996	2106	0.8	2156	11	2206	121.8
	ta22	2234	1815	2035	1.3	2075	14.2	2115	143.4
	ta23	2479	2008	2208	1.5	2248	12.7	2298	152.5
	ta24	2348	1917	2107	1.3	2137	9.9	2187	132.6
	ta25	2435	2022	2172	0.9	2212	8.7	2262	109.9
	ta26	2383	1969	2129	0.8	2169	10.5	2199	82.4
	ta27	2390	2067	2187	1	2217	8.1	2257	100.6
	ta28	2328	1892	2092	0.7	2142	11.8	2182	126
	ta29	2363	1915	2155	1.6	2195	16.4	2245	191.4
	ta30	2323	1914	2114	1	2154	10.5	2194	125.4
	Average	2372	1952	2131	1.1	2171	11.4	2215	128.6
50×5	ta31	2974	2730	2730	0.1	2730	1.3	2740	15.3
	ta32	3171	2958	2958	0	2958	0.4	2958	7.9
	ta33	2988	2657	2657	0.1	2667	4.4	2677	49.8
	ta34	3111	2751	2771	0.2	2781	1.1	2791	23.7
	ta35	3138	2842	2862	0	2862	0.4	2862	9.2
	ta36	3158	2827	2847	0.1	2857	3.3	2857	18.6

Table 12 continued

$n \times m$	Inst.	BKS	LB_{root}	$H = 1000$		$H = 10000$		$H = 100000$	
				LB	CPU	LB	CPU	LB	CPU
	ta37	3004	2724	2774	0.2	2784	9.5	2784	36.5
	ta38	3039	2838	2838	0	2838	0.4	2838	4.6
	ta39	2889	2547	2567	0.2	2577	2.3	2587	25.6
	ta40	3094	2798	2858	0.1	2858	2.6	2868	78
	Average	3057	2767	2786	0.1	2791	2.6	2796	26.9
50×10	ta41	3605	2926	3046	1.6	3056	9.1	3076	138.5
	ta42	3470	2828	2878	0.9	2898	9.6	2918	107.7
	ta43	3465	2830	2880	0.9	2890	5.3	2910	110.1
	ta44	3643	3021	3081	0.8	3091	8.3	3101	73.8
	ta45	3582	2908	3058	2.2	3068	14.5	3078	124.8
	ta46	3571	2941	3011	1.3	3021	8.4	3041	100.2
	ta47	3667	3075	3085	0.4	3095	5.7	3105	72.5
	ta48	3546	2991	3021	0.4	3031	5.2	3041	60.8
	ta49	3508	2823	2923	1	2933	6.2	2953	77.8
	ta50	3603	3046	3086	1.3	3106	13.6	3126	163.8
	Average	3566	2939	3007	1.1	3019	8.6	3035	103.0
50×20	ta51	4479	3527	3717	3.8	3737	29	3757	407.3
	ta52	4262	3529	3659	4.6	3679	41.8	3709	883.1
	ta53	4261	3370	3530	2.6	3550	25.5	3580	552.0
	ta54	4338	3336	3566	6.2	3586	45.2	3616	644.4

Table 12 continued

$n \times m$	Inst.	BKS	LB_{root}	$H = 1000$		$H = 10000$		$H = 100000$	
				LB	CPU	LB	CPU	LB	CPU
	ta55	4249	3371	3571	7.8	3591	60.9	3611	555.3
	ta56	4271	3495	3565	1.8	3585	22.4	3605	452.1
	ta57	4289	3480	3600	3.9	3630	46.6	3650	378.1
	ta58	4298	3451	3541	3.2	3571	45.1	3591	397.0
	ta59	4304	3457	3547	3.4	3567	31.1	3587	318.9
	ta60	4398	3498	3648	3.7	3678	45	3698	394.8
	Average	4315	3451	3594	4.1	3617	39.3	3640	498.3

Table 13 Comparison of lower bounds that are obtained by different strategies

$n \times m$	BKS	RON		LB_{root}	H=1000		H=10000		H=100000	
		LB	CPU		LB	CPU	LB	CPU	LB	CPU
20×5	1363	1277	7.7	1213	1277	0.1	1293	1.0	1319	11.2
20×10	1681	1416	21.9	1423	1529	0.3	1560	3.0	1595	33.3
20×20	2372	1847	32.8	1952	2131	1.1	2171	11.4	2215	128.6
50×5	3057	2788	16.9	2767	2786	0.1	2791	2.6	2796	26.9
50×10	3566	2954	30.5	2939	3007	1.1	3019	8.6	3035	103.0
50×20	4315	3414	36.2	3451	3594	4.1	3617	39.3	3640	498.3

Now we analyse lower bounds that are obtained from the BDP algorithm. Table 12 shows that the exact method is able to improve the root lower bound by a substantial margin. For $H = 1000$, CPU times are small and do not exceed eight seconds for the largest instances with 50×20. However, for the most part of instances, CPU times are less than one second. By increasing the window width to $H = 10000$, lower bounds are improved in a few seconds for most instances. Setting $H = 100000$ requires minutes for larger instances. The performance of the BDP algorithm is better if the proportion m/n is greater. If $m/n = 0.1$ (the case $m = 5$ and $n = 50$), the benefit of the BDP algorithm is negligible for the selected H values. However, if $m/n = 1$, the BDP algorithm is more efficient and LB values are substantially improved by increasing H.

It is natural to inspect other exact algorithms. We compare our BDP algorithm with the exact branch and bound algorithm proposed by Ronconi (2005). This algorithm is denoted as RON in Table 13. CPU times given by RON are comparable with those obtained by the BDP algorithm since RON is coded using a Pentium 4 with 2.4 GHz. Results of Ronconi (2005) are reported in Grabowski and Pempera 2007; Wang et al. 2010; Lin and Ying 2013 as reference values. Ronconi (2005) presented an interesting version of the branch and bound algorithm. However, the one-machine based lower bound is used. Table 13 reports average values for several benchmark sets of Taillard. As shown in Table 13, even the root lower bound, which is based on two machines, is competitive with the final lower bound LB_{RON} provided by RON. For the case 20×20, LB_{root} significantly outperforms LB_{RON} on average. It can be observed that LB values can be significantly improved running the BDP algorithm with the fixed H. The higher is H, the better is LB on average. However, the balance between CPU times and the selected H should be respected. Choosing $H = 10000$ guarantees that almost all CPU times are less than those reported by RON. On the other hand, LB values obtained by BDP algorithm can be even drastically higher than those obtained by RON. Examples with a higher proportion m/n highlight the superiority of BDP over RON.

Table 14 reports the average CPU times using different strategies of the BDP algorithm. BDP_0 stands for the results reported in Bautista et al. (2012). BDP_1 and BDP_2 stands for our version of the BDP algorithm. BDP_1 referees to the case when $UB = BKS$ (see Table 11), and BDP_2 referees to the strategy in which the optimal

Table 14 Comparison of CPU times with the previous version of the BDP algorithm

$n \times m$	[a]BDP_0			BDP_1		BDP_2
	ARPD	$\lvert Z_{max}\rvert$	CPU	$\lvert Z_{max}\rvert$	CPU	CPU
20×5	0.35	10000	1059	916233	48.8	161.2
20×10	1.11	10000	1395	5760132	887.6	3565.8

[a]Version of BDP algorithm proposed by Bautista et al. (2012)

makespan was obtained dynamically increasing the bound (see Table 12). Average relative percentage deviation ($ARPD$ in Table 14) is calculated as

$$RPD = 100 \cdot \frac{UB - BKS}{BKS}.$$

In Table 14, we skip the column $ARPD$ below titles 'BDP_1' and 'BDP_2' since the final $RPD = 0$ for all smaller benchmark instances.

Table 14 shows that our version of the BDP algorithm drastically increases the applicability of the BDP approach for solving the $Fm|block|C_{max}$ problem. For 20×5 type instances, CPU times decrease by 85% compared to those of BDP_0. Moreover, BDP_0 fails to solve optimally any instance from Taillard sets and the $ARPD$ value was still high. For the 20×10 case, CPU times are competitive with those reported in our work. However, the value $ARPD$ is significantly greater than 0 and the quality of final upper bounds obtained by BDP_0 is poor.

Table 14 above shows that BDP_1 is remarkably faster than BDP_2. Therefore, in order to solve small instances to optimality, a better strategy is to use good initial upper bound of the given instance. Several heuristics reported in the literature can obtain BKS (not proving the optimality) for smaller Taillard instances in a negligible time.

6 Conclusions

In this paper, the bounded dynamic programming (BDP) algorithm that solves the blocking flow shop problems is studied. We developed BDP for solving the blocking flow shop problem optimally. The proposed algorithm is able to obtain the optimal solution for instances with up to 400 operations (e.g. 20 jobs and 20 machines), and we report proven optimal solutions for 26 previously open benchmark instances. In addition, we provide new lower bounds for several sets of benchmark instances while requiring a relatively short CPU time.

There is still room for the improvement of the BDP algorithm. The algorithm that calculates an improved head presented in this paper could be improved thus reducing the computational expenses. Another idea could be to use some priority rules for the order of selecting job permutations from which the new job permutations are developed. The BDP approach could be applied for solving simultaneously direct and inverse instances exchanging data between them. How to do it remains an open

question. The author believes that the optimal solution of the blocking flow shop problems could be obtained in a more reasonable time for benchmark instances with higher n or m (e.g. $m = 10$ and $n = 30$) than investigated in the present work.

References

Bautista J, Cano A, Companys R, Ribas I (2012) Solving the $Fm|block|Cmax$ problem using bounded dynamic programming. Eng Appl Artif Intell 25(6):1235–1245

Companys R, Mateo M (2007) Different behaviour of a double branch-and-bound algorithm on $Fm|prmu|Cmax$ and $Fm|block|Cmax$ problems. Comput Oper Res 34(4):938–953

Companys R, Ribas I (2011) New insights on the blocking flow shop problem. Best solutions update. Tech. rep., working paper

Gilmore P, Gomory R (1964) Sequencing a one state-variable machine: a solvable case of the traveling salesman problem. Oper Res 12(5):655–679

Grabowski J, Pempera J (2007) The permutation flow shop problem with blocking. A tabu search approach. Omega 35(3):302–311

Gromicho J, Van Hoorn J, Saldanha-da Gama F, Timmer G (2012) Solving the job-shop scheduling problem optimally by dynamic programming. Comput Oper Res 39(12):2968–2977

Hall N, Sriskandarajah C (1996) A survey of machine scheduling problems with blocking and no-wait in process. Oper res 44(3):510–525

Heller J (1960) Some numerical experiments for an $M \times J$ flow shop and its decision-theoretical aspects. Oper Res 8(2):178–184

Lenstra J, Rinnooy Kan A, Brucker P (1977) Complexity of machine scheduling problems. Ann Discrec Math 1:343–362

Lin S, Ying K (2013) Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm. Omega 41(2):383–389

Pan Q, Wang L, Sang H, Li J, Liu M (2013) A high performing memetic algorithm for the flowshop scheduling problem with blocking. IEEE Trans Auto Sci Eng 10(3):741–756

Papadimitriou C, Kanellakis P (1980) Flowshop scheduling with limited temporary storage. J ACM (JACM) 27(3):533–549

Reddi S, Ramamoorthy C (1972) On the flow-shop sequencing problem with no wait in process. Oper Res Quart pp 323–331

Reeves C (1995) A genetic algorithm for flowshop sequencing. Comput Oper Res 22(1):5–13

Ribas I, Companys R, Tort-Martorell X (2011) An iterated greedy algorithm for the flowshop scheduling problem with blocking. Omega 39(3):293–301

Ronconi D (2005) A branch-and-bound algorithm to minimize the makespan in a flowshop with blocking. Ann Oper Res 138(1):53–65

Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64(2):278–285

Tasgetiren M, Pan Q, Suganthan P, Buyukdagli O (2013) A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem. Comput Oper Res 40(7):1729–1743

Tasgetiren M, Pan Q, Kizilay D, Suer G (2015) A populated local search with differential evolution for blocking flowshop scheduling problem. In: 2015 IEEE Congress on Evolutionary Computation (CEC), pp 2789–2796

Tasgetiren M, Kizilay D, Pan Q, Suganthan P (2017) Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion. Comput Oper Res 77:111–126

van Hoorn JJ (2016) Dynamic programming for routing and scheduling: Optimizing sequences of decisions

van Hoorn JJ, Nogueira A, Ojea I, Gromicho JA (2016) An corrigendum on the paper: solving the job-shop scheduling problem optimally by dynamic programming. Comput Oper Res

Wang L, Pan Q, Suganthan P, Wang W, Wang Y (2010) A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. Comput Oper Res 37(3):509–520

Zhang C, Xie Z, Shao X, Tian G (2015) An effective VNSSA algorithm for the blocking flowshop scheduling problem with makespan minimization. In: 2015 International Conference on Advanced Mechatronic Systems (ICAMechS), IEEE, pp 86–89