Oper Res Int J (2020) 20:1701–1728
https://doi.org/10.1007/s12351-018-0381-6

ORIGINAL PAPER

Bounded dynamic programming algorithm for the job shop problem with sequence dependent setup times

Ansis Ozolins[1]

Received: 30 November 2016 / Revised: 23 December 2017 / Accepted: 24 January 2018 /
Published online: 3 February 2018

Abstract In this paper, the job shop scheduling problem (JSP) with a makespan minimization criterion is investigated. Various approximate algorithms exist that can solve moderate JSP instances within a reasonable time limit. However, only a few exact algorithms are known in the literature. We have developed an exact algorithm by means of a bounded dynamic programming (BDP) approach. This approach combines elements of a dynamic programming with elements of a branch and bound method. In addition, a generalization is investigated: the JSP with sequence dependent setup times (SDST-JSP). The BDP algorithm is adapted for this problem. To the best of our knowledge, the dynamic programming approach has never been applied to the SDST-JSP before. The BDP algorithm can directly be used as a heuristic. Computational results show that the proposed algorithm can solve benchmark instances up to 20 jobs and 15 machines for the JSP. For the SDST-JSP, the proposed algorithm outperforms all the state-of-the-art exact algorithms and the best-known lower bounds are improved for 5 benchmark instances.

Keywords Job shop scheduling · Makespan · Sequence dependent setup times · Dynamic programming

1 Introduction

The job shop scheduling problem (JSP) is an important scheduling problem, which is NP-hard. Since the JSP has applications in several areas, the problem and its generalizations have been extensively studied.

✉ Ansis Ozolins
ansis.ozolins1989@gmail.com

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

Several approximate algorithms have been proposed to tackle the JSP within a reasonable amount of time. Adams et al. (1988) proposed the shifting bottleneck procedure. The tabu search technique has been applied by several researchers, e.g. Nowicki and Smutnicki (1996), Nowicki and Smutnicki (2005), Zhang et al. (2008). The genetic approach has been investigated by Della Croce et al. (1995) and Kurdi (2015). Gonçalves and Resende (2014) presented the biased random-key algorithm. Recently, Peng et al. (2015) presented a powerful hybrid search/path relinking algorithm.

Only a few exact algorithms are known for the JSP. One such algorithm is the branch and bound method (B&B). This method is proposed by several authors (Carlier and Pinson 1989; Brucker et al. 1994; Martin and Shmoys 1996). Carlier and Pinson (1989) proposed the B&B model combined with the concept of immediate selection, which is based on branching on disjunctions. It was the first exact method that solved Fisher and Thompson (1963) 10×10 benchmark instance. Brucker et al. (1994) proposed the B&B based on the disjunctive graph model. The basic scheduling decision is to fix precedence relations between the operations on the same machine. A block approach was used. Gromicho et al. (2012) a corrigendum on this paper by van Hoorn et al. 2016) proposed a dynamic programming (DP) algorithm with the complexity proven to be exponentially lower than exhaustive enumeration. However, computation results show that only 10×5 type benchmark instances were solved by the proposed algorithm. Recently, van Hoorn (2016) provided the bounded dynamic programming algorithm. Thus, the computational time of the previously proposed DP algorithm was drastically decreased for solving benchmark instances. However, the proposed algorithm was able to solve still limited size benchmark instances up to 10 machines and 10 jobs.

Consider now the job shop scheduling problem with sequence dependent setup times (SDST-JSP). This problem is a generalization of the JSP in which setup times occurs when the machine switches between two jobs. This feature significantly changes the nature of the problem. Thus, the problem becomes harder to solve. Within the current literature, the typical approach is to extend the methods that were applied to the classical JSP. We will highlight only the most important papers. Based on the work proposed by Brucker et al. (1994) and Brucker and Thiele (1996) extended the B&B for the general shop problem where SDST-JSP is a special case. They also provided new benchmark instances (denoted by BT96 and named t2-ps01 to t2-ps15). Cheung and Zhou (2001) used a hybrid algorithm that is based on a genetic algorithm and two dispatching rules. The shifting bottleneck approach is extended by Balas et al. (2008). An effective B&B was developed by Artigues and Feillet (2008). This algorithm extends constraint propagation techniques for the SDST-JSP. The lower bound calculation is based on the traveling salesman problem with time windows. The algorithm proposed by Artigues and Feillet (2008) was able to prove the optimality for the first time on two benchmark instances from BT96. Moreover, the lower bound is improved for six instances. The effective hybrid genetic algorithm that hybridizes a genetic algorithm with a local search is reported by González et al. (2008). Empirical results show that the new model outperforms all previous state-of-the-art results improving best-known solutions for 5 benchmark instances of BT96. González et al. (2009) combines a

genetic algorithm with a tabu search approach instead of a local search method thus outperforming several empirical results given by González et al. (2008). For both González et al. (2008, 2009), the problem is modeled using the disjunctive graph representation. Grimes and Hebrard (2010) proposed a constraint programming approach that extends the previous model given by Grimes et al. (2009). The model using simple binary disjunctive constraints combines AI strategies with the generic SAT.

In the previous study (Ozolins 2017) we have successfully developed the DP to the blocking flow shop scheduling problem. This problem is another variant of the shop scheduling. Ultimately, this approach outperforms all the known state-of-the-art exact algorithm for the blocking flow shop. This finding suggests that the DP technique can successfully be applied to other variants of the shop scheduling.

Gromicho et al. (2012) proposed the base version of the DP approach for the JSP. In this paper, we improve the DP method for solving the JSP thus increasing the applicability of the DP approach. The novel contribution of our work lies in the SDST-JSP. We extend the DP approach to the SDST-JSP with a makespan objective. As far as we know this is the first time when the DP approach is used for the SDST-JSP.

This paper is organized as follows. In Sect. 2, the JSP is introduced. Basic notations and definitions are given. Section 3 presents the BDP algorithm for the JSP and SDST-JSP cases. Computational results are described in Sect. 4. In Sect. 5, conclusions are given. Finally, "Appendix" is given in the last section.

2 Problem formulation

Basic notations used in the present paper are summarized in Table 1 for a quick reference.

Consider the JSP. The processing procedure of a job on a machine is called an operation. Let $\mathcal{O} = \{O_{i,k} \mid i = 1, \ldots, n, k = 1, \ldots m\}$ denotes the set of operations, partitioned into n jobs $J_1, \ldots, J_n$ that need to be scheduled on m machines $M_1, \ldots, M_m$. Let $\mathcal{J} = \{J_1, \ldots, J_n\}$ and $\mathcal{M} = \{M_1, \ldots, M_m\}$ denote the set of jobs and the set of machines, respectively. Each operation $O_{i,k}$ is associated with the specific job $J_i \in \mathcal{J}$ and the specific machine $M_{m_{i,k}} \in \mathcal{M}$. Hence, $m_{i,k}$ denotes the machine index of operation $O_{i,k}$. Alternatively, this index can be denoted by $M(O_{i,k})$. Each job has to visit all the machines following a specific order. Each machine can process only one job at the same time and each job can be processed by only one machine at the same time. We will study the special case of the JSP where each job has to visit all the machines exactly once.

The processing time of operation $O_{i,k}$ is denoted by $p(O_{i,k})$ or simply by $p_{i,k}$. Alternatively, the processing time associated with a machine M_k and a job J_j can be denoted by p_j^k. Denote by $\psi : \mathcal{O} \to \mathbf{N} \cup \{0\}$ the schedule. For simplicity, we will also use the notation $\psi_{i,k}$ as an alternative to the notation $\psi(O_{i,k})$. Let

$$C_{\max} = \max_{O \in \mathcal{O}} (\psi(O) + p(O))$$

Table 1 Basic notations

m	Number of machines
n	Number of jobs
$\mathcal{M}$	Set of machines, $\mathcal{M} = \{M_1, \ldots, M_m\}$
$\mathcal{J}$	Set of jobs, $\mathcal{J} = \{J_1, \ldots, J_n\}$
$O_{i,k}$	The kth operation of job J_i
$\mathcal{O}$	Set of operations, $\{O_{i,k} \mid i = 1, \ldots, n, k = 1, \ldots m\}$
$p_{j,k}$	Processing time of operation $O_{j,k} \in \mathcal{O}$
$p(O_{j,k})$	Alternative formulation of $p_{j,k}$
$M(O)$	Machine index of operation $O \in \mathcal{O}$
$m_{i,k}$	Alternative formulation of machine index, $m_{i,k} = M(O_{i,k})$
$\psi(O)$	The starting time of operation O
$\psi_{i,k}$	Alternative formulation of the starting time, $\psi_{i,k} = \psi(O_{i,k})$
$C_{\max}$	Maximum completion time among all operations, $C_{\max} = \max_{O \in \mathcal{O}}(\psi(O) + p(O))$
S	Set of scheduled jobs, $S \subset \mathcal{J}$
G	Set of scheduled operations, $G \subset \mathcal{O}$
T	Finite sequence of operations associated with S and $G, T = (T_1, T_2, \ldots, T_{\lvert G \rvert})$
$T + O$	Expansion of sequence T by adding operation $O \in \mathcal{O} \backslash G$
$s^k_{i,j}$	Setup time between operations O_{i,k_1} and O_{j,k_2} on the same machine M_k
$s^k_{0,i}$	Initial setup time of job J_i on machine M_k
$\rho_k(T)$	Job index corresponding to the last operation completed on machine M_k
$C_k(T)$	Completion time of the last operation scheduled on machine M_k
$Succ_j(G)$	Next operation needed to be scheduled by job J_j
$x_j(T)$	Binary variable representing whether $Succ_j(G)$ can be added to the end of T
$C_{\max}(T)$	Maximum completion time among all operations from T
$\psi_j(T)$	The earliest starting time of $Succ_j(G)$ where T is associated with G
$r_{j,k}(T)$	The earliest starting time of operation $O_{j,k} \in \mathcal{O} \backslash G$
$q_{i,k}$	Tail of operation $O_{i,k}$
$T^1 \preceq T^2$	T^1 weakly dominates T^2
$Z(G)$	Set of sequences associated to $G \subset \mathcal{O}$
$\lvert Z \rvert_t$	Number of sequences with size $t, \lvert Z \rvert_t = \sum_{G \subset \mathcal{O}: \lvert G \rvert = t} \lvert Z(G) \rvert$
$\lvert Z \rvert$	Total number of sequences, $\lvert Z \rvert = \sum_{G \subset \mathcal{O}} \lvert Z(G) \rvert$
UB	Upper bound
$LB(T)$	Lower bound of sequence T

be the makespan that corresponds to the schedule ψ. The problem can formally be stated as follows.

Definition 1 (*Job shop scheduling problem*)

$$
\begin{aligned}
&\underset{\psi}{\text{minimise}}\ C_{\max} \\
&\text{subject to:} \\
&C_{\max} \geq \psi_{i,m} + p_{i,m}, && i = 1, \ldots, n; \\
&\psi_{i,k} + p_{i,k} \leq \psi_{i,k+1}, && i = 1, \ldots, n,\ k = 1, \ldots, m-1; \\
&\psi_{i,k_1} + p_{i,k_1} \leq \psi_{j,k_2} \text{ or} \\
&\psi_{j,k_2} + p_{j,k_2} \leq \psi_{i,k_1} && O_{i,k_1}, O_{j,k_2} \in \mathcal{O} \text{ with } i \neq j \text{ and } m_{i,k_1} = m_{j,k_2}.
\end{aligned}
$$

Using the notation given by Graham et al. (1979), the JSP can also be denoted by $J||C_{\max}$.

The JSP can be generalized by defining the setup times $s^k_{i,j}$ between operations $O_{i,k_1}, O_{j,k_2} \in \mathcal{O}$ on the same machine $M_k \in \mathcal{M}$. In addition, an initial setup $s_{0,i}$ is defined for all $i \in \{1, \ldots, n\}$. This problem is denoted by $J|s^k_{i,j}|C_{\max}$ and can be formalized in the following way.

Definition 2 (*Job shop problem with sequence dependent setup times*)

$$
\begin{aligned}
&\underset{\psi}{\text{minimise}}\ C_{\max} \\
&\text{subject to:} \\
&C_{\max} \geq \psi_{i,m} + p_{i,m}, && i = 1, \ldots, n; \\
&\psi_{i,k} + p_{i,k} \leq \psi_{i,k+1}, && i = 1, \ldots, n,\ k = 1, \ldots, m-1; \\
&\psi_{i,k_1} + p_{i,k_1} + s^k_{i,j} \leq \psi_{j,k_2} \text{ or} && O_{i,k_1}, O_{j,k_2} \in \mathcal{O} \text{ with} \\
&\psi_{j,k_2} + p_{j,k_2} + s^k_{j,i} \leq \psi_{i,k_1} && i \neq j \text{ and } m_{i,k_1} = m_{j,k_2} \equiv k; \\
&\psi_{i,1} \geq s_{0,i} && i = 1, \ldots, n.
\end{aligned}
$$

In addition, we assume that setup times satisfy the triangular inequality

$$s^k_{i,j} \leq s^k_{i,h} + s^k_{h,j}$$

for all $k \in \{1, \ldots, m\}$ and for each triplet of distinct jobs (J_i, J_j, J_h).

3 Bounded dynamic programming algorithm

In this section, we will develop the bounded dynamic programming (BDP) approach for solving the JSP. The base version of this approach was proposed by Gromicho et al. (2012). In Sect. 3.1, the complete BDP algorithm will be proposed. The SDST-JSP will be studied in Sect. 3.2. In Sects. 3.3 and 3.4, we will discuss lower bounds for the JSP and SDST-JSP.

3.1 Bounded dynamic programming algorithm formulation

Let T be the sequence of operations. We will further say that T is associated with $G \subset \mathcal{O}$ if G contains all the operations that appear in T. Those operations that are

not in T are not included in G. We say that T is also associated with $S \subset \mathcal{J}$ if S contains those jobs for which all the operations are completed. For example, if we have the job shop instance with 3 jobs and 3 machines and

$$T = (O_{3,1}, O_{1,1}, O_{1,2}, O_{2,1}, O_{1,3}),$$

then T is associated with $S = \{J_1\}$ and

$$G = \{O_{3,1}, O_{1,1}, O_{1,2}, O_{2,1}, O_{1,3}\}.$$

A no-idle starting time of an operation means that this operation starts directly after the last operation on the same machine or directly after the last operation of the same job. We will further assume that $\psi(O_{i,k})$ stands for this no-idle starting time of operation $O_{i,k} \in G$.

The sequence T is called feasible if

$$\psi(T_1) + p(T_1) \leq \cdots \leq \psi(T_{|G|}) + p(T_{|G|}).$$

In addition, $M(T_g) < M(T_{g+1})$ for those $g \in \{1, \ldots, |G| - 1\}$ for which $\psi(T_g) + p(T_g) = \psi(T_{g+1}) + p(T_{g+1})$. From the construction of T it follows that each sequence T can uniquely be associated with the given schedule.

Denote by $C_{\max}(T)$ the maximum completion time among all the operations from T. Let $Succ_i(G)$ denotes the next operation that needs to be scheduled by job $J_i \in \mathcal{J} \backslash S$. Let $\psi_i(T)$ be the earliest starting time of operation $Succ_i(G)$. A sequence obtained after this insertion is denoted by $T + Succ_i(G)$. In fact, we have

$$T + Succ_i(G) = (T_1, \ldots, T_{|G|}, Succ_i(G)).$$

Denote by $O_{i,l} \in \mathcal{O} \backslash G$ the new operation that is inserted in T, i.e. $O_{i,l} = Succ_i(G)$. Fix $J_j \in \mathcal{J} \backslash S$. We explain how to recursively obtain the starting times $\psi_j(T + O_{i,l})$ from $\psi_j(T)$. Firstly, the makespan is

$$C_{\max}(T + O_{i,l}) = \psi_i(T) + p_{i,l}.$$

The resulting recurrence relation is

$$\psi_j(T + O_{i,l}) = \begin{cases} C_{\max}(T + O_{i,l}), & m_{i,l} = M(Succ_j(G)), \\ \psi_j(T), & \text{otherwise.} \end{cases}$$

Now define the head $r_j(T)$ as the earliest starting time of operation $Succ_j(G)$ such that $T + Succ_j(G)$ is feasible. These heads $r_j(T)$ are strongly connected with $\psi_j(T)$ and are obtained as follows:

$$r_j(T) = \begin{cases} \psi_j(T), & x_j(T) = 1, \\ C_{\max}(T), & x_j(T) = 0, \end{cases}$$

where $x_j(T)$ is a binary variable defined as

$$x_j(T) = \begin{cases} 1, & \psi_j(T) + p(Succ_j(G)) > C_{\max}(T) \text{ or} \\ & \left[\psi_j(T) + p(Succ_j(G)) = C_{\max}(T) \text{ and } M(T_{|G|}) < M(Succ_j(G))\right], \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

The variable $x_j(T)$ represents whether the next operation can be added to the end of T without a delay. In fact, $x_j(T) = 1$ if and only if

$$T + Succ_j(G) = (T_1, \ldots, T_{|G|}, Succ_j(G))$$

is feasible.

In order to understand the basic notations, we give an example. The basic parameters of an instance are given in Table 2. Here each cell contains a pair (p_i^k, l) where l means that the lth operation of job J_i is processed on machine M_k.

Figure 1 illustrates the feasible sequence

$$T = (O_{3,1}, O_{1,1}, O_{2,1}, O_{1,2}, O_{3,2}, O_{2,2}, O_{1,3}, O_{4,1}, O_{1,4}).$$

This sequence is associated with the set of scheduled jobs $S = \{J_1\}$ and the set of scheduled operations

$$G = \{O_{1,1}, O_{1,2}, O_{1,3}, O_{1,4}, O_{2,1}, O_{2,2}, O_{3,1}, O_{3,2}, O_{4,1}\}.$$

The starting times of operations belonging to G are given in Table 3.

The parameters $Succ_j(G), x_j(T), \psi_j(T)$, and $r_j(T)$ are given in Table 4. In addition, $C_{\max}(T)$, the earliest starting times, and the heads are illustrated in Fig. 1. Note that $x_3(T) = 0$ since

$$\psi_3(T) + p(Succ_3(G)) = 9 + 4 < C_{\max}(T)$$

The next definition introduces the complete expansion of a sequence.

Table 2 Processing times and processing orders of an instance

Jobs	Machines			
	M_1	M_2	M_3	M_4
J_1	(1, 3)	(2, 1)	(6, 2)	(7, 4)
J_2	(10, 3)	(8, 1)	(5, 2)	(4, 4)
J_3	(4, 3)	(1, 4)	(5, 1)	(4, 2)
J_4	(5, 2)	(5, 1)	(5, 3)	(3, 4)

Table 3 The earliest starting times related to Fig. 1

(i, k)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(2, 1)	(2, 2)	(3, 1)	(3, 2)	(4, 1)
$\psi_{i,k}$	5	6	9	15	0	8	0	5	15

Table 4 Parameters related to the schedule depicted in Fig. 1

j	2	3	4
$Succ_j(G)$	$O_{2,3}$	$O_{3,3}$	$O_{4,2}$
$x_j(T)$	1	0	1
$\psi_j(T)$	13	9	20
$r_j(T)$	13	22	20

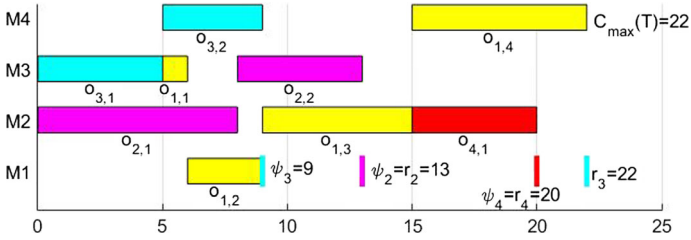


Fig. 1 Example of a schedule

Definition 3 Let T be the sequence of operations associated with $G \subset \mathcal{O}$. Let

$$T^* = (T_1^*, \ldots, T_{n \cdot m}^*)$$

be the sequence of operations associated with $\mathcal{O}$. We say that T^* is a complete expansion of T if $T_h = T_h^*$ $(h = 1, \ldots, |G|)$ and T^* is feasible.

We introduce the weak dominance '$\preceq$' relation as follows.

Definition 4 Let T^1 and T^2 be two sequences of operations such that both T^1 and T^2 are associated with $S \subset \mathcal{J}$. We say that T^1 weakly dominates T^2, i.e. $T^1 \preceq T^2$ if

$$r_i(T^1) \leq r_i(T^2)$$

for all $J_i \in \mathcal{J} \backslash S$.

When two sequences of operations, namely T^1 and T^2, are given, then the corresponding starting times will be denoted by $\psi^1(O)$ and $\psi^2(O)$, respectively. Hence $O \in \mathcal{O}$ belongs to T^1 and T^2. We are ready to present the theorem related to the weak dominance.

Theorem 1 *Let T^1 and T^2 be two sequences of operations such that both T^1 and T^2 are associated with $G \subset \mathcal{O}$. If T^1 weakly dominates T^2, then for all complete expansions of T^2 there exists a complete expansion of T^1 such that*

$$\psi^1(O) \leq \psi^2(O)$$

for all $O \in \mathcal{O} \backslash G$

In other words, Theorem 1 says that T^2 can be disregarded if $T^1 \preceq T^2$. The Proof of Theorem 1 will be given in Sect. 3.2 for the general case SDST-JSP (Theorem 3) where Theorem 1 is a special case.

The next definition introduces an indirect dominance.

Definition 5 Let T stands for the sequence of operations associated with $G \subset \mathcal{O}$ and $S \subset \mathcal{J}$. We say that T is indirectly dominated if there exists $M_k \in \mathcal{M}$ and $J_j \in \mathcal{J} \backslash S$ with $M(Succ_j(G)) = k$ such that the following two statements hold:

1. We have $x_j(T) = 0$.
2. For all $J_i \in \mathcal{J} \backslash S \backslash J_j$ with $M(Succ_i(G)) = k$ we have $r_i(T) \geq \psi_j(T) + p(Succ_j(G))$.

Informally speaking, we can add to T at least one operation $O \in \mathcal{O} \backslash G$ such that the heads remain the same for $T + O$. Thus, the sequences that are indirectly dominated can be disregarded. The next theorem formalizes this idea.

Theorem 2 *Let T be the sequence of operations associated with $G \subset \mathcal{O}$ and $S \subset \mathcal{J}$. If T is indirectly dominated, then there exists job $J_j \in \mathcal{J} \backslash S$ such that $x_j(T) = 0$ and*

$$r_i(T + Succ_j(G)) = r_i(T)$$

for all $J_i \in \mathcal{J} \backslash S \backslash J_j$.

The Proof of Theorem 2 directly follows from Definition 5. Theorem 2 is formulated in a slightly different way than the one proposed by Gromicho et al. (2012).

Now we investigate the previously studied example, see Fig. 1. Here, $x_3(T) = 0$. Moreover,

$$r_i(T) \geq \psi_3(T) + p(Succ_3(G)) = 13$$

for $i \in \{3, 4\}$. Thus, the schedule depicted in Fig. 1 is indirectly dominated.

We are ready to present the BDP algorithm, see Algorithm 1. This algorithm is similar to the one proposed by van Hoorn (2016). We have reformulated this

Algorithm 1: BDP algorithm.

```
   Input  : Job shop instance, upper bounds UB, window width H
   Output: Upper bound UB
1.1  Z({O_{i,1}}) = {(O_{i,1})}, i = 1,...,n
1.2  for t = 1 to n·m − 1 do
1.3      forall G ⊂ O with |G| = t do
1.4          forall T ∈ Z(G) do
1.5              forall O_{j,k} ∈ O \ G with x_j(T) = 1 and O_{j,k} = Succ_j(G) do
1.6                  T^1 = T + O_{j,k}
1.7                  G^1 = G ∪ {O_{j,k}}
1.8                  Z(G^1) ← filter_sequences(T^1, Z(G^1), UB)
1.9      if |Z|_{t+1} > H then
1.10         keep only H sequences according to some priority rule
1.11 New UB value is obtained from Z(O)
```

algorithm using our notation system. Algorithm 1 will also be useful for the SDST-JSP.

Now we briefly describe Algorithm 1. Initially, n sequences of operations are created. Then a loop through stages $t = 1$ to $n \cdot m - 1$ has been done, see line 1.2. For each stage t we loop through all subsets G with the size t (line 1.3). For each G, we loop through all sequences $T \in Z(G)$. These sequences are expanded with the operations that satisfy requirements given in line 1.5. Function *filter_sequences* (line 1.8) firstly tests whether T^1 is indirectly dominated. Then we try to reduce $Z(G^1)$ by keeping only non-dominated sequences of operations. If T^1 is not dominated by another sequence from $Z(G^1)$, then we test whether $LB(T^1) < UB$ holds. Finally, if T^1 is not discarding, then T^1 is included in $Z(G^1)$. If the state space is growing too large, then only H sequences with size $t + 1$ are kept (line 1.9). Also, Algorithm 1 becomes the heuristic BDP algorithm. The sorting criterion can be *LB* values.

In contrast to the previous version of the BDP (van Hoorn 2016), we use function *filter_sequences*, see line 1.8. We have two main contribution of this function. Firstly, we use a stronger lower *LB* than that one used by van Hoorn (2016). This bound will be described in Sect. 3.3. Secondly, if T^1 is indirectly dominated, then we do not test whether $T^1 \preceq T^2$ for another T^2. Thus, we do not need to save T^1 if T^1 is indirectly dominated.

Algorithm 2: Pseudocode for solving the job shop scheduling problem to optimality.

Input : Job shop instance, the upper bound UB_0
Output: *Opt* - the optimal makespan

2.1 $Z(\{O_{i,1}\}) = \{(O_{i,1})\}$, $i = 1, \ldots, n$
2.2 $Opt = BDP(1, Z, UB_0)$
2.3 $BDP(t, Z_t, UB)$/* the procedure for obtaining UB */
2.4 **begin**
2.5 $Z(G) = \emptyset$ for all $G \subset \mathcal{O}$ with $|G| = t + 1$
2.6 **forall** $G \subset \mathcal{O}$ *with* $|G| = t$ **do**
2.7 **forall** $T \in Z(G)$ **do**
2.8 $Z(G) = Z(G) \setminus \{T\}$
2.9 **forall** $O_{j,k} \in \mathcal{O} \setminus G$ *with* $x_j(T) = 1$ *and* $O_{j,k} = Succ_j(G)$ **do**
2.10 $T^1 = T + O_{j,k}$
2.11 $G^1 = G \cup \{O_{j,k}\}$
2.12 $Z(G^1) \leftarrow filter\_sequences(T^1, Z(G^1), UB)$
2.13 **if** $|Z|_{t+1} > H$ **then**
2.14 $Z_1 = Z$
2.15 $UB = BDP(t + 1, Z, UB)$
2.16 $UB = BDP(t, Z_1, UB)$
2.17 **if** $Z(\mathcal{O}) \neq \emptyset$ **then**
2.18 New UB is obtained from $Z(\mathcal{O})$
2.19 **return** UB

Algorithm 2 obtains the optimal solution of the JSP. This algorithm is based on Algorithm 1. However, lines 1.9–1.10 are replaced by lines 2.13–2.16. In this case

the subproblem (line 2.16) is solved if the state space is growing too large. The parameter H (line 2.13 in Algorithm 2) depends on the computer system requirements.

In summary, the main contributions of Algorithms 1 and 2 are summarized below.

1. A stronger lower bound LB is used. This bound is described is Sect. 3.3. It can be expected that more sequences T will be discarded.
2. We do not have to save those sequences that are indirectly dominated. Thus, less memory could be required during the run of the BDP algorithm.
3. We have included lines 2.13–2.16 in Algorithm 2. Thus, the given algorithm is theoretically able to solve any instance without running out of the given amount of memory.

3.2 Bounded dynamic programming algorithm for the SDST-JSP

Again, let T be the sequence of operations. Throughout this section we will assume that T is associated with $S \subset \mathcal{J}$ and $G \subset \mathcal{O}$.

Some additional notations will be introduced. Let $\rho_k(T)$ denotes the job index that corresponds to the last operation completed on machine $M_k \in \mathcal{M}$. The completion time of this operation is denoted by $C_k(T)$. If no operation is scheduled on machine M_k, then $C_k(T) = 0$.

Table 5 reports $\rho_k(T)$ and $C_k(T)$ values for the previously studied example (see Table 2 and Fig. 1). In this example we assume that all setup times are equal to 0.

Now we will analyse the transition from T to $T + O_{i,l}$ where $O_{i,l} \in \mathcal{O}\backslash G$ is the next operation that job $J_i \in \mathcal{J}\backslash$ has to schedule. Let $k = m_{i,l}$ denotes the machine index of operation $O_{i,l}$. Firstly, the makespan is equal to

$$C_{\max}(T + O_{i,l}) = \psi_i(T) + p_{i,l}.$$

Secondly, the earliest starting times $\psi_i(T + O_{i,l})$ can be greater than $C_{\max}(T + O_{i,l})$, i.e.

$$\psi_i(T + O_{i,l}) = \max\left\{C_{\max}(T + O_{i,l}), C_k(T) + s^k_{\rho_k(T),i}\right\}.$$

Also, setup times have to be taken into account in order to obtain $\psi_j(T + O_{i,l})$ for $J_j \in \mathcal{J}\backslash S\backslash\{J_i\}$ with $M(Succ_j(G)) = k$. For this case we have

$$\psi_j(T + O_{i,l}) = C_{\max}(T + O_{i,l}) + s^k_{\rho_k(T),j}.$$

Table 5 Parameters $\rho_k(T)$ and $C_k(T)$ related to the schedule depicted in Fig. 1

k	1	2	3	4
$\rho_k(T)$	1	4	2	1
$C_k(T)$	9	20	13	22

The resulting recurrence relation is

$$\psi_j(T+O_{i,l}) = \begin{cases} \max\left\{C_{\max}(T+O_{i,l}), C_k(T)+s^k_{\rho_k(T),j}\right\}, & i=j, \\ C_{\max}(T+O_{i,l})+s^k_{\rho_k(T),j}, & M(Succ_j(G))=k, \\ \psi_j(T), & \text{otherwise.} \end{cases}$$

where index j is such that $J_j \in \mathcal{J}\backslash S$.

The heads $r_j(T)$ can be calculated as follows:

$$r_j(T) = \begin{cases} \psi_j(T), & x_j(T)=1, \\ \max\left\{C_{\max}(T), C_k(T)+s^k_{\rho_k(T),j}\right\}, & x_j(T)=0, \end{cases}$$

where $x_j(T)$ is the binary variable defined in (1). Note that $r_j(T)$ can be greater than $C_{\max}(T)$ due to the setup times.

Let us describe the general heads $r_{j,g}(T)$. These heads represent the earliest starting time of $O_{j,g} \in \mathcal{O}\backslash G$. Let $k = m_{j,g}$ denotes the machine index of $O_{j,g}$. In general, $r_{j,g}(T)$ can recursively be estimated as follows:

$$r_{j,l}(T) = r_j(T),$$
$$r_{j,g}(T) = \max\left\{r_{j,g-1}(T)+p_{j,g-1}, C_k(T)+s^k_{\rho_k(T),j}\right\}, \quad g = l+1,\ldots,m,$$

where index l means that $Succ_j(G) = O_{j,l}$.

Theorem 1 about the weak dominance cannot directly be applied to the SDST-JSP due to the setup times. Two simple examples are given in Figs. 2 and 3. These examples show that Theorem 1 fails in these cases with unit processing times. The setup times corresponding to Fig. 2 are

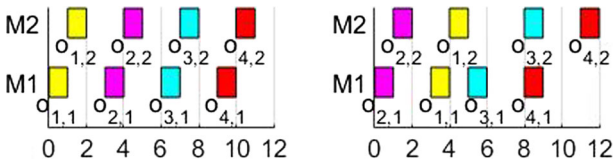
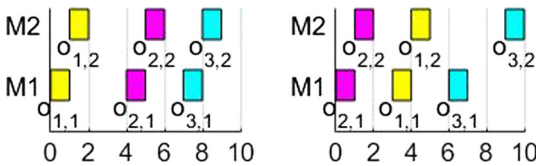


Fig. 2 On the left side: the optimal schedule. On the right side: the non-optimal schedule

Fig. 3 On the left side: the optimal schedule. On the right side: the non-optimal schedule



$$\left(s^1_{i,j}\right) = \begin{pmatrix} 0 & 2 & 1 & 3 \\ 2 & 0 & 2 & 3 \\ 3 & 3 & 0 & 2 \\ 3 & 3 & 3 & 0 \end{pmatrix} \text{ and } \left(s^2_{i,j}\right) = \begin{pmatrix} 0 & 2 & 3 & 4 \\ 2 & 0 & 2 & 4 \\ 4 & 4 & 0 & 2 \\ 4 & 4 & 4 & 0 \end{pmatrix}.$$

For the second example depicted in Fig. 3, we define the setup times as follows:

$$\left(s^1_{i,j}\right) = \begin{pmatrix} 0 & 3 & 2 \\ 2 & 0 & 2 \\ 4 & 4 & 0 \end{pmatrix} \text{ and } \left(s^2_{i,j}\right) = \begin{pmatrix} 0 & 3 & 4 \\ 2 & 0 & 2 \\ 4 & 4 & 0 \end{pmatrix}.$$

In the first case, we set

$$T^1 = (O_{1,1}, O_{1,2}, O_{2,1}, O_{2,2}, O_{3,1}, O_{3,2}),$$
$$T^2 = (O_{2,1}, O_{2,2}, O_{1,1}, O_{1,2}, O_{3,1}, O_{3,2}).$$

Then

$$r_4(T^1) = 9 > r_4(T^2) = 8.$$

However,

$$C_{\max}(T^1 + O_{4,1} + O_{4,2}) = 11 < C_{\max}(T^2 + O_{4,1} + O_{4,2}) = 12.$$

In this case, the problem lies in the fact that the completion times of operation $O_{3,2}$ on the machine M_2 differs, i.e. $C_2(T^1) < C_2(T^2)$.

Now we study the example depicted in Fig. 3. In this case, we set

$$T^1 = (O_{1,1}, O_{1,2}, O_{2,1}, O_{2,2}),$$
$$T^2 = (O_{2,1}, O_{2,2}, O_{1,1}, O_{1,2}).$$

Then

$$r_3(T^1) = 7 > r_3(T^2) = 6$$

and

$$C_{\max}(T^1 + O_{3,1} + O_{3,2}) = 9 < C_{\max}(T^2 + O_{3,1} + O_{3,2}) = 10.$$

In this case, $\rho_2(T^1) \neq \rho_2(T^2)$.

Counterexamples in Figs. 2 and 3 show that Theorem 1 cannot be applied to the SDST-JSP. Thus, a stronger dominance rule has to be used. The next definition introduces this rule.

Definition 6 Let T^1 and T^2 be two sequences of operations such that both T^1 and T^2 are associated with $G \subset \mathcal{O}$ and $S \subset \mathcal{J}$. We say that $T^1 \preceq T^2$ if the following two statements hold:

1. $r_j(T^1) \leq r_j(T^2)$ for all $J_j \in \mathcal{J} \backslash S$.

2. Let i denotes the job index for which $i = \rho_k(T^1)$ and let l denotes the index with $k = m_{j,l}$. If $i \neq \rho_k(T^2)$ or $C_k(T^1) > C_k(T^2)$, then

$$C_k(T^1) + s^k_{i,j} \leq r_{j,l}(T^2)$$

for all $J_j \in \mathcal{J}\backslash S$.

Note that Definition 6 reduces to Definition 4 if $s^k_{i,j} = 0$ for all $i,j \in \{1,\ldots n\}$ and $k \in \{1,\ldots,m\}$.

Theorem 3 *Let T^1 and T^2 be two sequences of operations such that both T^1 and T^2 are associated with $G \subset \mathcal{O}$. If $T^1 \preceq T^2$, then for all complete expansions of T^2 there exists a complete expansion of T^1 such that*

$$\psi^1(O) \leq \psi^2(O)$$

for all $O \in \mathcal{O}\backslash G$

Proof Let T^4 denotes the complete expansion of T^2. Thus, we have

$$T^4 = (T^2_1,\ldots,T^2_{|G|},T_{|G|+1},\ldots,T_{n\cdot m}).$$

Further, let T^3 denotes an expansion of T^1. We assume that all operations from $\mathcal{O}\backslash G$ are in the same order as in T^4, i.e.

$$T^3 = (T^1_1,\ldots,T^1_{|G|},T_{|G|+1},\ldots,T_{n\cdot m}).$$

By reordering operations from T^3, we can obtain a feasible sequence of operations.

Let O_{j,l_1} denotes the next operation $T_{|G|+1}$ and let k denotes the machine index of O_{j,l_1}. From Condition 1 in Definition 6 it follows that $\psi^3(O_{j,l_1}) \leq \psi^4(O_{j,l_1})$. The same can be stated for all other operations T_h ($h > |G| + 1$) for which the following two statements hold:

- $M(T_g) \neq M(T_h)$;
- The job index between operations T_g and T_h are different

for all $g \in \{|G| + 1,\ldots,h-1\}$.

Let T_h ($h > |G|$) denotes the first operation for which there exists previous operation T_g that has to be scheduled on the same machine or by the same job as operation T_h. Let $O_{i,l_2} = T_g$ denotes this operation.

Now we will study two cases. For the case $M(T_h) = M(T_g)$, we have

$$\psi^3(T_h) = \psi^3(T_g) + p(T_g) + s^k_{i,j}, \tag{2}$$

$$\psi^4(T_h) = \psi^4(T_g) + p(T_g) + s^k_{i,j}. \tag{3}$$

Since $\psi^3(T_g) \leq \psi^4(T_g)$, it follows from (2)–(3) that $\psi^3(T_h) \leq \psi^4(T_h)$. For the case $i = j$, we have $O_{i,l_2} = O_{j,l_1-1}$. If no operation from $\mathcal{O}\backslash G$ has been scheduled on

machine M_k, then Condition 2 in Definition 6 ensures that $\psi^3(O_{j,l_1}) \leq \psi^4(O_{j,l_2})$ since $\psi^3(O_{j,l_1-1}) \leq \psi^4(O_{j,l_1-1})$. Otherwise, we have

$$\rho_k((T_1^3, \ldots, T_{h-1}^3)) = \rho_k((T_1^4, \ldots, T_{h-1}^4)),$$
$$C_k((T_1^3, \ldots, T_{h-1}^3)) \leq C_k((T_1^4, \ldots, T_{h-1}^4)),$$

from which it follows that $\psi^3(T_h) \leq \psi^4(T_h)$

In summary, we have proved $\psi^3(T_g) \leq \psi^4(T_g)$ for all $g \in \{|G|+1, \ldots, h\}$. Hence, T_h is the first operation for which there exists previous operation that has to be scheduled on the same machine or by the same job. By repeating the same steps of the proof, we can recursively prove that $\psi^4(T_g) \leq \psi^4(T_g)$ for all $g \in \{|G|+1, \ldots, n \cdot m\}$. By reordering the operations from sequence T^3 such that precedence relations among operations remain, we can obtain feasible sequence. This sequence is the complete expansion of T^3 such that

$$\psi^3(O) \leq \psi^4(O)$$

for all $O \in \mathcal{O} \backslash G$. The proof is completed. □

Definition 5, which introduces the indirect dominance, has to be generalized.

Definition 7 Let T stands for the sequence of operations associated with $G \subset \mathcal{O}$ and $S \subset \mathcal{J}$. We say that T is indirectly dominated if there exists $M_k \in \mathcal{M}$ and $J_j \in \mathcal{J} \backslash S$ with $M(Succ_j(G)) = k$ such that the following two statements hold:

1. We have $x_j(T) = 0$.
2. For all $J_i \in \mathcal{J} \backslash S \backslash J_j$ with $M(Succ_i(G)) = k$ we have $r_i(T) \geq \psi_j(T) + p(Succ_j(G)) + s_{j,i}^k$.

Theorem 4 *Let T be the sequence of operations associated with $G \subset \mathcal{O}$ and $S \subset \mathcal{J}$. If T is indirectly dominated, then there exists job $J_j \in \mathcal{J} \backslash S$ such that $x_j(T) =$ 0 and*

$$r_i(T + Succ_j(G)) = r_i(T)$$

for all $J_i \in \mathcal{J} \backslash S \backslash J_j$.

The Proof of Theorem 4 immediately follows from Definition 7.

3.3 A lower bound for the JSP

Throughout this section we assume that all operations from $G \subset \mathcal{O}$ are scheduled. The set of scheduled jobs is denoted by S. Let T denotes the sequence of operations associated with S and G. Let $q_{j,k}$ be a tail, which is defined as a lower bound of the time period between the completion time of operation $O_{j,k} \in \mathcal{O} \backslash G$ and the optimal makespan $C_{\max}$. The tails $q_{j,k}$ are calculated as follows:

$$q_{j,k} = \sum_{h=k+1}^{m} p_{j,h}. \tag{4}$$

Let $r_{j,k}(T)$ denotes the earliest starting time of operation $O_{j,k} \in \mathcal{O}\backslash G$. These starting times can be estimated as follows:

$$r_{j,k}(T) = r_j(T) + q_{j,l} - q_{j,k-1}, \tag{5}$$

where index l is such that $O_{j,l+1} = Succ_j(G)$ holds.

Fix machine $M_k \in \mathcal{M}$. Consider the one-machine scheduling problem with heads $r_{j,l}(T)$ and the corresponding tails where $O_{j,l} \in \mathcal{O}$ and $M(O_{j,l}) = k$. These heads and tails are defined by (5) and (4) respectively. Let $LB_k(T)$ be the optimal solution of this one-machine sequencing problem. Despite the NP-hardness, the problem can efficiently be solved in practice using the branch and bound method proposed by Carlier (1982). The final lower bound of T is

$$LB(T) = \max_{k\in\{1,\ldots,m\}} LB_k(T).$$

3.4 A lower bound for the SDST-JSP

Again, we assume that all operations from $G \subset \mathcal{O}$ are scheduled and all jobs from $S \subset \mathcal{J}$ are completed. Let T stands for the sequence of operations associated with G and S.

The computation of the lower bounds can be reduced to the traveling salesman problem with time windows (TSPTW). Fix machine $M_k \in \mathcal{M}$. For the sake of simplicity, denote by δ^k the following set:

$$\delta^k = \{i \mid \text{ exists } l \text{ with } O_{i,l} \in \mathcal{O}\backslash G \text{ and } M(O_{i,l}) = k\}$$

For all $j \in \delta^k$ define the time windows $[a_j, b_j]$ as follows:

$$a_j = r_{j,l}(T) + p_{j,l},$$
$$b_j = UB - q_{j,l} - 1,$$

where l is such that $O_{j,l} \in \mathcal{O}\backslash G$.

The costs $c_{i,j}$ for $i, j \in \delta^k$ with $i \neq j$ are defined as

$$c_{i,j} = s^k_{i,j} + p_{j,l},$$

where l is such that $O_{j,l} \in \mathcal{O}\backslash G$. The cost $c_{i,j}$ includes both the service time of i and the time needed to travel from i to j. The feasible version of the TSPTW denoted by F-TSPTW asks to find the feasible schedule satisfying time windows constraints. It can be concluded that $LB \geq UB$ if there exists machine $M_k \in \mathcal{M}$ such that the solution of F-TSPTW is infeasible.

Obviously, the problem F-TSPTW is NP-hard since it is the generalization of the TSP without the time windows. Similarly as by Artigues and Feillet (2008), we use

the dynamic programming algorithm proposed by Feillet et al. (2004). This algorithm solves the elementary shortest path problem with resource constraints (ESPPRC) where TSPTW can be interpreted as a special case of the ESPPRC.

Algorithm 3: Algorithm that finds whether or not $LB < UB$.

Input : $[a_i, b_i]$ - time windows, $(c_{i,j})$ - cost matrix
Output: success or failure
3.1 Find feasible solution using some heuristics
3.2 **if** *solution is found* **then**
3.3 **return** success
3.4 Estimate LB using some polynomial time algorithm
3.5 **if** $LB \geq UB$ **then**
3.6 **return** failure
3.7 **return** F_TSPTW(a,b,c)

We try to speed up the calculation due to the NP-hardness of the problem F-TSPTW. The general scheme of the algorithm is given in Algorithm 3. The first step in Algorithm 3 is to find a feasible solution of the TSPTW using a time efficiency heuristic (line 3.1). In this paper, we use a variable neighborhood search (VNS) heuristic proposed by Da Silva and Urrutia (2010). Papalitsas et al. (2015) empirically shown that the sorting-based approach is not better than the random-based procedure. However, the VNS algorithm modified by Papalitsas et al. (2015) shows better performance.

If a feasible solution is not found immediately, then the weaker lower bound *LB* is calculated (line 3.5). We apply the calculation of *LB* given by Brucker and Thiele (1996). Let $h(\delta^k)$ be defined as

$$h(\delta^k) = \min_{i \in \delta^k,\, m_{i,l}=k} r_{i,l}(T) + \sum_{i \in \delta^k,\, m_{i,l}=k} p_{i,l} + \min_{i \in \delta^k,\, m_{i,l}=k} q_{i,l} + setup_{\min}(\delta^k). \quad (6)$$

The value $setup_{\min}(\delta^k)$ in (6) denotes a solution of the TSP with $|\delta^k|$ vertices and setup times (costs) $s^k_{i,j}$ between those jobs J_i, J_j for which there exists unscheduled operation on machine M_k. The values $setup_{\min}(\delta^k)$ for all δ^k can be preprocessing at the beginning of Algorithm 3. Then $h(\delta^k)$ is calculated for the new δ^k that is obtained by deleting one operation from the previous δ^k. We choose these operations according to the order of non-decreasing heads until δ^k is empty. Then we repeat the same procedure but now according to the order of non-decreasing tails. LB_k is obtained by taking the maximum value among all $h(\delta^k)$ values.

Finally, if the result is not obtained in the previous two steps, then the relaxation of the ESPRC algorithm given by Feillet et al. (2004) has to be solved, see Algorithm 4. The value $dist(V, i)$ denotes the shortest path for a pair (V, i) where $i \in V$ is the current vertex and $V \subset \delta^k$ is the set of vertices already visited. The sets Λ_i and Λ_i^{new} consist of the sets V for which $dist(V, i) \neq \infty$. The reachability test (line 4.9 in Algorithm 4) is the same as the one proposed by Artigues and Feillet (2008).

Algorithm 4: The relaxation of ESPRC algorithm proposed by Feillet et al (2004).

```
4.1  F_TSPTW(a,b,c)
4.2  begin
       Initialization: dist({i}, i) = a_i
       dist(V, i) = ∞ for V ⊂ δ^k with |V| > 1
       Λ_i = {{i}}, i ∈ δ^k
4.3    for k = 2 to |δ^k| do
4.4      Λ_i^new = ∅, i ∈ δ^k
4.5      forall i ∈ δ^k do
4.6        forall V ∈ Λ_i do
4.7          forall j ∈ δ^k \ V do
4.8            F_{i,j} = max{dist(V, i) + c_{i,j}, a_j}
4.9            if F_{i,j} ≤ dist(V ∪ {j}, j) and all nodes reachable from
                 j with F_{i,j} then
4.10             Λ_j^new = Λ_j^new ∪ {V ∪ {j}}
4.11             keep non-dominated vertices in Λ_j^new
4.12     Λ = Λ^new
```

4 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. The presented computation results will be in two directions: to solve the problem without knowing a lower and upper bound, and the optimality proof by taking *UB* equals the optimal solution. For the JSP we will only prove the optimality with the aim to show the dimension of benchmark instances that can be solved by the BDP algorithm. For the SDTS-JSP we will try to solve the instances to optimality using two strategies.

Section 4.1 presents computational results for the JSP. In Sect. 4.2, the SDST-JSP is studied.

4.1 Case JSP

In this section, we will analyse how proposed Algorithm 2 works on practice proving the optimality for different benchmark instances. The effectiveness will be analysed in terms of |Z| and CPU times. Hence, |Z| is defined as

$$|Z| = \sum_{G \subset \mathcal{O}} |Z(G)|.$$

In other words |Z| is a number of the non-dominated sequences T for which $LB(T) < UB$.

The termination criterion of Algorithm 2 is chosen in the following way. The calculation is interrupted if $|Z|_{t+1} > H$ for the subproblem (line 2.16 in Algorithm 2).

Table 6 shows that the optimality is proven for all $10 \times 5, 10 \times 10, 15 \times 10, 30 \times 10$ type benchmark instances. However, the BPP algorithm is able to solve only 5 out of 10 benchmark instances for the case 20×10 and 3 out of 5 instances for the case 15×15. Furthermore, some 20×10 type instances were solved because of the good estimation of the lower bound. Thus, the dominance relation between sequences was not important for those 20×10 type instances as well as for the cases $15 \times 5, 20 \times 5$, and 30×10. Finally, the BDP algorithm is not able to solve any 20×15 type instance as well as instances with a larger dimension. We observe from Table 6 that all 10×10 type instances were solved within a CPU time less than 1 min. CPU times varies from seconds up to 4 h for 15 jobs and 10 machines.

The memory would be overreached for instances la21, la25, and la40 if the termination criterion has not be used (line 2.13 in Algorithm 2).

van Hoorn (2016) gives computational results for his version of the BDP algorithm where the optimality was proven only for instances up to 10 jobs. Table 7 compares CPU times with those reported by van Hoorn (2016). We observe that almost all instances were solved significantly faster, except for instances with CPU time zero second. Moreover, van Hoorn algorithm is able to prove the optimality only for instances with a maximum of 10 jobs. It can be concluded that the BDP algorithm proposed in this paper significantly increases the applicability of the BDP approach for solving the classical JSP.

4.2 Case SDST-JSP

Table 11 reports the calculation of UB values whereas Table 8 gives a summary. In this case Algorithm 1 is used where H is taken as 100, 1000, ... and finally 1,000,000 if it is necessary. Memory requirements of the computer allow us to use $H = 10{,}000{,}000$. However, then CPU times would be increased significantly. For larger instances t2-ps11 to t2-ps15, $H = 1000$ is the last time window. We start the calculation by setting $H = 100$ and $UB = \infty$. Then we try to iteratively improve UB value until no improvement is possible for $H = 100$. Then we set $H = 1000$ and repeat the same procedure, etc. In Table 8, the three columns with the title 'Summary' represent final results obtained for each instance. The calculations are terminated when the optimal value is reached or the maximal H value is reached. The relative percentage deviation (RPD) is calculated as follows:

$$RPD = \frac{UB_{final} - BKS}{BKS} \times 100$$

where BKS denotes the best-known solution previously reported in the literature.

As seen in Table 8, the optimality can be proven within few seconds for small instances of type $5 \times 10 \times 5$. CPU times significantly grow for medium instances (named t2-ps06 to t2-ps10). Computational times are huge compared to those obtained by the state-of-the-art heuristic methods, e.g. a genetic algorithm combined

Table 6 The optimality proof for each instance

Instance	$n \times m$	Opt	|Z|	CPU (s)
ft06	6×6	55	6	0
ft10	10×10	930	1,505,084	14
ft20	20×5	1165	41	0
la01	10×5	666	10	0
la02	10×5	655	10	0
la03	10×5	597	12	0
la04	10×5	590	4566	0
la05	10×5	593	10	0
la06	15×5	926	15	0
la07	15×5	890	15	0
la08	15×5	863	15	0
la09	15×5	951	15	0
la10	15×5	958	15	0
la11	20×5	1222	20	0
la12	20×5	1039	20	0
la13	20×5	1150	20	0
la14	20×5	1292	20	0
la15	20×5	1207	20	0
la16	10×10	945	2,255,158	18
la17	10×10	784	381,977	3
la18	10×10	848	360,939	3
la19	10×10	842	534,988	3
la20	10×10	902	268,916	2
la21	15×10	1046	577,830,315	16,297
la22	15×10	927	4,092,387	90
la23	15×10	1032	15	0
la24	15×10	935	10,004,379	178
la25	15×10	977	606,558,571	9553
la26	20×10	1218	20	0
la27	20×10	1235	20	0
la28	20×10	1216	20	0
la30	20×10	1355	20	0
la31	30×10	1784	30	0
la32	30×10	1850	30	0
la33	30×10	1719	30	0
la34	30×10	1721	30	0
la35	30×10	1888	30	0
la37	15×15	1397	23	0
la39	15×15	1233	39,828,979	653
la40	15×15	1222	104,054,7408	18,204
orb01	10×10	1059	1,057,741	9
orb02	10×10	888	1,479,717	10

Table 6 continued

Instance	$n \times m$	Opt	$\|Z\|$	CPU (s)
orb03	10×10	1005	5,372,642	53
orb04	10×10	1005	1,112,160	10
orb05	10×10	887	2,330,397	17
orb06	10×10	1010	1,383,871	13
orb07	10×10	397	228,073	2
orb08	10×10	899	130,560	1
orb09	10×10	934	1,307,918	11
orb10	10×10	944	34,683	0
abz05	10×10	944	1,434,865	11
abz06	10×10	1234	78,464	1
swv02	20×10	1475	20	0

Table 7 Comparison of CPU times with the previous results of the BDP algorithm

Instance	$n \times m$	Opt	CPU (s)	
			Algorithm 2	van Hoorn algorithm[a]
ft06	6×6	55	0	0
ft10	10×10	930	14	60
la01	10×5	666	0	25
la02	10×5	655	0	1
la03	10×5	597	0	0
la04	10×5	590	0	0
la05	10×5	593	0	466
la16	10×10	945	18	44
la17	10×10	784	3	1
la18	10×10	848	3	14
la19	10×10	842	3	9
la20	10×10	902	2	3
orb01	10×10	1059	9	36
orb02	10×10	888	10	29
orb03	10×10	1005	53	185
orb04	10×10	1005	10	20
orb05	10×10	887	17	29
orb06	10×10	1010	13	40
orb07	10×10	397	2	8
orb08	10×10	899	1	9
orb09	10×10	934	11	14
orb10	10×10	944	0	1
abz05	10×10	944	11	42
abz06	10×10	1234	1	3

[a]Algorithm proposed by van Hoorn (2016)

Table 8 Upper bounds: case SDST-JSP

Inst.	$n \times m$	Best lit. UB	Summary		
			Best found	RPD	CPU
t2-ps01	10×5	**798***	**798***	0	3
t2-ps02	10×5	**784***	**784***	0	0
t2-ps03	10×5	**749***	**749***	0	6
t2-ps04	10×5	**730***	**730***	0	0
t2-ps05	10×5	**691***	**691***	0	0
t2-ps06	15×5	**1009***	**1009***	0.00	6707
t2-ps07	15×5	**970***	**970***	0.00	62,334
t2-ps08	15×5	**963***	965	0.21	22,388
t2-ps09	15×5	**1060***	**1060***	0.00	1875
t2-ps10	15×5	**1018***	**1018***	0.00	456
t2-ps11	20×5	**1438**	1522	5.84	9370
t2-ps12	20×5	**1269**	1358	7.01	36,649
t2-ps13	20×5	**1406**	1439	2.35	10,058
t2-ps14	20×5	**1452**	1540	6.06	1214
t2-ps15	20×5	**1485**	1551	4.44	3108

Values in bold: best-known result is reached

Asterisk '*': upper bound is equal to optimal solution

with a tabu search or a local search (González et al. 2008, 2009). However, the BDP algorithm is able to reach the optimal value for those instances except for t2-ps08 for which the H value should be increased to obtain the optimal solution. The optimality has not been reached for instances t2-ps11 to t2-ps15 with $n = 20$ jobs. However, the proposed algorithm can be applied for improving the best-known lower bounds for harder instances.

The main contribution of the current research for the SDST-JSP is associated with the computational results for large instances. In Table 12 we calculate lower bounds using the following strategy. We start with some initial bound. We replace UB in Algorithm 2 with the current bound. 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 the current value. Thus, the current bound is increased by one for instances t2-ps1 to t2-ps10. For larger instances t2-ps11 to t2-ps15, the increment is taken as five. If the solution is obtained for the current bound, then the previous bound (namely $\mathrm{LB}_{\mathrm{fin}}$) is also the optimal value of the given instance. If there exists t with $|Z|_t > 3 \cdot 10^5$, then the run is terminated. Thus, the optimal solution is not found and the previous bound is taken as the final lower bound of the given instance.

Table 9 gives a summary of lower bounds obtained by the DP. The value $\sum |Z|$ in Table 9 stands for the sum of all $|Z|$ that is obtained for all previous bounds. $\mathrm{LB}_{\mathrm{fin}}$ denotes the best lower bound that is found by BDP algorithm. The initial lower bound LB_0 is set significantly less in our runs. For the sake of simplicity we report only the final six bounds in Table 12.

Table 9 Lower bounds: case SDST-JSP

Inst.	$n \times m$	Best lit.	Summary			
		LB	LB	UB	$\sum \lvert Z \rvert$	CPU
t2-ps01	10×5	**798***	**798***	**798***	41,600	3
t2-ps02	10×5	**784***	**784***	**784***	45,792	3
t2-ps03	10×5	**749***	**749***	**749***	227,583	13
t2-ps04	10×5	**730***	**730***	**730***	8260	1
t2-ps05	10×5	**691***	**691***	**691***	3235	0
t2-ps06	15×5	**1009***	**1009***	**1009***	7,102,055	1744
t2-ps07	15×5	**970***	**970***	**970***	718,337	287
t2-ps08	15×5	**963***	**963***	**963***	5,402,280	6825
t2-ps09	15×5	**1060***	**1060***	**1060***	9,107,100	7227
t2-ps10	15×5	**1018***	**1018***	**1018***	8,856,013	1646
t2-ps11	20×5	1395	**1402**	–	0	15
t2-ps12	20×5	1242	**1249**	–	1,668,701	74,265
t2-ps13	20×5	1342	**1364**	–	4,738,581	186,625
t2-ps14	20×5	1432	**1442**	–	5,923,970	29,028
t2-ps15	20×5	1406	**1426**	–	2,550,037	245,084

Values in bold: best known result is reached

Asterisk: bound is equal to optimal solution

Value underlined: previously best known result is improved

As it is shown in Table 12, all small and medium instances t2-ps01 to t2-ps10 are solved to optimally. The best-known lower bounds are improved for all $5 \times 20 \times 10$ type benchmark instances. However, the proposed algorithm is not able to obtain the optimal solution without overreaching the width $3 \cdot 10^5$. CPU times varies from less than 1 day to almost 3 days. In practice CPU times below the column 'LB_{fin}' can be significantly less than those below the column '$LB_{fin}+1$' (e.g. the instances t2-ps07 and t2-ps10). On the other hand, for some cases these values can be similar (e.g. the instances t2-ps06 and t2-ps09).

It is interesting to note that a higher $|Z|$ value not always means that a CPU value will be higher. The highest $\sum |Z|$ value is for the instance t2-p14. However, this instance is solved in 8 h that is significantly faster than the time required to solve three other instances, namely t2-ps12, t2-ps13, and t2-ps15.

Now we compare two strategies for solving the SDST-JSP by BDP (see Tables 8 and 9). We have that the first approach is slightly better for small instances in terms of CPU times. For medium instances, the second strategy shows better performance on average. The instance t2-ps06 is solved 200 times faster by the second strategy and for the instance t2-ps07 the optimal solution is not even reached by the first strategy.

The computational results of the state-of-the-art exact algorithms are summarized in Table 10. BT96, ABF04, AF08, and BDP stand for computational results obtained by Brucker and Thiele (1996), Artigues et al. (2004), Artigues and Feillet (2008), and our BDP approach, respectively. For the BDP we have shown the CPU_{LB} times for the second strategy, see Tables 9 and 12. The results under AF08 are obtained by similar computer requirements as used in the current work. 2 GHz processor is used for ABF04. Unfortunately, it is difficult to compare our results to

Table 10 Comparison versus state-of-the-art exact algorithms: case SDST-JSP

Inst.	$n \times m$	BT96			ABF04		
		LB	UB	CPU	LB	UB	CPU
t2-ps01	10 × 5	756	**798***	502	**798***	**798***	522
t2-ps02	10 × 5	705	**784***	158	**784***	**784***	8
t2-ps03	10 × 5	658	**749***	1892	**749***	**749***	48
t2-ps04	10 × 5	627	**730***	189	**730***	**730***	34
t2-ps05	10 × 5	653	**691***	770	**691***	**691***	30
t2-ps06	15 × 5	986	–	7200	996	1026	7200
t2-ps07	15 × 5	940	–	7200	**970***	**970***	16,650
t2-ps08	15 × 5	913	–	7200	923	1002	7200
t2-ps09	15 × 5	1001	–	7200	1037	**1060***	7200
t2-ps10	15 × 5	1008	–	7200	**1018***	**1018***	498
t2-ps11	20 × 5	1322	–	7200	–	–	–
t2-ps12	20 × 5	1139	–	7200	1159	1319	7200
t2-ps13	20 × 5	1250	–	7200	1250	1439	7200
t2-ps14	20 × 5	1402	–	7200	–	–	–
t2-ps15	20 × 5	1307	–	7200	–	–	–

Inst.	$n \times m$	AF08				BDP			
		LB	UB	CPU_{LB}	CPU_{UB}	LB	UB	CPU_{LB}	CPU_{UB}
t2-ps01	10 × 5	**798***	**798***	57		**798***	**798***	3	
t2-ps02	10 × 5	**784***	**784***	105		**784***	**784***	3	
t2-ps03	10 × 5	**749***	**749***	353		**749***	**749***	13	
t2-ps04	10 × 5	**730***	**730***	16		**730***	**730***	1	
t2-ps05	10 × 5	**691***	**691***	16		**691***	**691***	0	
t2-ps06	15 × 5	**1009***	**1009***	1798		**1009***	**1009***	1744	
t2-ps07	15 × 5	**970***	**970***	782		**970***	**970***	287	
t2-ps08	15 × 5	**963***	**963***	349,923	22,717	**963***	**963***	6825	
t2-ps09	15 × 5	1051	1061	169,582	26,755	**1060***	**1060***	7227	
t2-ps10	15 × 5	**1018***	**1018***	35	46,590	**1018***	**1018***	1646	
t2-ps11	20 × 5	1395	1494	916,833	39,489	**1402**	1522	15	9370
t2-ps12	20 × 5	1242	1381	914,086	26,678	**1249**	1358	74,266	36,649
t2-ps13	20 × 5	1342	1457	895,059	50,336	**1364**	1439	186,626	10,058
t2-ps14	20 × 5	1432	1483	306,899	37,236	**1442**	1540	29,029	1214
t2-ps15	20 × 5	1406	1661	792,196	46,590	**1426**	1551	245,085	3108

Values in bold: best known result is reached

Asterisk: bound is equal to optimal solution

Value underlined: previously best known result is improved

those obtained by Brucker and Thiele (1996). Thus, we provide only a rough comparison with BT96. We apply factor 1.5 and 15 for CPU time comparisons with ABF04 and BT96, respectively.

CPU_{LB} and CPU_{UB} denote CPU times necessary to obtain LB and UB respectively. Some table cells under column CPU_{LB} are empty thus meaning that CPU=CPU_{LB}, i.e. LB and UB are obtained in a single run. Artigues and Feillet (2008) uses two slightly different strategies for their branch and bound method. Thus, for each instance the best-obtained results are reported in Table 10.

Table 10 shows that our CPU times outperforms all the state-of-the-art exact algorithms for 10 instances that our approach is able to solve to optimality. The only exception is instance t2-ps06 where the results of AF08 show similar performance. Especially good performance is observed for five smallest BT instances where the BDP algorithm can be competitive even with the heuristic methods. Even using factor 15 for CPU times, the BDP method performs much better than BT96. In fact, for five small instances the average CPU time of the BDP are almost 200 times less than the one of BT96.

All medium instances t2-ps06 to t2-ps10 are solved in a reasonable time limit. This cannot be said about AF08 method, which spends several days to solve the instance t2-ps08. Moreover, the instance t2-ps09 is not solved to optimality since the calculation is terminated after 2 days. On the other hand, our BDP algorithm can solve this instance in 1 h. Lower bounds are improved for all larger instances. Moreover, the CPU times are significantly less than those reported by Artigues and Feillet (2008). The UB-values for large instances are better on average compared to those of AF08. However, the overall performance for obtaining the upper bounds is poor compared to the best-known solutions.

5 Conclusions

In this paper, the bounded dynamic programming (BDP) algorithm that solves the job shop scheduling problem (JSP) is investigated. We developed the BDP algorithm for solving the JSP to optimality. The proposed algorithm is able to prove the optimality for moderate benchmark instances (e.g. 15 jobs and 15 machines, or 20 jobs and 10 machines) thus significantly increasing the applicability of the BDP approach.

The main contribution of the present work is to adapt the BDP algorithm for the JSP with sequence dependent setup times (SDST-JSP). As far as we know this has not been done before. The quality of the BDP algorithm is strongly dependent on the quality of lower bounds (LB). We present the algorithm that calculates an LB. The performance of the BDP is evaluated against 15 benchmark instances proposed by Brucker and Thiele (1996). The comparison among the best-known algorithms shows that the BDP algorithm developed in the current work outperforms all the current state-of-the-art exact methods for the SDST-JSP case. Moreover, we have improved the best-known lower bounds for all 5 unsolved instances given by Brucker and Thiele (1996).

Appendix

See Tables 11 and 12.

Table 11 Calculation of upper bounds: case SDST-JSP

Inst.	$n \times m$	BKS	H = 100		H = 1000		H = 10,000		H = 100,000		H = 1,000,000	
			UB	CPU	UB	CPU	UB	CPU	UB	CPU	UB	CPU
t2-ps01	10 × 5	**798***	804	1	798	2						
t2-ps02	10 × 5	**784***	784	0								
t2-ps03	10 × 5	**749***	759	1	759	2	749	4				
t2-ps04	10 × 5	**730***	730	0								
t2-ps05	10 × 5	**691***	693	0	691	0						
t2-ps06	15 × 5	**1009***	1077	17	1034	198	1026	1315	1009	1625		
t2-ps07	15 × 5	**970***	1028	34	1022	266	997	2838	993	9413	970	18,615
t2-ps08	15 × 5	**963***	1036	7	996	49	975	467	965	3072	965	7598
t2-ps09	15 × 5	**1060***	1062	2	1061	34	1061	132	1060	1707		
t2-ps10	15 × 5	**1018***	1075	2	1064	8	1047	108	1018	338		
t2-ps11	20 × 5	1438	1612	562	1522	18,177						
t2-ps12	20 × 5	1269	1370	6908	1358	66,390						
t2-ps13	20 × 5	1406	1442	3263	1439	16,852						
t2-ps14	20 × 5	1452	10,000	1	1540	1213						
t2-ps15	20 × 5	1485	1680	18	1551	6197						

Values in bold: best known result is reached

Asterisk: bound is equal to optimal solution

Table 12 Calculation of lower bounds: case SDST-JSP

Inst.	$n \times m$	Best lit.	CPU					
		LB	$LB_{fin} - 4$	$LB_{fin} - 3$	$LB_{fin} - 2$	$LB_{fin} - 1$	LB_{fin}	$LB_{fin} + 1$
t2-ps01	10×5	**798***	0	0	0	1	1	1
t2-ps02	10×5	**784***	0	0	0	0	0	0
t2-ps03	10×5	**749***	1	1	1	2	2	2
t2-ps04	10×5	**730***	0	0	0	0	0	0
t2-ps05	10×5	**691***	0	0	0	0	0	0
t2-ps06	15×5	**1009***	0	0	384	405	439	493
t2-ps07	15×5	**970***	14	15	15	18	18	179
t2-ps08	15×5	**963***	284	302	340	365	382	2861
t2-ps09	15×5	**1060***	247	322	340	402	2445	2607
t2-ps10	15×5	**1018***	0	0	0	0	0	1646

Inst.	$n \times m$	Best lit.	CPU				
		LB	$LB_{fin} - 20$	$LB_{fin} - 15$	$LB_{fin} - 10$	$LB_{fin} - 5$	LB_{fin}
t2-ps11	20×5	1395	1	2	2	2	2
t2-ps12	20×5	1242	2908	4579	10,047	18,579	34,967
t2-ps13	20×5	1342	13,976	19,673	30,767	44,780	61,548
t2-ps14	20×5	1432	1186	1653	2380	10,156	12,635
t2-ps15	20×5	1406	14,079	20,716	30,075	55,128	118,591

Value in bold: the best-known result is reached

Asterisk '*': a lower bound is equal to the optimal solution

References

Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job shop scheduling. Manag Sci 34(3):391–401

Artigues C, Feillet D (2008) A branch and bound method for the job-shop problem with sequence-dependent setup times. Ann Oper Res 159(1):135–159

Artigues C, Belmokhtar S, Feillet D (2004) A new exact solution algorithm for the job shop problem with sequence-dependent setup times. In: International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming. Springer, Berlin, pp 37–49

Balas E, Simonetti N, Vazacopoulos A (2008) Job shop scheduling with setup times, deadlines and precedence constraints. J Sched 11(4):253–262

Brucker P, Thiele O (1996) A branch and bound method for the general-shop problem with sequence dependent setup-times. Oper Res Spektrum 18(3):145–161

Brucker P, Jurisch B, Sievers B (1994) A branch and bound algorithm for the job-shop scheduling problem. Discrete Appl Math 49(1):107–127

Carlier J (1982) The one-machine sequencing problem. Eur J Oper Res 11(1):42–47
Carlier J, Pinson É (1989) An algorithm for solving the job-shop problem. Manag Sci 35(2):164–176
Cheung W, Zhou H (2001) Using genetic algorithms and heuristics for job shop scheduling with sequence-dependent setup times. Ann Oper Res 107(1–4):65–81
Da Silva RF, Urrutia S (2010) A general VNS heuristic for the traveling salesman problem with time windows. Discrete Optim 7(4):203–211
Della Croce F, Tadei R, Volta G (1995) A genetic algorithm for the job shop problem. Comput Oper Res 22(1):15–24
Feillet D, Dejax P, Gendreau M, Gueguen C (2004) An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. Networks 44(3):216–229
Fisher H, Thompson GL (1963) Probabilistic learning combinations of local job-shop scheduling rules. Ind Sched 3:225–251
Gonçalves JF, Resende MG (2014) An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. Int Trans Oper Res 21(2):215–246
González MA, Vela CR, Varela R (2008) A new hybrid genetic algorithm for the job shop scheduling problem with setup times. In: ICAPS, pp 116–123
González MA, Vela CR, Varela R (2009) Genetic algorithm combined with tabu search for the job shop scheduling problem with setup times. In: International work-conference on the interplay between natural and artificial computation. Springer, Berlin, pp 265–274
Graham RL, Lawler EL, Lenstra JK, Kan AR (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discrete Math 5:287–326
Grimes D, Hebrard E (2010) Job shop scheduling with setup times and maximal time-lags: a simple constraint programming approach. In: International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming. Springer, Berlin, pp 147–161
Grimes D, Hebrard E, Malapert A (2009) Closing the open shop: contradicting conventional wisdom. In: International conference on principles and practice of constraint programming. Springer, Berlin, pp 400–408
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
Kurdi M (2015) A new hybrid island model genetic algorithm for job shop scheduling problem. Comput Indus Eng 88:273–283
Martin P, Shmoys DB (1996) A new approach to computing optimal schedules for the job-shop scheduling problem. In: H.W. Cunningham, T.S. McCormick, M. Queyranne (eds.) International Conference on Integer Programming and Combinatorial Optimization. Springer, Berlin, pp 389–403
Nowicki E, Smutnicki C (1996) A fast taboo search algorithm for the job shop problem. Manag Sci 42(6):797–813
Nowicki E, Smutnicki C (2005) An advanced tabu search algorithm for the job shop problem. J Sched 8(2):145–159
Ozolins A (2017) Improved bounded dynamic programming algorithm for solving the blocking flow shop problem. Cent Eur J Oper Res. https://doi.org/10.1007/s10100-017-0488-5
Papalitsas C, Giannakis K, Andronikos T, Theotokis D, Sifaleras A (2015) Initialization methods for the TSP with time windows using variable neighborhood search. In: 2015 6th international conference on information, intelligence, systems and applications (IISA). IEEE, Washington, pp 1–6
Peng B, Lü Z, Cheng T (2015) A tabu search/path relinking algorithm to solve the job shop scheduling problem. Comput Oper Res 53:154–164
van Hoorn JJ (2016) Dynamic programming for routing and scheduling: optimizing sequences of decisions [Ph.D. thesis]. VU University Amsterdam
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 78:381
Zhang CY, Li P, Rao Y, Guan Z (2008) A very fast TS/SA algorithm for the job shop scheduling problem. Comput Oper Res 35(1):282–294