Computers & Operations Research 39 (2012) 2968–2977

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Solving the job-shop scheduling problem optimally by dynamic programming

Joaquim A.S. Gromicho [a,b], Jelke J. van Hoorn [a,b,*], Francisco Saldanha-da-Gama [c], Gerrit T. Timmer [a,b]

[a] Department of Econometrics and OR, VU University, Amsterdam, The Netherlands
[b] ORTEC, Gouda, The Netherlands
[c] Faculdade de Ciências & Centro de Investigação Operacional, Universidade de Lisboa, Portugal

ARTICLE INFO

Available online 8 March 2012

Keywords:
Job-shop scheduling
Dynamic programming
Complexity analysis

ABSTRACT

Scheduling problems received substantial attention during the last decennia. The job-shop problem is a very important scheduling problem, which is NP-hard in the strong sense and with well-known benchmark instances of relatively small size which attest the practical difficulty in solving it. The literature on the job-shop scheduling problem includes several approximation and exact algorithms. So far, no algorithm is known which solves the job-shop scheduling problem optimally with a lower complexity than the exhaustive enumeration of all feasible solutions. We propose such an algorithm, based on the well-known Bellman equation designed by Held and Karp to find optimal sequences and which offers the best complexity to solve the Traveling Salesman Problem known to this date. For the TSP this means $\mathcal{O}(n^2 2^n)$ which is exponentially better than $\mathcal{O}(n!)$ required to evaluate all feasible solutions. We reach similar results by first recovering the principle of optimality, which is not obtained for free in the case of the job-shop scheduling problem, and by performing a complexity analysis of the resulting algorithm. Our analysis is conservative but nevertheless exponentially better than brute force. We also show very promising results obtained from our implementation of this algorithm, which seem to indicate two things: firstly that there is room for improvement in the complexity analysis (we observe the generation of a number of solutions per state for the benchmark instances considered which is orders of magnitude lower than the bound we could devise) and secondly that the potential practical implications of this approach are at least as exciting as the theoretical ones, since we manage to solve some celebrated benchmark instances in processing times ranging from seconds to minutes.

1. Introduction

In a job-shop scheduling problem n jobs have to be processed by m dedicated machines. Each job has to visit all the machines following a specific order. The time each job requires in each machine depends on the job and on the machine and it is assumed to be known in advance. The jobs cannot overlap in the machines and no job can be processed simultaneously by two or more machines. Preemption is not allowed. The goal is to schedule the jobs so as to minimize the makespan, which is the maximum of their completion time.

The job-shop scheduling problem is one of the most studied combinatorial optimization problems. Nevertheless, it still remains a very challenging problem to solve optimally. From a complexity point of view, the problem is NP-hard [24]. Even some 'simplified versions' of the problem are NP-Hard such as three machines and three jobs with an arbitrary number of operations per job (a job may have to visit a machine more than once) [6]; three machines and unitary processing times [6]; three machines and no more than two operations per job [25]; two machines and no more than three operations per job [25]. Some particular cases of the job-shop scheduling problem can be solved in polynomial time, such as the problem with two machines and no more than two operations per job [16] and the problem with two machines and unitary processing times [14].

Among the methodologies that have been considered for tackling the job-shop scheduling problem we can distinguish exact procedures (working for instances with a rather limited size), heuristic procedures and polynomial time approximation algorithms.

To the best knowledge of the authors, all the exact algorithms that have been proposed for the job-shop scheduling problem are branch-and-bound procedures. Without being exhaustive we can cite the works of Applegate and Cook [3], Brucker et al. [5], Carlier and Pinson [7], Lageweg et al. [19] and Martin and Shmoys [27].

* Corresponding author at: Department of Econometrics and OR, VU University, Amsterdam, The Netherlands.
E-mail addresses: j.a.dossantos.gromicho@vu.nl (J.A.S. Gromicho), j.j.van.hoorn@vu.nl (J.J. van Hoorn), fsgama@fc.ul.pt (F. Saldanha-da-Gama), Gerrit.Timmer@ortec.com (G.T. Timmer).

0305-0548/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2012.02.024

Many heuristic approaches have been proposed in the literature for obtaining good quality solutions to the job-shop scheduling problem. A well-known procedure is the so-called shifting bottleneck procedure by Adams et al. [1]. Genetic algorithms were proposed by Croce et al. [8] and Dorndorf and Pesch [9]. Van Laarhoven et al. [33] and Steinhöfel et al. [31] proposed simulated annealing procedures for obtaining feasible solutions to the problem. A tabu search procedure was proposed in the works by Taillard [32] and Nowicki and Smutnicki [28]. Recently, Zhang et al. [36] proposed a new hybrid approach combining tabu search and simulated annealing, which proved to be very efficient for finding optimal or near-optimal solutions for many benchmark instances of the problem. Rego and Duarte [29] proposed a procedure that combines the basic shifting bottleneck procedure by Adams et al. [1] with a dynamic and adaptive neighborhood search procedure based on the so-called filter-and-fan method.

As far as polynomial time approximation algorithms are concerned, Shmoys et al. [30] and Leighton et al. [23] propose a procedure with a performance guarantee $O(\log(m\mu)\log(\min(m\mu,p_{\max}))/\log\log(m\mu))$ for a job-shop scheduling problem in which a job may have to return more than once to each machine; $p_{\max}=\max_{ij} p_{ij}$ with p_{ij} the processing time of job j on machine i, m is the number of machines and μ denotes the maximum number of operations over all jobs. When each job has to visit each machine exactly once, the factor μ can be ignored in the performance guarantee. Goldberg et al. [12] improve the above performance by presenting a polynomial time approximation algorithm with a performance guarantee $O(\log(m\mu)\log(\min(m\mu,p_{\max}))/\log\log(m\mu)^2)$. For the case in which each job visits each machine at most once, the above performance is still improved by Feige and Scheideler [10], who provide an approximation guarantee $O(m\mu\log(m\mu)\log\log(m\mu))$. For a fixed number of machines and also for a fixed maximum number of operations per job, Shmoys et al. [30] present a $2+\epsilon$ approximation algorithm. Jansen et al. [18] improve the previous performance guarantee. Leighton et al. [21,22] propose polynomial time approximation algorithms with constant performance guarantee for the job-shop scheduling problem with unitary processing times assuming that each job has to be processed exactly once on each machine.

Unlike the above-mentioned complexity results about approximations, little seems to be known about the complexity of exact algorithms for many NP-hard problems. Woeginger [34] stresses the importance of such algorithms and points out that for many such problems it is possible to do better than 'brute force'. In particular, for a single machine scheduling problem, an algorithm based on dynamic programming is mentioned. This algorithm follows along the lines of the celebrated work by Held and Karp [15] and Bellman [4] for the Traveling Salesman Problem, which offers still to this date the best complexity of an exact algorithm for the TSP (see Woeginger [35]).

In the case of the job-shop scheduling problem, the decision to be made regards the order by which the n jobs will be processed on each of the m machines. Accordingly, a brute-force enumerative algorithm for the problem has worst-case complexity $\mathcal{O}^*((n!)^m)$, which is lower than the worst-case complexity for branch-and-bound algorithms, enumerating over all variables.

In this paper we present an exact algorithm for the job-shop scheduling problem with a complexity that we can prove to be $\mathcal{O}((p_{\max})^{2n}(m+1)^n)$. This leads to a complexity of 2^{2n} in the special case of three machines and unitary processing times. The new algorithm is based on dynamic programming. We show that for the job-shop scheduling problem a straightforward application of the Held and Karp equation is not possible because the optimality principle does not work immediately. Nevertheless, we show that it is possible to redefine the state space and adjust the Bellman equation accordingly, in order to recover the optimality principle.

The remainder of the paper is organized as follows. In the next section, we present some background which includes notation, definitions and basic properties of the problem. In Section 3, we present the new algorithm. Section 4 is devoted to a complexity analysis. In Section 5, some computational tests performed with the new procedure are presented and discussed. The paper ends with a conclusion about the research done.

2. Notation, definitions and basic properties

Consider the job-shop scheduling problem as defined in the previous section. Let $\mathcal{J}=\{j_1,j_2,\ldots j_n\}$ denote the set of jobs and $\mathcal{M}=\{m_1,m_2,\ldots,m_m\}$ the set of machines. p_{ij} stands for the processing time of job $j\in\mathcal{J}$ on machine $i\in\mathcal{M}$. A job consists of m task operations each of which is associated with a specific machine. The sequence of operations for each job $j\in\mathcal{J}$ is denoted by $\pi_j(1),\ldots,\pi_j(m)$, that is, for job $j\in\mathcal{J}$, $\pi_j(i)$ is the ith machine that job j has to visit. $\mathcal{O}=\{o_1,o_2,\ldots,o_{n\times m}\}$ is the set of operations. The first n operations refer to the first operation of each job (in the order of the jobs), operations $o_{n+1},\ldots,o_{2n}$ concern the second operation of the n jobs, and so on. For an operation $o\in\mathcal{O}$, we denote by $j(o)$ and $m(o)$ the corresponding job and machine, respectively. Note that $j(o_i)=i \bmod n$. We denote by $p(o)$ the processing time of operation $o\in\mathcal{O}$. Note that $p(o)=p_{m(o)j(o)}$.

Definition 1. A schedule is a function $\psi:\mathcal{O}\longrightarrow\mathbb{N}\cup\{0\}$ such that for each operation $o\in\mathcal{O}$, $\psi(o)$ gives the starting time of operation o. A schedule ψ is said to be feasible if:

1. $\psi(o)\geq 0$, for each $o\in\mathcal{O}$;
2. For all $o_k,o_l\in\mathcal{O}$ such that $j(o_k)=j(o_l)$ and $k<l$ one has $\psi(o_k)+p(o_k)\leq\psi(o_l)$;
3. For all $o_k,o_l\in\mathcal{O}$ such that $o_k\neq o_l$ and $m(o_k)=m(o_l)$ we have that $\psi(o_l)+p(o_l)\leq\psi(o_k)$ or $\psi(o_k)+p(o_k)\leq\psi(o_l)$.

The goal in a job-shop scheduling problem is to find a feasible schedule with the minimum makespan, that is, a feasible schedule ψ which minimizes $\max_{o\in\mathcal{O}}\{\psi(o)+p(o)\}$. In this paper we consider the value 0 for the origin of time.

For every feasible schedule for the job-shop scheduling problem, it is possible to associate a sequence of operations where the order of the operations processed on each machine as well as the order defined for each job is preserved. One example of such type of sequence is obtained by sorting all operations by their starting time in the schedule considered. However, one single sequence of operations defines an infinite number of schedules (e.g. all those obtained by the addition of positive constants to the starting time of all operations). On the other hand, not all the sequences define a feasible solution.

Hereafter, we will be interested only in sequences that correspond to feasible solutions to the problem, that is, feasible sequences. For the sake of simplicity, in the remainder of the paper we will omit the word "feasible" every time we refer to a sequence.

Given a sequence of operations of a job-shop scheduling problem, we can obtain an unique schedule that results from starting all the operations as soon as possible in the order defined by the sequence and keeping feasibility. This unique schedule has the minimum makespan among all the schedules that can be associated to the sequence. Hereafter, the schedule we associate with a sequence will be this unique schedule.

Note that different sequences of operations can lead to the same schedule. Fig. 1 depicts a feasible solution for a small instance of the job-shop scheduling problem. The schedule associated with this solution can be obtained from the following

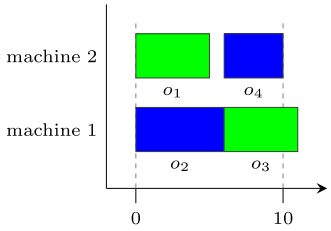

Fig. 1. An illustrative schedule.

feasible sequences: $o_1o_2o_3o_4$, $o_1o_2o_4o_3$, $o_2o_1o_3o_4$ and $o_2o_1o_4o_3$. This small example makes it clear that there is not a one-to-one correspondence between feasible schedules and sequences of operations. The following result defines a unique sequence for every schedule.

Proposition 1. *For every feasible solution for the job-shop scheduling problem there is one and only one sequence of operations defining the schedule such that the completion time of the operations along the sequence is non-decreasing and in which the order of the machines is increasing for two consecutive operations with equal completion time.*

Proof. Consider a feasible solution for the job-shop scheduling problem. A sequence of operations featuring the conditions stated is obtained by sorting the operations non-decreasingly in terms of their completion time and for those that have an equal completion time, by sorting them in increasing order of the machine associated with them. The total lexicographic order imposed on this sequence assures unicity. □

An immediate consequence of the previous proposition is stated in the following corollary.

Corollary 1. *For an optimal solution of the job-shop scheduling problem there exists one and only one sequence featuring the conditions stated in Proposition 1.*

With the purpose of building (feasible) sequences (and eventually the optimal sequence) to a job-shop scheduling problem, the operations can be iteratively added to the sequence. Before all the operations are in the sequence, we have an incomplete or partial sequence which we define formally as follows.

Definition 2. A partial sequence is a sequence of operations defined by the first k operations of a sequence for any $k=0,\ldots,|\mathcal{O}|$.

For a partial sequence T, we denote by $C_{\max}(T)$ the corresponding (partial) makespan, which is the maximum completion time for the operations scheduled in the order of the sequence.

Definition 3.

1. An ordered sequence is a sequence ordered as stated in Proposition 1.
2. An ordered partial sequence is a sequence defined by the first k operations of an ordered sequence for any $k=0,\ldots,|\mathcal{O}|$.
3. A sequence or a partial sequence not ordered as stated in Proposition 1 is called unordered.

It is straightforward to conclude that any unordered partial sequence can be converted into an ordered partial sequence with the same makespan. In fact, reordering the operations as stated in Proposition 1 does not increase the makespan.

Definition 4. Let $S\subseteq\mathcal{O}$ denote a subset of operations such that at least one ordered partial sequence can be associated to it. Let T denote an ordered partial sequence that can be associated to S.

1. Any set of operations obtained by adding to S one operation that is not in S but such that all the operations that precede it in the same job are already in S is called an expansion of S.
2. A partial sequence that is obtained by adding to the end of T one operation o that can be used to expand S is called an expansion of the ordered partial sequence T with o. Let $T+o$ denote this expansion.
3. Every partial sequence can be progressively expanded leading to a sequence of all operations thus defining a feasible solution. Such a final sequence is called a *completion* of the initial partial solution.

Remark 1. A subset of operations $S\subset\mathcal{O}$ has no (ordered) partial sequence associated with it when for some job an operation is in S while one of the preceding operations of the same job is not in S.

Hereafter, we consider only subsets of operations $S\subseteq\mathcal{O}$ such that at least one ordered partial sequence can be associated to them.

It should be noted that the expansion of an ordered partial sequence does not necessarily lead to a new ordered partial sequence. This is the case when the operation selected to join the sequence has an earliest completion time lower than the makespan of the partial sequence.

The job-shop scheduling problem can now be defined as the problem of finding the ordered sequence of operations which leads to the minimum makespan, that is, the job-shop shop scheduling problem can be seen simply as a sequencing problem involving all the operations that have to be performed. This motivates the approach that is presented and discussed in the following sections.

3. A dynamic programming approach for the job-shop scheduling problem

Held and Karp [15] presented a Dynamic Programming (DP) formulation for sequencing problems, which is famous for its application to the Traveling Salesman Problem. For the TSP, the resulting Bellman equation is

$$\begin{cases} C(\{u\},u)=c_{0u}, \\ C(S,u)=\min\limits_{v\in S\setminus\{u\}} C(S\setminus\{u\},v)+c_{vu}, \end{cases}$$

where $C(S,u)$ is the cost of the optimal path starting at 0 visiting all nodes in S and ending in u, and c_{vu} is the cost of going from v to u.

As mentioned above, the job-shop scheduling problem can be seen as a sequencing problem. Therefore, one could be tempted to use the Bellman equation above using the $C_{\max}$ of partial sequences to calculate the minimum in the Bellman equation and considering only ordered expansions.

However, a small example is enough to show that the optimality of the minimum in the Bellman equation principle does not hold in the case of the job-shop scheduling problem. Specifically, it is not true that partial sequences of the optimal sequence are optimal partial sequences. Fig. 2a depicts the optimal solution for an instance of the job-shop scheduling problem with three jobs and three machines (the data can be easily retrieved from the figure). The ordered sequence defining the solution depicted is $o_1o_3o_6o_2o_5o_4o_9o_7o_8$. However, the ordered partial sequence $o_2o_3o_6o_1o_5$ which leads to the schedule depicted in Fig. 2b dominates the partial sequence $o_1o_3o_6o_2o_5$ from the optimal solution. Also any completion of the schedule in Fig. 2b will not be completed before time 9 because operations o_4 and o_8 may not start before 4 and together they take 5 to complete on the same machine.

In order to use the recursive equation above, we propose several adjustments, which allow us to 'restore' the optimality principle. For the example presented both the partial solution that can be

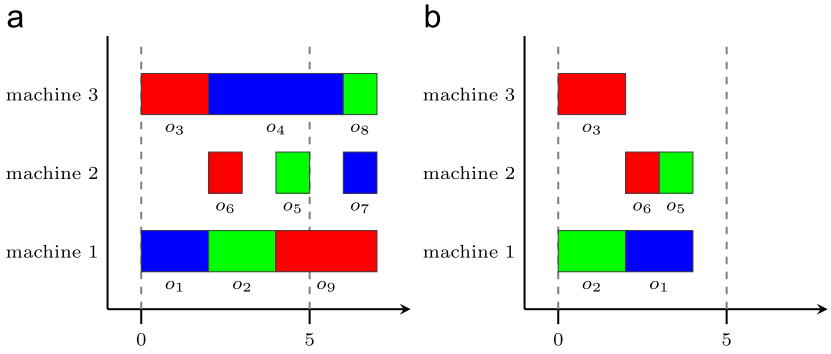

Fig. 2. Losing optimality. (a) Optimal solution and (b) dominating partial solution.

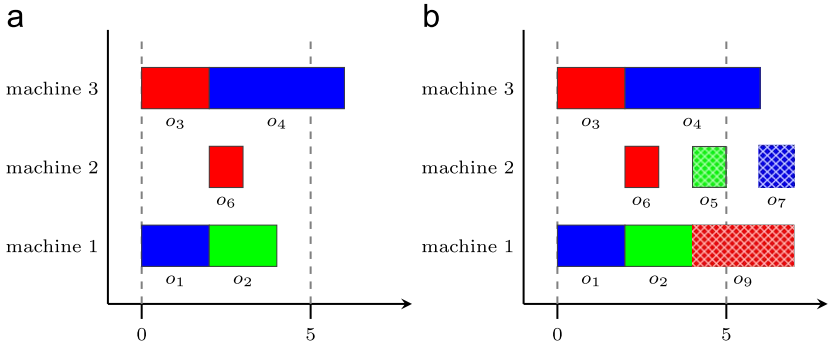

Fig. 3. Illustration of the values $\psi(T,o)$ and $\xi(T,o)$. (a) Scheduled defined by T and (b) possible schedules defined by $T+o$.

completed to the optimal solution (Fig. 2a) as well as the partial solution dominating it (Fig. 2b) should be considered for expansion.

3.1. Restoring the optimality guarantee

Let $S \subseteq \mathcal{O}$. Denote by $\Xi(S)$ the set of all the ordered (partial) sequences that can be associated with S (assuming that there is at least one such sequence, that is, $\Xi(S) \neq \emptyset$).

Denote by $\varepsilon(S)$ the set of all operations that can be used to obtain an expansion of S. Note that $|\varepsilon(S)| \leq n$ because S can only be expanded with the first unsequenced operation in each job. Note also that if $|\varepsilon(S)| < n$, there are $n-|\varepsilon(S)|$ jobs that have all their operations already sequenced.

Denote by $\eta(T)$ the set of all operations that can be used to obtain an ordered expansion of an ordered partial sequence T. For $T \in \Xi(S)$, we have that $\eta(T) \subseteq \varepsilon(S)$. For an ordered partial sequence $T \in \Xi(S)$, let $i^*(T)$ denote the machine with the largest completion time (i.e. the machine determining the value $C_{\max}(T)$). In case there are multiple such machines, let $i^*(T)$ denote the highest-numbered machine. As T is ordered $i^*(T) = m(o^*(T))$, where $o^*(T)$ is the last operation of sequence T.

For each $o \in \varepsilon(S)$ and for each $T \in \Xi(S)$ define $\psi(T,o)$ as the earliest starting time for operation o if this operation is added to the end of the ordered partial sequence T (denoted by $T+o$). We have $o \in \eta(T)$ if and only if $T+o$ is an ordered expansion of T, that is

$$\psi(T,o)+p(o) > C_{\max}(T) \quad \text{or} \quad \psi(T,o)+p(o) = C_{\max}(T) \wedge m(o) > i^*(T).$$

So $\psi(T,o)+p(o) \geq C_{\max}(T)$ where equality is only allowed when o has to be processed on a higher machine number compared to the last operation $o^*(T)$ of T.

Define also

$$\xi(T,o) = \begin{cases} \psi(T,o)+p(o) & \text{if } o \in \eta(T), \\ C_{\max}(T)+p(o) & \text{otherwise.} \end{cases}$$

Note that $C_{\max}(T) \leq \xi(T,o) \leq C_{\max}(T)+p(o)$, $\forall o \in \varepsilon(S)$.

$\xi(T,o)$ represents an 'aptitude' value for operation o if this operation is used to expand T. In case $T+o$ is an ordered partial sequence ($o \in \eta(T)$), $\xi(T,o)$ gives the (minimum) makespan for the operations in $T+o$. Otherwise, $\xi(T,o)$ gives a loose upper bound on this makespan.

Actually $\xi(T,o)$ gives a lower bound of the completion of o in any ordered sequence that starts with T as a subsequence. For $o \in \eta(T)$ it follows directly from the definition. Otherwise, if $o \notin \eta(T)$ and o is added directly to T, then $T+o$ is unordered. To add o to any (ordered) expansion T' of T such that $T'+o$ is ordered, an operation o' with $m(o') = m(o)$ should be part of this expansion ($T' = T+\cdots+o'+\cdots$). Since $T+\cdots+o'+\cdots$ is ordered we have $\psi(T+\cdots,o')+p(o') \geq C_{\max}(T+\cdots) \geq C_{\max}(T)$. Since o' and o are on the same machine we have that $\psi(T',o) \geq \psi(T+\cdots,o')+p(o') \geq C_{\max}(T)$, thus $\psi(T',o)+p(o) \geq C_{\max}(T)+p(o) = \xi(T,o)$.

Denote by $\lambda(S)$ the set of operations that are the last operation of some job represented in S. Note that this set only depends on the set S and not on any sequence $T \in \Xi(S)$. Taking into account this definition we always have $|\lambda(S)| \leq n$, $\forall S \subseteq \mathcal{O}$.

In order to clarify the previous concepts, we consider the instance of the job-shop scheduling problem that was introduced in Fig. 2. In particular, for this instance, consider the ordered partial sequence $T = o_1o_3o_6o_2o_4$ (which leads to the schedule depicted in Fig. 3a). In this case we have $C_{\max}(T) = 6$, $S = \{o_1,o_2,o_3,o_4,o_6\}$, $\varepsilon(S) = \{o_5,o_7,o_9\}$, $\lambda(S) = \{o_2,o_4,o_6\}$ and $\eta(T) = \{o_7,o_9\}$. Taking into account that $p(o_5) = 1$, $p(o_7) = 1$ and $p(o_9) = 3$ we obtain (see Fig. 3b):

$$\psi(T,o_5) = 4, \quad \psi(T,o_7) = 6, \quad \psi(T,o_9) = 4,$$

$$\xi(T,o_5) = 7, \quad \xi(T,o_7) = 7, \quad \xi(T,o_9) = 7.$$

Note that if we have $T' = o_1o_3o_6o_2o_5o_4$ with $T' \in \Xi(S')$ and $S' = \{o_1,o_2,o_3,o_4,o_5,o_6\}$, we get $\lambda(S') = \{o_4,o_5,o_6\}$.

Proposition 2. *If T_1 and T_2 are two ordered partial sequences associated with $S \subseteq \mathcal{O}$ (that is $T_1,T_2 \in \Xi(S)$) such that $\xi(T_2,o) \leq \xi(T_1,o)$, $\forall o \in \varepsilon(S)$, then any ordered expansion of T_1 with*

an operation $o \in \eta(T_1)$ is dominated by the (possibly unordered) expansion of T_2 with o.

Proof. Let $o \in \eta(T_1)$ (so that $T_1 + o$ is an ordered partial sequence). We have

$$\xi(T_1,o) = \psi(T_1,o) + p(o) = C_{\max}(T_1 + o) \geq C_{\max}(T_1).$$

The inequality holds because otherwise adding o to T_1 would not lead to an ordered partial solution.

Accordingly, taking into account that $\xi(T_2,o) \leq \xi(T_1,o)$ we can write $\psi(T_1,o) + p(o) \geq \xi(T_2,o)$. Therefore, either we have

$$\psi(T_1,o) + p(o) \geq \psi(T_2,o) + p(o)$$

directly, or we have

$$\psi(T_1,o) + p(o) \geq C_{\max}(T_2) + p(o) \geq \psi(T_2,o) + p(o).$$

In any case, $\psi(T_1,o) + p(o) \geq \psi(T_2,o) + p(o)$ and thereby $\psi(T_1,o) \geq \psi(T_2,o)$.

This shows that it is possible to add o to T_2 starting at time $\psi(T_1,o)$. In this case, we would obtain $C_{\max}(T_2 + o) = \psi(T_1,o) + p(o) = C_{\max}(T_1 + o)$. However, the best starting time for o when the operation is added to T_2 is $\psi(T_2,o)$ which can be lower than $\psi(T_1,o)$. In such case we could get $C_{\max}(T_2 + o) < C_{\max}(T_1 + o)$. In any case we get $C_{\max}(T_2 + o) \leq C_{\max}(T_1 + o)$, which proves the result. □

Despite its simplicity and almost trivial nature, Proposition 2 plays a crucial leading to our algorithm. The following corollary helps understand its importance.

Corollary 2. *Given the conditions stated in Proposition2, every ordered completion of T_1 is dominated by the same (possibly unordered) completion of T_2.*

Proof. Let ψ_{T_1} be an ordered completion of T_1. For any operation $o \in \mathcal{O} \backslash S$ we have two cases: $o \in \varepsilon(S)$, that is, the next operation for each job, or $o \notin \varepsilon(S)$, for the following operations. All $o \in \varepsilon(S)$ can be scheduled in T_2 starting at time $\psi_{T_1}(o)$, since $\psi_{T_1}(o) \geq \xi(T_1,o) - p(o)$ and $\xi(T_2,o) \leq \xi(T_1,o)$, $\forall o \in \varepsilon(S)$. All other operations ($o \in (\mathcal{O} \backslash S) \backslash \varepsilon(S)$) are dependent of the operations in $\varepsilon(S)$ and can thereby also be scheduled starting at time $\psi_{T_1}(o)$. This creates a possible idle schedule ψ'_{T_2} with $C_{\max}(\psi'_{T_2}) = C_{\max}(\psi_{T_1})$. So the completion of T_1 is dominated by the non-idle version of schedule ψ'_{T_2} which is a (possibly unordered) completion of T_2. □

To clarify this we show the partial solution of Fig. 2b (T_a) and another partial solution which actually dominates this solution (see Fig. 4). We have that $\xi(T_a,o_4) = 8$, $\xi(T_a,o_8) = 5$, $\xi(T_a,o_9) = 7$, $\xi(T_b,o_4) = 8$, $\xi(T_b,o_8) = 4$ and $\xi(T_b,o_9) = 7$, so clearly we have that $\xi(T_b,o) \leq \xi(T_a,o)$, $\forall o \in \varepsilon(S)$. As we can see all operations of a completion of T_a in Fig. 4a can be added at the same starting times after the partial solution T_b of Fig. 4b, actually the operations o_8, o_4 and o_7 can be advanced by a single time unit. While Fig. 4 shows just a single possible completion of T_a this principle holds for all possible completions of T_a.

The previous proposition and its corollary establish a means for comparing two ordered partial sequences, namely, in terms of their possible expansions if done using the same operation. We introduce some additional notation.

For $S \subseteq \mathcal{O}$ and $T_1, T_2 \in \Xi(S)$ if $\xi(T_2,o) \leq \xi(T_1,o)$, $\forall o \in \varepsilon(S)$, with at least one strict inequality, we write $T_2 \lessdot T_1$. The reverse relation will be denoted by $\gtrdot$ and equality by $\doteq$. Note that if $\varepsilon(S) = \emptyset$, then $T_1, T_2 \in \Xi(\mathcal{O})$ and we will use $T_2 \lessdot T_1$ as another way to write $C_{\max}(T_2) < C_{\max}(T_1)$.

The relation just established between ordered partial sequences allows us to partition the set $\Xi(S)$ into two sets, say $\widehat{\Xi}(S)$ and $\Xi(S) \backslash \widehat{\Xi}(S)$, such that for every $T_1 \in \widehat{\Xi}(S)$, $\nexists T_2 \in \Xi(S)$ such that $T_2 \lessdot T_1$. We define the set $\widehat{\Xi}(S)$ to be the minimal set among all the possibilities, that is, if two or more potential elements for being included in $\widehat{\Xi}(S)$, say T_1, T_2, …, are such that $T_1 \doteq T_2 \doteq \ldots$, then only one is chosen to be included in the set. Any rule can be defined to determine this choice, as long as the same rule is used everywhere. By construction, $|\widehat{\Xi}(\mathcal{O})| = 1$ and it becomes clear that for every possible choice of a sequence $T \in \widehat{\Xi}(\mathcal{O})$, T is optimal for the job-shop scheduling problem.

Denote by $\overset{\triangle}{\Xi}(S)$ the set of ordered partial sequences $T \in \widehat{\Xi}(S)$, such that all subsequences of T (naturally all ordered), are also elements of the set $\widehat{\Xi}(\cdot)$ associated with their correspondent sets. This means all subsequences of $T \in \overset{\triangle}{\Xi}(S)$ are non dominated in their corresponding sets.

Now we have to find a way to construct these sets and assure that $\overset{\triangle}{\Xi}(\mathcal{O}) \neq \emptyset$.

Proposition 3. *The set $\overset{\triangle}{\Xi}(\mathcal{O})$ it is non-empty.*

Proof. Let $T_1 \in \Xi(\mathcal{O})$ be an optimal sequence and suppose that $T_1 \notin \overset{\triangle}{\Xi}(\mathcal{O})$. In this case, there is at least one partial subsequence $T'_1 \in \Xi(S')$ of T_1 such that $T'_1 \notin \widehat{\Xi}(S')$, where $S' \subset \mathcal{O}$ is the set of operations involved in T'_1. Among such sequences, let $\overline{T}'_1 \in \Xi(\overline{S}')$ be the one with the minimum number of operations. Then $\exists T'_2 \in \widehat{\Xi}(\overline{S}')$ such that $T'_2 \lessdot \overline{T}'_1$ or $T'_2 \doteq \overline{T}'_1$. Accordingly, we can consider the completion T_2 of T'_2 with the same operations and in the same order that is necessary to obtain T_1 from $\overline{T}'_1$. We can distinguish the following cases:

I. T_2 is ordered. This gives us two cases.
   (a) $T_2 \in \overset{\triangle}{\Xi}(\mathcal{O})$.
   (b) $T_2 \notin \overset{\triangle}{\Xi}(\mathcal{O})$. Now we can use the same procedure to find a new optimal solution T_3. However, since $T'_2 \in \widehat{\Xi}(\overline{S}')$, the subsequence T''_2 of T_2 with the minimum number of

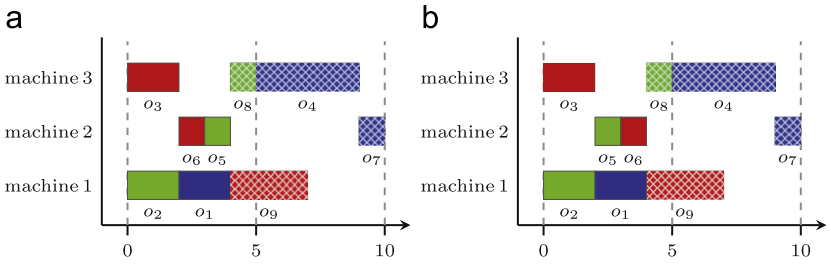

Fig. 4. Example of domination. (a) Dominated partial solution T_a and (b) dominating partial solution T_b.

operations among those such that $\overline{T_2''} \notin \widehat{\Xi}(\overline{S''})$ has more operations than $\overline{T}_1'$.

II. T_2 is not ordered. We have that the ordered sequence T_2^* corresponding to the schedule defined by this completion satisfies, for all operations $o \in \mathcal{O}\backslash S'$, $\psi_{T_2^*}(o) \le \psi_{T_1}(o)$ with strict inequality for at least one of them.

In Case Ia we are finished. Case Ib increases the number of operations of the partial subsequence eventually expanded to the optimal sequence found in Case Ia. Case II does not necessarily increase the number of operations of the partial subsequence nor decreases it. As the property stated in II, is transitive it cannot include any circular relations. As Case Ib increases the number of operations known of the optimal sequence $T \in \overset{\triangle}{\Xi}(\mathcal{O})$ to be found, the procedure of finding T may alternate between cases Ib and II. but is finite and will end in Case Ia. □

Corollary 3. *The set $\overset{\triangle}{\Xi}(\mathcal{O})$ contains the optimal sequence to the job-shop scheduling problem.*

Proof. The proposition assures that $\overset{\triangle}{\Xi}(\mathcal{O}) \neq \emptyset$. Being non-empty, taking into account the way the set is built, we conclude that it contains a single element, which is optimal. □

3.2. Dynamic Programming formulation for the job-shop scheduling problem

The question arising now is how can $\overset{\triangle}{\Xi}(\mathcal{O})$ be progressively obtained. In order to give an appropriate answer, we define $X(S)$ as the set of all ordered partial sequences $T \in \Xi(S)$ such that T is an expansion of an ordered partial sequence T', where $T = T' + o$ and $T' \in \overset{\triangle}{\Xi}(S\backslash\{o\})$. More formally, we define

$$X(S) = \bigcup_{o \in \lambda(S)} \bigcup_{T' \in \overset{\triangle}{\Xi}(S\backslash\{o\}) \text{ with } o \in \eta(T')} \{T' + o\}.$$

Define $\widehat{X}(S)$ in a similar way as done for defining $\widehat{\Xi}(S)$. Note that the same rule need to be used to determine which sequence belongs to the set if $T_1 \doteq T_2$. The following result establishes a relation between $\overset{\triangle}{\Xi}(S)$ and $\widehat{X}(S)$.

Proposition 4. $\overset{\triangle}{\Xi}(S) = \widehat{X}(S)$.

Proof. By definition $\overset{\triangle}{\Xi}(S) \supseteq \widehat{X}(S)$. Let $T \in \overset{\triangle}{\Xi}(S)$ and suppose $T \notin \widehat{X}(S)$ and define T' such that $T = T' + o$. Accordingly, $T' \in \widehat{\Xi}(S')$, where $S' = S\backslash\{o\}$ and $T' \notin \overset{\triangle}{\Xi}(S')$. Since $T' \notin \overset{\triangle}{\Xi}(S')$ but $T' \in \widehat{\Xi}(S')$ we conclude that there exists an ordered subsequence $T'' \in \Xi(S'')$ of T' for which $T'' \notin \widehat{\Xi}(S'')$, so we conclude that $T \notin \overset{\triangle}{\Xi}(S)$ and thereby $\overset{\triangle}{\Xi}(S) = \widehat{X}(S)$. □

We have finally gathered all the necessary elements to propose a Dynamic Programming approach for the job-shop scheduling problem.

Considering $\overset{\triangle}{\Xi}(S)$ and $\widehat{X}(S)$ ($S \subseteq \mathcal{O}$) as defined above, we can introduce the following Bellman equation for the job-shop scheduling problem:

$$\begin{cases} \overset{\triangle}{\Xi}(\{o\}) = \{T\} & \text{where } T = o, \\ \overset{\triangle}{\Xi}(S) = \widehat{X}(S), & S \subseteq \mathcal{O}. \end{cases}$$

Based on this equation we propose Algorithm 1 for building the set $\overset{\triangle}{\Xi}(\mathcal{O})$, that is, to find the optimal solution to the problem.

Algorithm 1. Dynamic Programming for the job-shop scheduling problem.

Input: An instance of the problem with n jobs and m machines
Assume: $\widehat{X}(S) = \emptyset$ for all S
Output: A sequence T associated with an optimal schedule to the problem and its makespan $C_{\max}(T)$
for all $o \in \varepsilon(\emptyset)$ **do**
  $\widehat{X}(\{o\}) = \{T\}$ with $T = o$
for $l = 1$ **to** $n * m$ **do** // for all sequence lengths
  for all $S \subseteq \mathcal{O} : |S| = l$ **do**
    for all $T_1 \in \widehat{X}(S)$ **do**
      for all $o \in \eta(T)$ **do**
        $T_1' = T_1 + o$
        if $T_1' \not\succcurlyeq T_2$ for all $T_2 \in \widehat{X}(S \cup \{o\})$ **then**
          for all $T_2 \in \widehat{X}(S \cup \{o\})$ **do**
            if $T_1' \prec T_2$ **then**
              $\widehat{X}(S \cup \{o\}) = \widehat{X}(S \cup \{o\})\backslash\{T_2\}$
          $\widehat{X}(S \cup \{o\}) = \widehat{X}(S \cup \{o\}) \cup \{T_1'\}$
Let T be such that $\{T\} = \widehat{X}(\mathcal{O})$ // $\widehat{X}(\mathcal{O}) = \overset{\triangle}{\Xi}(\mathcal{O})$ and $|\overset{\triangle}{\Xi}(\mathcal{O})| = 1$
return T and $C_{\max}(T)$

In short, Algorithm 1 can be described as follows: First n sequences of a single operation are constructed (the first operation for each job). Note that all these sequences have a different set of operation(s) S. Next for all these sets S with $|S| = 1$ the sequences are expanded with all possible operations in $o \in \epsilon(S)$, expansions leading to ordered partial sequences are kept for the following step grouped ($\widehat{X}(S')$) by their new set S' with $|S'| = 2$. When an ordered partial sequence is added to a non-empty group it is compared to all items already in that group and dominated sequences are removed. After all sequences of length l are expanded (and thus sequences of length $l+1$ are created) we proceed with expanding all sequences of length $l+1$, and so on.

3.3. State space reduction

In this section we propose a reduction in the state space of our Dynamic Programming approach for the job-shop scheduling problem.

Consider the additional notation $\varepsilon(S,i) = \{o \in \varepsilon(S) : m(o) = i\}$, $\varepsilon(S,i)$ contains all the possible expansions of S that are associated with machine i ($i = 1,\ldots,m$). Naturally, $\bigcup_{i=1}^{m} \varepsilon(S,i) = \varepsilon(S)$. The following result holds.

Proposition 5. *Let $T \in \Xi(S)$. If $\exists i \in \{1,\ldots,m\}$ such that*

(i) $\exists o^* \in \varepsilon(S,i) : o^* \notin \eta(T)$ *and*
(ii) $\forall o \in \varepsilon(S,i)$, $\xi(T,o) = C_{\max}(T) + p(o)$

then there exists an optimal solution to the problem such that the corresponding ordered sequence does not start with T.

Proof. Consider an optimal solution such that the corresponding ordered sequence results from a completion of T. One of the operations in the completion is o^* that appears after all operations in T. This means that in the ordered sequence associated with the optimal solution, $\psi(o^*) + p(o^*) \ge C_{\max}(T)$. However, in the conditions of the proposition it is possible to schedule o^* starting at time $\psi(T,o^*)$ ending in time $\psi(T,o^*) + p(o^*) \le C_{\max}(T)$. With this change, it might be possible to schedule earlier the remaining operations in the completion of T. By reordering the sequence thus obtained we obtain a new ordered sequence that does not increase the optimal makespan (so is optimal) and does not have T as an ordered partial sequence. □

Remark 2. For the operations o that satisfy the second condition of Proposition 5 we cannot conclude that $o \notin \eta(T)$. In fact it can be the case that $\psi(T,o) = C_{\max}(T)$.

Proposition 6. *If we change the definitions of $\overset{\triangle}{\Xi}(S)$ and $X(S)$ by removing from these sets every $T \in \Xi(S)$ satisfying Proposition 5, the Bellman equation still gives an optimal solution.*

Proof. According to the conditions stated in Proposition 5 only possibly optimal solutions are removed where it is possible to change the schedule by advancing at least a single operation (o^* as defined in Proposition 5). For the optimal solution found according to the proof of Proposition 3 no operation can be advanced without loosing the feasibility of the schedule, so this optimal solution is not removed. □

This state space reduction can be easily incorporated in Algorithm 1 by checking the conditions of Proposition 5 for each T_1. When these conditions are satisfied do not expand T_1 any further. Note that T_1 has to be added to $X(S)$ to enable it to discard other sequences (when $T_1 \lessdot T_2$).

4. Complexity analysis

In this section we study the complexity of Algorithm 1 and present an upper bound on this complexity. In this analysis, we are not taking into account the state space reduction presented in Section 3.3, thereby also giving an upper bound for the complexity of Algorithm 1 with the state space reduction.

4.1. The complexity of our algorithm

Considering the main body of Algorithm 1 we see that the first two `for` loops together loop over all sets $S \subseteq \mathcal{O}$. This is at most 2^{nm}. As we will see below, this value can be decreased. Next, there is a loop over all elements $T_1 \in \overset{\triangle}{\Xi}(S)$, after which all expansions $o \in \eta(T_1)$ are made. Inside these loops we have to determine $\xi(T_1{}',o)$ for $o \in \varepsilon(S \cup \{o\})$, and we need to compare $T_1{}'$ with all $T_2 \in \overset{\triangle}{\Xi}(S \cup \{o\})$ which is presented in Algorithm 1 as looping twice but can be implemented by looping once over all elements of $\overset{\triangle}{\Xi}(S \cup \{o\})$. This leads to a complexity

$$O(2^{nm}|\overset{\triangle}{\Xi}(S)|\,|\eta(T_1)|(|\varepsilon(S \cup \{o\})| + |\overset{\triangle}{\Xi}(S \cup \{o\})|)).$$

First we take a look into the number of subsets $S \subseteq \mathcal{O}$. Note that not all of these subsets have (feasible) ordered sequences associated with them, in which cases we have $\Xi(S) = \emptyset$. This is the case when an operation of some job is in the set but one of the preceding operations of the same job is not in the set. In fact, each job imposes a sequence of precedence relations which must be respected by an ordered sequence. This gives us n independent precedence sequences of length m. According to Gromicho et al. [13] this leads to a reduction of $((m+1)/2^m)^n$ in the possible subsets $S \subseteq \mathcal{O}$. So we have at most $((m+1)/2^m)^n 2^{nm} = (m+1)^n$ subsets $S \subseteq \mathcal{O}$ for which $\Xi(S) \neq \emptyset$.

Furthermore, observe that $|\eta(T_1)| \leq n$ and $|\varepsilon(S \cup \{o\})| \leq n$.

Finally, we need to determine $|\overset{\triangle}{\Xi}(S)|$ and $|\overset{\triangle}{\Xi}(S \cup \{o\})|$.

Let c be the minimal value of $C_{\max}(T)$ with $T \in \overset{\triangle}{\Xi}(S)$ and let T_L be the sequence such that $C_{\max}(T_L) = c$. Observe that $\forall o \in \varepsilon(S)$ we have $c \leq \xi(T_L,o) \leq c + p_{\max}$. Now we have $\forall T \in \overset{\triangle}{\Xi}(S)$ that $C_{\max}(T) < c + p_{\max}$, otherwise $T_L \lessdot T$. So we have at most $p_{\max}$ possible values for $C_{\max}(T)$ for $T \in \overset{\triangle}{\Xi}(S)$. For each of these values $c+i$ ($i < p_{\max}$) we have that $T \in \overset{\triangle}{\Xi}(S)$ and $C_{\max}(T) = c+i$ results in $c+i \leq \xi(T,o) \leq c + p_{\max} + i$, $\forall o \in \varepsilon(S)$. As we know that $T,T' \in \overset{\triangle}{\Xi}(S)$ forbids that $\forall o \in \varepsilon(S)$ $\xi(T,o) = \xi(T',o)$, as this would result in $T \doteq T'$ which is forbidden by the definition of $\overset{\triangle}{\Xi}$, we conclude that all $T \in \overset{\triangle}{\Xi}(S)$ should have least one value $\xi(T,o)$ different. This results in an upper bound for $|\overset{\triangle}{\Xi}(S)|$ of $p_{\max}(1+p_{\max})^n$.

We can improve this bound by looking at the maximum possible values of $\xi(T,o)$ which does not allow for any domination. Considering the values of $\xi(T,o)$, $\forall o \in \varepsilon(S)$ and $C_{\max}(T) = c$ relative to c, we have $0 \leq \xi(T,o) - c \leq p_{\max}$. These values of $\xi(T,o)$ can be represented with a subset of the multiset $\mathcal{S} = k_1,\ldots,k_n$, with $k_i = p_{\max}+1$ for $i = 1,\ldots,n$. $\mathcal{S}$ consists of $k_i = p_{\max}+1$ copies of n different elements x_i, $i = 1,\ldots,n$, where x_i is associated with job i. Denote by $\sigma(T)$ the subset associated with T. Thus, $\sigma(T) \subseteq \mathcal{S}$ is composed by taking $\forall o \in \varepsilon(S)$, $\xi(T,o) - c$ copies of x_i, where $i = j(o)$. Now observe that for $T_1,T_2 \in \Xi(S)$, $T_1 \lessdot T_2$ if and only if $\sigma(T_1) \subseteq \sigma(T_2)$ with $\sigma(T_1) = \sigma(T_2)$ when $T_1 \doteq T_2$. We conclude that $\forall T_1,T_2 \in \overset{\triangle}{\Xi}(S)$ we have $\sigma(T_1) \not\subseteq \sigma(T_2)$ and $\sigma(T_1) \not\supseteq \sigma(T_2)$.

Before proceeding with our analysis, we recall the concept of an Antichain (see [2] for further details).

Definition 5. An Antichain is a collection of subsets of a set where no two elements of the collection are subsets of each other.

According to this definition, the collection of $\sigma(T)$, with $T \in \overset{\triangle}{\Xi}(S)$, is an antichain. In particular, the collection of $\sigma(T)$, with $T \in \overset{\triangle}{\Xi}(S)$ with $C_{\max}(T) = c+i$ for fixed i, is an antichain. To make this text self-contained, we recall the following two results on antichains.

Proposition 7. *The largest antichain in the collection of all subsets of a multiset is smaller or equal to the largest rank number N_i (N_i the number of elements with rank i).*

Proof. See [2]. □

Proposition 8. *The size of the largest rank for a multiset is equal to size of the middle rank λ which is*

$$N_\lambda \approx \left(\frac{2}{\pi}\right)^{1/2} \frac{\prod_i (k_i+1)}{\sqrt{\frac{1}{3}\sum_i k_i(k_i+2)}}.$$

Proof. See [2]. □

In our case we have $k_i = p_{\max}+1$ for all i. Accordingly, we have

$$|\overset{\triangle}{\Xi}(S)| \approx p_{\max}\left(\frac{2}{\pi}\right)^{1/2} \frac{(p_{\max}+2)^n}{\sqrt{\frac{1}{3}n((p_{\max})^2 + 4p_{\max} + 3)}}.$$

Therefore

$$|\overset{\triangle}{\Xi}(S)| = \mathcal{O}\left(\frac{(p_{\max})^{n+1}}{\sqrt{n(p_{\max})^2}}\right) = \mathcal{O}\left(\frac{(p_{\max})^n}{\sqrt{n}}\right).$$

Note that we only looked at a dominance for $T \in \overset{\triangle}{\Xi}(S)$ with equal $C_{\max}(T)$ and ignored the dominance between sequences with different values of $C_{\max}$.

We can finally state and prove the following result.

Proposition 9. *Algorithm1 has complexity*

$\mathcal{O}((p_{\max})^{2n}(m+1)^n)$.

Proof. The value obtained for $|\overset{\triangle}{\Xi}(S)|$ leads to the following value for the complexity of Algorithm 1:

$$\mathcal{O}\left(\left(\frac{(p_{\max})^n}{\sqrt{n}}+n\right)\frac{n(p_{\max})^n}{\sqrt{n}}(m+1)^n\right).$$

From here we obtain successively

$$\mathcal{O}(((p_{\max})^{2n}+n\sqrt{n}(p_{\max})^n)(m+1)^n),$$

$$\mathcal{O}((p_{\max})^{2n}(m+1)^n). \quad \square$$

With the analysis above, we were able to give an upper bound on $|\overset{\triangle}{\Xi}(S)|$ with $S\subseteq\mathcal{O}$. This results in an upper bound on the complexity of Algorithm 1. Note, however, that the measured values of $|\overset{\triangle}{\Xi}(S)|$ are much lower and thus we strongly suspect that the actual complexity of our procedure is much lower. This is also evidenced by the computational results presented in Section 5. Nevertheless, this upper bound can already be useful as we show next.

4.2. Comparison with brute force

The following results holds.

Proposition 10. *For a fixed $p_{\max}$, the Dynamic Programming approach that we propose for the job-shop scheduling problem has a complexity that is exponentially smaller than brute-force in n and in m.*

Proof. Using Stirling's approximation ($n!\approx\sqrt{2\pi n}(n/e)^n$) we can evaluate the ratio between the complexity associated with brute-force and the complexity of our procedure. We have:

$$\frac{(n!)^m}{(p_{\max})^{2n}(m+1)^n}\approx\frac{\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)^m}{(p_{\max})^{2n}(m+1)^n}$$

$$\approx\sqrt{2\pi n}^m\left(\frac{\left(\frac{n}{e}\right)^m}{(p_{\max})^2(m+1)}\right)^n.$$

This is clearly exponential in m, but is only exponential in n if

$$\frac{\left(\frac{n}{e}\right)^m}{(p_{\max})^2(m+1)}>1$$

or equivalently, if $(n/e)^m>(p_{\max})^2(m+1)$. Accordingly, for a fixed $p_{\max}$ the result holds. $\square$

This result shows that the new algorithm solves the job-shop scheduling problem with a complexity exponentially lower than 'brute force'.

5. Empirical analysis

In order to evaluate our complexity analysis of the new algorithm, we ran some computational tests considering benchmark instances for the job-shop scheduling problem. In particular, we considered instances *ft06* and *la01-la05*. The first instance was first proposed by Fisher and Thompson [11]. The other five were proposed by Lawrence [20]. These benchmark instances (among many others) are available in the OR Library [26].

Table 1
Computational results.

Instance	Size			Resources	
	n	m	$p_{\max}$	CPU time (s)	Memory (MB)
ft06	6	6	10	<1	2
la01	10	5	98	1533	1922
la02	10	5	99	1961	2313
la03	10	5	91	1206	1400
la04	10	5	98	1629	2023
la05	10	5	97	965	1321

Table 2
Computational observations.

Instance	Observed		Theoretical upper bound	
	$\max_S\lvert\overset{\triangle}{\Xi}(S)\rvert$	$\lvert\bigcup_S\overset{\triangle}{\Xi}(S)\rvert$	$\max_S\lvert\overset{\triangle}{\Xi}(S)\rvert$	$\lvert\bigcup_S\overset{\triangle}{\Xi}(S)\rvert$
ft06	13	30 409	408×10^3	$48\,030\times10^6$
la01	142	63 170 946	$25\,838\times10^{15}$	$1\,562\times10^{24}$
la02	157	80 862 876	$28\,599\times10^{15}$	$1\,729\times10^{24}$
la03	191	50 910 284	$12\,314\times10^{15}$	745×10^{24}
la04	113	68 208 819	$25\,838\times10^{15}$	$1\,562\times10^{24}$
la05	182	40 229 147	$23\,319\times10^{15}$	$1\,410\times10^{24}$

Table 3
Running times of the brute-force algorithm (measured times in seconds).

Instance	Terminated after x solutions					Extrapolated running time (years)
	10^4	10^5	10^6	10^7	10^8	
ft06	<1	3	25	237	2397	1.06×10^5
la01	<1	3	35	359	3666	7.31×10^{20}
la02	<1	3	35	363	3731	7.44×10^{20}
la03	<1	2	33	323	3253	6.49×10^{20}
la04	<1	3	35	366	3775	7.53×10^{20}
la05	<1	3	35	364	3567	7.11×10^{20}

We implemented the new algorithm including the state space reduction of Section 3.3 with C++ and the tests were performed on a windows 64 bit machine with a 2.66 GHz CPU and 8 GB memory.

In Table 1, we can observe the results obtained. In this table, the first three columns contain the information defining the instance namely, the name, the number of jobs (n) and the number of machines (m). Columns 5 and 6 depict the computational resource requirements for the instances considered. In particular, we present the CPU time in seconds for solving each instance as well as the memory required.

Table 2 contains the observed values and upper bounds for the maximal size of $\overset{\triangle}{\Xi}(S)$ ($\max_S|\overset{\triangle}{\Xi}(S)|$) and for the total number of sequences in the state space ($|\bigcup_S\overset{\triangle}{\Xi}(S)|$).

As we can see, the number of sequences needed to solve each instance is much smaller than the upper bound obtained in Section 4, which gives strong evidence that the bounds presented in the previous section can be improved significantly. For the first instance we were able to run our algorithm without the state space reduction using 2.6 s and 8 MB, having values 16 and 190 592 for $\max_S|\overset{\triangle}{\Xi}(S)|$ and $|\bigcup_S\overset{\triangle}{\Xi}(S)|$ respectively. According to these values even without state space reduction we suspect

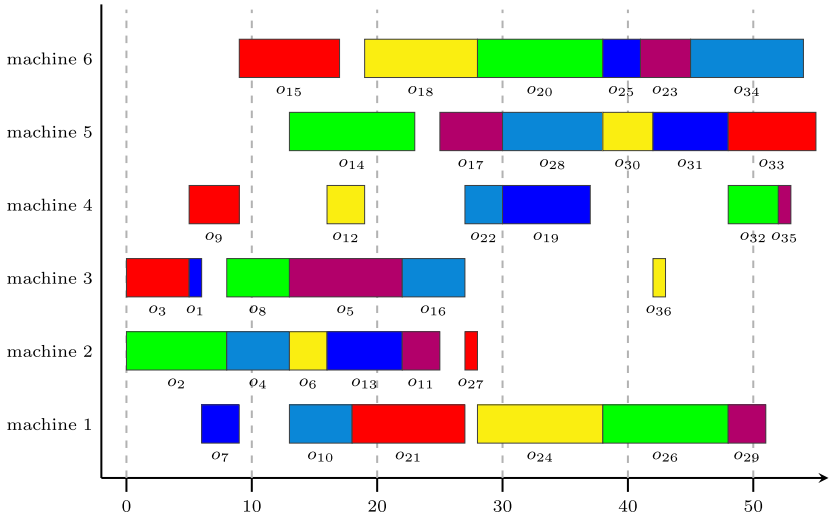

Fig. 5. Optimal solution of *ft*06.

our complexity analysis might be not tight and this worst-case complexity guarantee might largely overestimate the running time.

In order to compare these results with brute-force, we implemented a brute-force algorithm (also in C++) and tested the same instances with this algorithm on the same machine. After 10^i solutions ($i \in \{4,5,6,7,8\}$) we terminated the runs to be able to extrapolate the total running time for the brute-force algorithm.

In Table 3, we can observe that for each step the running time multiplies by a factor 10 as expected. If we extrapolate the running times of 10^8 solutions by multiplying them by $6!^6/10^8$ and $10!^5/10^8$ (respectively for instance *ft06* and for instances *la*) we get a running time of the brute-force algorithm of 1.06×10^5 years for the *ft06* instance and the running time for the five *la* instances has a range of $6.49-7.53 \times 10^{20}$ years.

We should stress here that although using an exponential amount of memory our algorithm can be implemented in such a way that only the optimal value is found saving a constant factor in memory. Nevertheless, for the instances that we could run not only the optimal value, but also an optimal solution was found.

We present our algorithm as one to find the optimal sequence of operations. From this sequence the optimal solution immediately follows. For example, in the case of instance *ft06* we obtained the following optimal sequence:

$o_3o_1o_2o_7o_9o_4o_8o_6o_{15}o_{10}o_{12}o_{13}o_5o_{14}o_{11}o_{21}o_{16}o_{27}o_{18}o_{22}o_{17}o_{19}o_{24}$
$o_{28}o_{20}o_{25}o_{30}o_{36}o_{23}o_{26}o_{31}o_{29}o_{32}o_{35}o_{34}o_{33}$.

This sequence corresponds with the solution in Fig. 5. This solution has the well-known optimal value of 55 (see for instance [17]).

6. Conclusion

In this paper we proposed a Dynamic Programming approach for the classical job-shop scheduling problem. The main achievements of this paper are twofold:

1. We produce an exact algorithm with complexity proven to be exponentially lower than exhaustive enumeration, which is to date the best complexity guarantee of an exact algorithm for the job-shop scheduling problem that we are aware of. Furthermore, our complexity analysis might be untight and it may be possible to improve the bounds presented in the previous section significantly.
2. Despite the theoretical interest, our algorithm proves to work in practice by solving some moderate benchmark instances.

References

[1] Adams J, Balas E, Zawack D. The shifting bottleneck procedure for job shop scheduling. Management Science 1988;34:391–401.
[2] Anderson I. Combinatorics of finite sets. Dover Publications Inc.; 1987.
[3] Applegate D, Cook W. A computational study of job-shop scheduling. ORSA Journal of Computing 1991;3:149–56.
[4] Bellman R. Dynamic programming treatment of the travelling salesman problem. Journal of the Association for Computing Machinery 1962;9:61.
[5] Brucker P, Jurisch B, Sievers B. A branch and bound algorithm for the job-shop scheduling problem. Discrete Applied Mathematics 1994;49:109–27.
[6] Brucker P. Scheduling algorithms. 2nd ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc.; 1998.
[7] Carlier J, Pinson E. Management Science 1989;35:164–76.
[8] Croce FD, Tadei R, Volta G. A genetic algorithm for the job shop problem. Computers and Operations Research 1995;22:15–24.
[9] Dorndorf U, Pesch E. Evolution based learning in a job shop scheduling environment. Computers and Operations Research 1995;22:25–40.
[10] Feige U, Scheideler C. Improved bounds for acyclic job shop scheduling. In: Proceedings of the 30th annual ACM symposium on the theory of computing; 1998. p. 624–33.
[11] Fisher H, Thompson GL. Probabilistic learning combinations of local job-shop scheduling rules. In: Muth JF, Thompson GL, editors. Industrial scheduling. Englewood Cliffs, NJ: Prentice Hall; 1963. p. 225.
[12] Goldberg LA, Paterson M, Srinivasan A, Sweedyk E. Better approximation guarantees for jobshop scheduling. In: Proceedings of the eighth symposium on discrete algorithms; 1997. p. 599–608.
[13] Gromicho J, van Hoorn JJ, Kok AL, Schutten JMJ. Restricted dynamic programming: a flexible framework for solving realistic VRPs. Computers and Operations Research 2012;39:902–9.
[14] Hefetz N, Adiri I. An efficient optimal algorithm for the two-machines unit-time jobshop schedule-length problem. Mathematics of Operations Research 1982;7:354–60.
[15] Held M, Karp RM. A dynamic programming approach to sequencing problems. SIAM Journal of Applied Mathematics 1962;10:196–210.
[16] Jackson JR. Scheduling a production line to minimize maximum tardiness. Technical report, Management Science Research Project, University of California. Los Angeles; 1955.
[17] Jain AS, Meeran S. Deterministic job-shop scheduling: past, present and future. European Journal of Operational Research 1999;113:390–434.
[18] Jansen K, Solis-Oba R, Sviridenko MI. Makespan minimization in job shops: a polynomial time approximation scheme. In: Proceedings of the 31st annual ACM symposium on the theory of computing; 1999. p. 418–26.
[19] Lageweg BJ, Lenstra JK, Rinnooy Kan AHG. Job-shop scheduling by implicit enumeration. Management Science 1977;24:441–50.
[20] Lawrence S. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. GSIA, Carnegie-Mellon University; 1984.
[21] Leighton FT, Maggs BM, Rao SB. Packet routing and job shop scheduling in O(congestion+dilation) steps. Combinatorica 1994;14:167–86.

[22] Leighton FT, Maggs BM, Richa A. Fast algorithms for finding O(congestion+dilation) packet routing schedules. Technical report CMU-CS-96-152, School of Computer Science, Carnegie Mellon University, Pittsburgh, USA; 1996.
[23] Leighton T, Maggs B, Rao S. Universal packet routing algorithms. In: Annual IEEE symposium on foundations of computer science; 1988. p. 256–69.
[24] Lenstra JK, Rinnooy Kan AHG. Computational complexity of discrete optimization problems. Annals of Discrete Mathematics 1979;4:121–40.
[25] Lenstra JK, Rinnooy Kan AHG, Brucker P. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1977;1:343–62.
[26] OR Library, 〈http://people.brunel.ac.uk/~mastjjb/jeb/info.html〉; 2009.
[27] Martin P, Shmoys DB. A new approach to computing optimal schedules for the job-shop scheduling problem. In: Proceedings of the fifth international IPCO conference on integer programming and combinatorial optimization. London, UK: Springer-Verlag; 1996. p. 389–403.
[28] Nowicki E, Smutnicki C. A fast taboo search algorithm for the job shop scheduling problem. Management Science 1996;42:797–813.
[29] Rego C, Duarte R. A filter-and-fan approach to the job shop scheduling problem. European Journal of Operational Research 2009;194:650–62.
[30] Shmoys DB, Stein C, Wein J. Improved approximation algorithms for shop scheduling problems. In: Proceedings of the second annual ACM-SIAM symposium on discrete algorithms, vol. 23; 1994. p. 617–32.
[31] Steinhöfel K, Albrecht A, Wong CK. Two simulated annealing-based heuristics for the job shop scheduling problem. European Journal of Operational Research 1999;118:524–48.
[32] Taillard ED. Parallel taboo search techniques for the job-shop scheduling problem. ORSA Journal on Computing 1994;6:108–17.
[33] Van Laarhoven PJM, Aarts EHL, Lenstra JK. Job shop scheduling by simulated annealing. Operations Research 1992;40:113–25.
[34] Woeginger GJ. Exact algorithms for NP-hard problems: a survey. In: Jünger M, Reinelt G, Rinaldi G, editors. Combinatorial optimization—eureka, you shrink!. New York, NY, USA: Springer-Verlag New York, Inc.; 2003. p. 185–207.
[35] Woeginger GJ. Open problems around exact algorithms. Discrete Applied Mathematics 2008;156:397–405.
[36] Zhang CY, Li P-G, Rao Y-Q, Guan Z-L. A very fast TS/SA algorithm for the job shop scheduling problem. Computers and Operations Research 2008;35:282–94.