Engineering Applications of Artificial Intelligence ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at SciVerse ScienceDirect

Engineering Applications of Artificial Intelligence

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

Solving the $Fm|block|C_{max}$ problem using Bounded Dynamic Programming

Joaquín Bautista [a,*], Alberto Cano [a], Ramon Companys [b], Imma Ribas [b]

[a] Nissan Chair, Escola Tècnica Superior d'Enginyeria de Barcelona, Universitat Politècnica de Catalunya, Barcelona, Spain
[b] Departament d'Organització d'Empreses, Escola Tècnica Superior d'Enginyeria de Barcelona, Universitat Politècnica de Catalunya, Barcelona, Spain

A R T I C L E   I N F O

Article history:
Received 6 May 2011
Received in revised form
27 July 2011
Accepted 1 September 2011

Keywords:
Blocking flow shop
Scheduling
Production
Logistics
Dynamic programming
Meta-heuristics

A B S T R A C T

We present some results attained with two variants of Bounded Dynamic Programming algorithm to solve the $Fm|block|C_{max}$ problem using as an experimental data the well-known Taillard instances. We have improved the best known solutions for 17 of Taillard's instances, including the 10 instances from set 12.

1. Introduction

The *flowshop scheduling problem* (FSP) is one of the problems which has received most attention over the last fifty years and which continues to receive the attention of professionals and researchers due to the huge variety of productive contexts it makes it possible to model. In the FSP, a set of n jobs must be processed in a set of m machines. All the jobs must be processed in all the machines following the same order, starting in machine 1 and finishing in machine m. Each job, $i \in I$, requires a processing time, $p_{i,k} > 0$, in each of the machines, $k \in K$. The aim is to find a job processing sequence, which optimizes a given criterion.

In the most popular version of the problem, known as *permutation flowshop scheduling problem* (PFSP), the storage capacity between two consecutive phases of the process, where the jobs can wait until they can be processed by the following machine, is assumed to be unlimited. However, there are many productive systems, in diverse sectors, such as fine chemicals, pharmaceuticals, plastic molding, electronics, steel, food, etc.; in general, all those systems in which there is a production line with no mechanical drag and therefore a cyclical repetition of operations, in which storage capacity is limited. If we assume there to be no possibility of storage between two successive phases of the process, a major structural change takes place in the behavior of the system, since a part cannot leave the machine which is processing it until the following machine is free. If this is not the case, the part is forced to stay in the previous machine, blocking it and preventing it from performing operations on other parts. This variant is known as *blocking flowshop scheduling problem* (BFSP) and is the one we are going to consider in this article. If the intermediate storage capacity is limited, the problem can also be reduced to a BFSP in which each storage space is treated as a dummy machine with a processing time equal to zero (McCormick et al., 1989).

In this article, we discuss the BFSP with the aim of minimizing the maximum completion time of jobs or makespan. Making use of the notation proposed by Graham et al. (1979), the problem considered is denoted by $Fm|block|C_{max}$ (and the PFSP by $Fm|prmu|C_{max}$). The research carried out on this problem is not very extensive. A good review of flowshop with blocking and no waits in the process can be found in Hall and Sriskandarajah (1996), where they also demonstrated, using a result from Papadimitriou and Kanellakis (1980), that the problem $Fm|block|C_{max}$ for $m \geq 3$ machines is strongly NP-hard. However, for $m=2$, Reddi and Ramamoorthy (1972) demonstrated the existence of a polynomial algorithm which reaches the optimal solution to the $Fm|block|C_{max}$ problem. The reason lies in the fact that the $F2|block|C_{max}$ problem can be reduced to a *traveling salesman problem* (TSP) with $n+1$ cities $(0,1,2,\ldots n)$. The sequence

* Correspondence to: Nissan Chair, Escola Tècnica Superior d'Enginyeria de Barcelona, Universitat Politècnica de Catalunya, Avda. Diagonal 647, 08028 Barcelona, Spain. Tel.: +34 934011703; fax: +34 934016054.
E-mail addresses: joaquin.bautista@nissanchair.com (J. Bautista), alberto.cano-perez@upc.edu (A. Cano), ramon.companys@upc.edu (R. Companys), imma.ribas@upc.edu (I. Ribas).
URLS: http://www.nissanchair.com (J. Bautista), http://www.nissanchair.com (A. Cano).

0952-1976/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2011.09.001

of cities in an optimal circuit is associated with an optimal permutation of the parts in the original problem. Gilmore and Gomory (1964) proposed a polynomial algorithm to solve the TSP; this algorithm has a time complexity of $O(n\log n)$ (Gilmore et al., 1991).

Given the NP-hard nature of the problem, few exact procedures have been proposed to solve it. Levner (1969) presented one of the first works on this problem. Levner proposed a branch-and-bound algorithm, associating to each instance and permutation a graph, and obtaining lower bounds of the branch-and-bound tree nodes from the length of paths on this graph. Other branch-and-bound algorithms were presented by Suhami and Mah (1981), Ronconi and Armentano (2001) and Ronconi (2005). Companys and Mateo (2007) presented the LOMPEN algorithm, another branch-and-bound type approach, in which they used the reversibility property of the problems $Fm|prmu|C_{max}$ and $Fm|block|C_{max}$. Both in Ronconi (2005) and in Companys and Mateo (2007), the Taillard instances were used, being considered as instances of the $Fm|block|C_{max}$ problem, to assess the efficiency of the procedure.

On the other hand, more effort has been made in the development of heuristic procedures for finding quality solutions in a timely fashion. McCormick et al. (1989) studied a special cyclical case and presented the constructive heuristic, *profile fitting*. Leisten (1990) adapted certain procedures used in the PFSP and concluded that the NEH heuristic, proposed by Nawaz et al. (1983), suitably adapted to the problem, was the one which obtained the best results. Abadi et al. (2000) presented an improvement heuristic to minimize cycle time in a flowshop with blocking, which can also be used in the $Fm|block|C_{max}$ problem. Using the aforementioned idea, Caraffa et al. (2001) developed a *genetic algorithm* (GA) to solve high dimension flowshop problems, among which the $Fm|block|C_{max}$ problem was a special case, and obtained better results than with the heuristic of Abadi et al. (2000). Ronconi (2004) proposed two variants of the NEH heuristic, which he called MME and PFE, in which he proposed replacing the LPT ordination for the MM or PF ordination. Ribas et al. (2011) took up the constructive algorithm MME again and showed that, combined with the reversibility property, it was more efficient than other procedures based on the NEH scheme. Grabowski and Pempera (2007) presented two algorithms based on *tabu search* (TS) (TS and TS+M). Wang et al. (2006) proposed an hybrid genetic algorithm (HGA), Liu et al. (2008) an algorithm based on *particle swarm optimization* (HPSO), Qian et al. (2009) proposed an algorithm based on *differential optimization* (DE) and Wang et al. (2010) an *hybrid discrete differential evolution* (HDDEA) algorithm, which exceeded the efficiency of the TS+M algorithm of Grabowski and Pempera (2007). Finally, Ribas et al. (2011) presented an *iterated greedy algorithm* (IGA) more efficient than the HDDEA and an updated list of the best solutions for the Taillard instances.

For this manuscript, we used a procedure based on *Bounded Dynamic Programming* (BDP). This procedure combines features of dynamic programming (determination of extreme paths in graphs) with features of branch and bound algorithms. The principles of Bounded Dynamic Programming have been described by Bautista et al. (1996). Previous work on similar approaches has been done by Morin and Marsten (1976) and Marsten and Morin (1978), and extended by Carraway and Schmidt (1991).

In the present manuscript, our proposals are:

1. A dynamic programming based procedure to solve the $Fm|block|C_{max}$ problem.
2. General bounds for C_{max} for this problem. These general bounds take into account machines and jobs and also may depend on a partial subsequence of jobs already sequenced.
3. An application of the proposed procedure to the 12 sets of instances from the literature (Taillard's benchmark instances).

As results, we have improved the best known solutions in 17 instances from a total of 120. In particular, we have obtained better solutions in the 10 instances of the set 12 from Taillard, with 500 jobs and 20 machines.

This manuscript is organized as follows. Section 2 presents the problem description. Section 3 describes the graph associated with the problem under consideration and establishes dominance properties between their vertices. Section 4 proposes general and partial bounds on the C_{max} value shown by the sequences. Section 5 introduces a procedure based on BDP to solve the problem under consideration and an example. Section 6 describes the computational experiments performed and presents the results. Finally, Section 7 shows the conclusions of the study.

2. Problem description

At time zero, n jobs must be processed, in the same order, on each of m machines. Each job goes from machine 1 to machine m. The processing time for each operation is $p_{i,k}$, where $k\in K=\{1,2,\ldots,m\}$ denotes a machine and $i\in I=\{1,2,\ldots,n\}$ a job. Setup times are included in processing times. These times are fixed, known in advance and positive. The objective function considered is the minimization of the makespan (C_{max}).

Given a permutation, $\boldsymbol{\pi}$, of the n jobs, $[t]$ indicates the job that occupies position t in the sequence. For example, in $\boldsymbol{\pi}=(3,1,2)$ $[1]=3$, $[2]=1$, $[3]=2$. For this permutation, in every machine, job 2 occupies position 3. In a feasible schedule associated to a permutation, let $s_{k,t}$ be the beginning of the time destined in machine k to job that occupies position t and $e_{k,t}$ the time of the job that occupies position t releases machine k. The $Fm|prmu|C_{max}$ problem can be formalized as follows:

$$s_{k,t}+p_{[t],k}\leq e_{k,t}\quad k=1,2,\ldots,m;\quad t=1,2,\ldots,n \tag{1}$$

$$s_{k,t}\geq e_{k,t-1}\quad k=1,2,\ldots,m;\quad t=1,2,\ldots,n \tag{2}$$

$$s_{k,t}\geq e_{k-1,t}\quad k=1,2,\ldots,m;\quad t=1,2,\ldots,n \tag{3}$$

$$C_{max}=e_{m,n} \tag{4}$$

Being $e_{k,0}=0\ \forall k$, $e_{0,t}=0\ \forall t$, the initial conditions.

The schedule is semi-active if Eq. (1) is written as $s_{k,t}+p_{[t],k}=e_{k,t}$ and Eqs. (2) and (3) are summarized as $s_{k,t}=\max\{e_{k,t-1},e_{k-1,t}\}$.

When there is no storage space between stages, $Fm|block|C_{max}$ problem, if a job i finishes its operation on a machine k and if the next machine, $k+1$, is still busy on the previous job, the completed job i has to remain on the machine k blocking it. This condition requires an additional Eq. (5) in the formulation of the problem

$$e_{k,t}\geq e_{k+1,t-1}\quad k=1,2,\ldots,m;\quad t=1,2,\ldots,n \tag{5}$$

The initial condition $e_{m+1,t}=0$ $t=1,2,\ldots,n$ must be added.

The schedule obtained is semi-active if Eqs. (1) and (5) are summarized as (6):

$$e_{k,t}=\max\{s_{k,t}+p_{[t],k},e_{k+1,t-1}\}\quad\forall k,\forall t \tag{6}$$

Consequently, the $Fm|prmu|C_{max}$ problem can be seen as a relaxation of the $Fm|block|C_{max}$ problem.

3. Graph associated with the problem

Similar to Bautista et al. (1996) and Bautista and Cano (2011), we can build a linked graph without loops or direct cycles of $T+1$

levels. At level 0 of the graph, there is only one vertex $J(0)$. The set of vertices in level t ($t=0,...,T$) will be noted as $J(t)$, and are associated to the partial sequences of t jobs. Let $J(t,j)$ ($j=1,...,|J(t)|$) a vertex j of level t, which is represented by the triad ($\vec{q}(t,j)$, $\vec{e}(t,j)$, $C_{max}(t,j)$), where:

- $\vec{q}(t,j)=(q_1(t,j),\ldots,q_n(t,j))$ is the vector of scheduled jobs (or not) in the jth vertex of the level t, where $q_i(t,j), \forall i \in I$ ($i=1,...,n$) is the ith component of the vector $\vec{q}(t,j)$ that takes the value 1 if the job i has been completed, and the value 0 otherwise.
- $\vec{e}(t,j)=(e_1(t,j),\ldots,e_m(t,j))$ is the vector of completion times of the last scheduled job in each machine.
- $C_{max}(t,j)$ is the completion time of the last scheduled job in vertex j of level t.

The vertex $J(t,j)$ has the following properties:

$$\sum_{i=1}^{n} q_i(t,j)=t \quad (7)$$

$$q_i(t,j) \in \{0,1\} \forall i \quad (8)$$

$$C_{max}(t,j)=e_m(t,j) \quad (9)$$

In short, a vertex $J(t,j)$ will be represented as follows:

$$J(t,j)=\{(t,j), \vec{q}(t,j), \vec{e}(t,j)\} \quad (10)$$

Initially, we may consider that at level t, $J(t)$ contains the vertices associated with all of the sub-sequences that can be built with t jobs that satisfy properties (7) and (8). However, it is easy to reduce the cardinal that $J(t)$ may present a priori, establishing the following dominance and equivalence rules:

$$J(t,j) \prec J(t,j') \Leftrightarrow [\vec{q}(t,j)=\vec{q}(t,j')] \wedge [\vec{e}(t,j)<\vec{e}(t,j')] \quad (11)$$

$$J(t,j) \equiv J(t,j') \Leftrightarrow [\vec{q}(t,j)=\vec{q}(t,j')] \wedge [\vec{e}(t,j)=\vec{e}(t,j')] \quad (12)$$

With these rules, we can reduce the search space for solutions in the graph. Therefore, at level t of the graph, $J(t)$ will contain the vertices associated with non-dominated and non-equivalent sub-sequences, and at level T, $J(T)$ will contain all the vertices associated with non-equivalent and non-dominated completed sequences.

A transition arc through the type of job i exists between vertices $J(t,j)$ of level t and vertex $J(t+1,j_i)$ of level $t+1$ ($J(t,j) \xrightarrow{i} J(t+1,j_i)$) in the following case:

$$\vec{q}(t,j) \prec \vec{q}(t+1,j_i) \quad (13)$$

For vertex $J(t+1,j_i)$ to be completely defined through the transition from $J(t,j)$, it is necessary to determine:

$$J(t+1,j_i)=\{(t+1,j_i), \vec{q}(t+1,j_i), \vec{e}(t+1,j_i)\} \quad (14)$$

as follows:

$$q_i(t+1,j_i)=1 \quad (15)$$

$$q_h(t+1,j_i)=q_h(t,j) \quad \forall h : h \neq i \in I \quad (16)$$

$$e_k(t+1,j_i)=\max\{e_k(t,j)+p_{i,k}, e_{k-1}(t+1,j_i), e_{k+1}(t,j)\} \quad \forall k \in K \quad (17)$$

where $e_0(t+1,j_i)=0$.

Fig. 1 illustrates a scheme of the state graph associated to the BFSP. The initial vertices $J(t,j)$ and $J(t,j')$ of stage t are developed by scheduling the jobs i and i', respectively. This development results in a unique vertex, $J(t+1,j_i)$, using the dominance and equivalence rules (11) and (12); then, the properties of the vertex $J(t+1,j_i)$ of stage $t+1$ can be determined; finally, this vertex can be developed by scheduling a job i'', giving as a result the vertex $J(t+2,j_{i_{i''}})$ of the stage $t+2$.

Indirectly, contribution to the partial C_{max} generated in the transition from $J(t,j)$ to $J(t+1,j_i)$ may be calculated by incorporating the job i to the latter vertex, as follows:

$$a((t,j)\rightarrow(t+1,j_i))=e_m(t+1,j_i)-e_m(t,j) \quad (18)$$

Under these conditions, finding a sequence that optimizes the total C_{max} is equivalent to finding an optimum path from vertex $J(0)$ to the set of vertices $J(T)$ of level T.

Therefore, any algorithm of extreme paths in the graphs is valid for finding solutions to the proposed problem. However, realistic industrial problems where n and m are large give rise to graphs with a large number of vertices. Therefore, we recommend resorting to procedures that do not explicitly require the presence of all of the vertices for calculation.

4. Bounding the values of the sequences

First, we establish general bounds for C_{max}, and then we establish the bounds associated with the path for building (complement) when a segment or subsequence of t members has already been built.

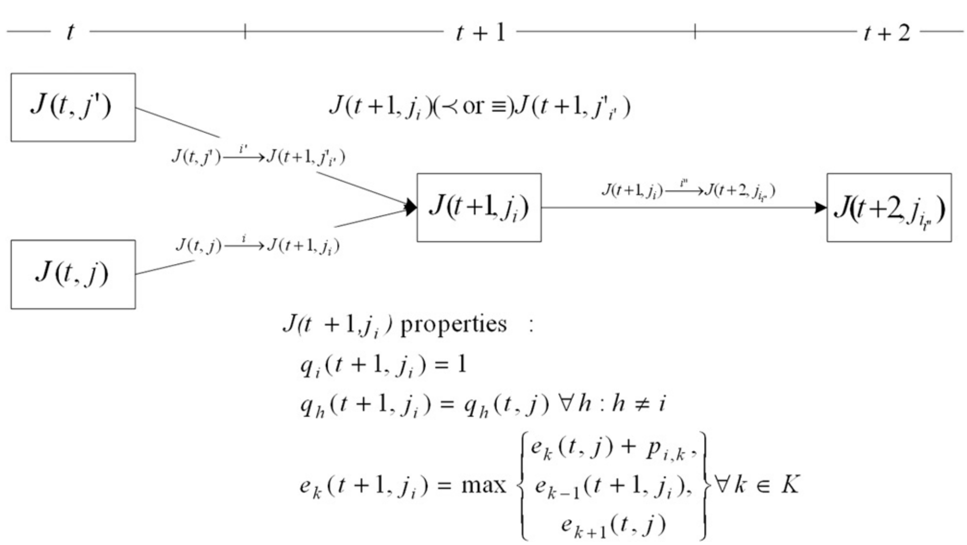

Fig. 1. Scheme of transitions through the state graph associated to the BFSP.

Please cite this article as: Bautista, J., et al., Solving the $Fm|block|C_{max}$ problem using Bounded Dynamic Programming. Engineering Applications of Artificial Intelligence (2011), doi:10.1016/j.engappai.2011.09.001

In this paper, we use the bounds proposed by Lageweg et al. (1978) for the PFSP. These bounds have been adapted as general and partial bounds for the BFSP, considering that the PFSP is a relaxation of the BFSP.

4.1. General bounds for C_{max}

If we account for the machines independently, then we can write the following:

$$LB1(k) = \sum_{i=1}^{n} p_{i,k} + \min_{(i,h) \in I: i \neq h} \left\{ \sum_{k'=1}^{k-1} p_{i,k'} + \sum_{k'=k+1}^{m} p_{h,k'} \right\} \forall k \in K \quad (19)$$

which is a bound of C_{max}, through the machine k.

Therefore, considering all machines, we have the following:

$$LB1 = \max_{k \in K}\{LB1(k)\} \quad (20)$$

In the same manner, we can also consider a bound for C_{max} through the job i:

$$LB2(i) = \sum_{k=1}^{m} p_{i,k} + \sum_{h \in I: h \neq i} \min_{k \in K}\{p_{h,k}\} \; \forall i \in I \quad (21)$$

Considering all the jobs, then we have

$$LB2 = \max_{i \in I}\{LB2(i)\} \quad (22)$$

4.2. Bound of C_{max} through a given segment

Let us assume that we have built a path from $J(0)$ to vertex $J(t,j)$, and thus we have the information $\vec{q}(t,j)$ and $\vec{e}(t,j)$.

To complete a sequence up to level T, we will need to link with $J(t,j)$, $T-t$ vertices, associated each of them with a different unscheduled job.

Under these conditions, we can delimit C_{max} through the vertex $J(t,j)$ adapting the overall bound $LB1$.

$$LB1(t,j) = \max_{k \in K} \left\{ e_k(t,j) + \sum_{\substack{i \in I: \\ q_i(t,j)=0}} p_{i,k} + \min_{\substack{i \in I: \\ q_i(t,j)=0}} \left\{ \sum_{k'=k+1}^{m} p_{i,k'} \right\} \right\} \quad (23)$$

If we focus on the jobs, we will have

$$LB2(t,j) = e_1(t,j) + \max_{\substack{i \in I: \\ q_i(t,j)=0}} \left\{ \sum_{k=1}^{m} p_{i,k} + \sum_{\substack{h \in I-\{i\}: \\ q_h(t,j)=0}} \min_{k \in K}\{p_{h,k}\} \right\} \quad (24)$$

5. The use of Bounded Dynamic Programming

The procedure we propose (from Bautista et al., 1996; Bautista and Pereira, 2009; Bautista and Cano, 2011) is called Bounded Dynamic Programming (BDP) and consists of generating a part of the graph described in Section 3 from level 0 to level T, one level at a time.

The generated vertices may potentially form a part of an optimum path (from 0 to T) that is based on the construction of an optimum segment of t levels, from $J(0)$ to $J(t,j)$, and on the evaluation of the bound of C_{max} to reach level T, for example $LB1(t,j)$.

The procedure only keeps the information of two consecutive levels in memory, t and $t+1$ ($t=0,\ldots,T-1$), for which it uses the following lists $\Lambda(t)$ and $\Lambda(t+1)$, respectively:

- List $\Lambda(t)$ contains information about the vertices consolidated in level t that can potentially form part of an optimum or good quality path.
- List $\Lambda(t+1)$ contains the vertices that are tentatively generated one-by-one from each vertex of list through the possible transitions between levels t and $t+1$.

A record $\lambda(J(t,j))$ of list $\Lambda(t)$, $\lambda(J(t,j)) \in \Lambda(t)$, is composed of three elements:

$$\lambda(J(t,j)) = \{J(t,j), LB1(t,j), \Gamma^{-}(J(t,j))\} \quad (25)$$

where $\Gamma^{-}(J(t,j))$ is the vertex of level $t-1$ ancestor of $J(t,j)$.

Although the use of $\Lambda(t)$ and $\Lambda(t+1)$ notably reduces memory needs, the number of vertices that can be generated for a level can be very large. Therefore, we impose a limitation on the number of $H(t)$ vertices stored in level t. This limitation, called window width, is represented as H, $H(t) \leq H$ ($t=1,\ldots,T$). In addition, we set the maximum number of transitions from a vertex in level t to the value $n-t$.

To obtain an initial solution with value Z_0 (the upper bound of the value of C_{max}), it is sufficient to use a Greedy procedure, a local search, or *BDP* with a small window width, e.g., $H=1$.

We have developed two variants based on BDP:

1. The ordered pair of values $(LB1(t,j), e_m(t,j))$ is used as priority rule or guide (GZ) to obtain solutions: a partial solution is more promising than another when it has the best bound for C_{max} ($LB1(t,j)$). In case of tie between two partial solutions (equal $LB1(t,j)$), the partial solution with less $e_m(t,j)$ will be considered the best.
2. In the Variant 2, the ordered pair of values $(e_m(t,j), LB1(t,j))$ is used as priority rule or guide (GZ): a partial solution is more promising than another when it has less value for his partial C_{max} (i.e. $e_m(t,j)$). In case of tie between two partial solutions (equal $e_m(t,j)$), the partial solution with less $LB1(t,j)$ will be considered the best.

Evidently, some vertices tentatively generated in level t will not be recorded in list $\Lambda(t+1)$.

In effect, we use the following rules:

1. We "remove" an $J(t+1,j_i)$ vertex generated when the value of its lower bound, $LBZ=LB1(t+1,j_i)$, is greater than or equal to the value of a known solution Z_0 (upper bound for C_{max}), because it is not possible to obtain a solution with a better value than Z_0 through $J(t+1,j_i)$.
2. We "reject" an $J(t+1,j_i)$ vertex generated when there is a record $\lambda(J(t+1,h)) \in \Lambda(t+1)$ with a vertex that dominates or is equivalent to $J(t+1,j_i)$: $J(t+1,h)(\prec \vee \equiv)J(t+1,j_i)$.
3. We "discard" the placement of an $J(t+1,j_i)$ vertex generated on the list $\Lambda(t+1)$ when the list is full ($H(t+1)=H$) and $J(t+1,j_i)$ has a GZ (Variant 1: $GZ=(LB1(t+1,j_i), e_m(t+1,j_i))$ or Variant 2: $GZ=(e_m(t+1,j_i), LB1(t+1,j_i))$) that is greater than or equal to the largest value of the priority rule or guide ($Variant\ 1: GZ_{max} = (LB1(t+1,h_{max}), e_m(t+1,h_{max}))$ or $Variant\ 2: GZ_{max} = (e_m(t+1,h_{max}), LB1(t+1,h_{max}))$) of the vertices already recorded in $\Lambda(t+1)$, although an optimum path may pass through $J(t+1,j_i)$.
4. The $J(t+1,j_i)$ vertex generated "replaces" a vertex $J(t+1,h)$ recorded on list $\Lambda(t+1)$, when $J(t+1,j_i)$ dominates $J(t+1,h)$, or when $J(t+1,j_i)$ has a GZ that is lower than $J(t+1,h)$ and $H(t+1)=H$, although the optimum path may pass through the moved vertex.

Under these conditions, we can write the following algorithm (Variants 1 and 2):

```
BDP–Fm|block|Cmax
Input: T, |I|, |K|, d_i(∀i∈I), p_i,k(∀i∈I,∀k∈K), Z0, H
Output: list of sequences obtained by BDP (Λ(T))
Initialization: t = 0;  LBZmin = ∞
while (t < T) do
      t=t+1
        While (list of consolidated vertices in level t−1
(Λ(t−1)) not empty) do
            Select_vertex (t)
            Develop_vertex (t)
            Λ(t)←Filter_vertices (Z0, H, LBZmin)
      end while
      End_level()
end while
end BDP–Fm|block|Cmax
```

As can be seen in the pseudocode of the procedure, The BDP algorithm uses the following functions:

- Select_vertex (t): a vertex of level $t-1$ is selected in the order established in the consolidated list $\Lambda(t-1)$. This order depends on the variant of the algorithm used. The Variant 1 sorts the vertices according to a non-decreasing order of $LB1$ and, to break ties, according to a non-decreasing order of the partial C_{max} of the subsequence associated to the selected vertex. Instead, in the Variant 2, vertices are sorted according to a non-decreasing order of the partial C_{max} and, in case of ties, according to a non-decreasing order of $LB1$.
- Develop_vertex (t): the vertex selected by the function Select_ vertex (t) is developed by adding an unscheduled job in the associated subsequence of the selected vertex. During this development $LB1$, the partial C_{max} and the completion times of the added job, in all machines, are determined, taking into account the original subsequence. Logically, the development of a vertex implies to evaluate all the possible extensions of the subsequence associated with the selected vertex, by considering, one by one, all the unscheduled jobs.
- Filter_vertices (Z_0,H,LBZ_{min}): a maximum number (H) of extensions are selected between all the extensions resulting by the function Develop_vertex (t). The extensions selected are those that have a better value of partial C_{max} or $LB1$ according to the variant of the algorithm used for the list $\Lambda(t)$. Moreover, some vertices may be discarded on the selection process for the following reasons: (1) the vertex is dominated by a more promising one or is equivalent to another one; and (2) the extension has a value of $LB1$ greater than or equal that the best known solution Z_0. Throughout the process, the lower value of $LB1$, between all the extensions that have not been selected, is kept: LBZ_{min}.
- End_level (): the selected extensions by the function Filter_ vertices (Z_0,H,LBZ_{min}) are consolidated at the level t, confirming the list $\Lambda(t)$ (a maximum of H vertices).

When the procedure ends, we can initially find two possible situations:

- List $\Lambda(T)$ is empty, which means that we are unable to find a solution with a value less than Z_0.
- List $\Lambda(T)$ is not empty, which means that the records contained in $\Lambda(T)$, $\lambda(T,h)\in\Lambda(T)$, are associated with vertices, $J(T,h)$, whose C_{max} is $e_m(T,h)<Z_0$. In this case, we can regressively reconstruct a sequence from any of these vertices with a better value than Z_0 using the $\Lambda(t)$ list and the ancestors of the vertices.

In addition, we can guarantee that we are able to build an optimum sequence from the $\lambda(T,h)\in\Lambda(T)\neq\{\emptyset\}$ records in any of the following cases:

$$\text{Case 1}: \max_{0\le t\le T}\{H(t)\}<H \tag{26}$$

$$\text{Case 2}: (\max_{0\le t\le T}\{H(t)\}=H)\wedge(e_m(T,h)\le LBZ_{min}) \tag{27}$$

LBZ_{min} corresponds to the value of the "discarded" or "replaced" vertex during the procedure with lower bound LBZ.

In any other case, the procedure is heuristic.

Consider the following example to illustrate the use of the BDP procedure: there are four jobs (n=4: A, B, C, D). The jobs are processed in three machines (m=3: m_1, m_2 and m_3), and the processing times, $p_{i,k}$, of each job (i=1,…,4) at each machine (k=1,…,3), are indicated in Table 1.

The objective is to obtain an optimal sequence under the conditions of the $Fm|block|C_{max}$ problem.

Fig. 2 illustrates an application of the proposed procedure (Variant 1) to the example using an initial solution Z_0=25 and a window width H=8. In the graph associated with Fig. 2 we can see the following:

(1) At level t=2, the vertices (A–B) and (B–A) are removed because both presents a lower bound for C_{max} (LB) greater than or equal to Z_0=25.

(2) At level t=2, the vertices (A–C), (A–D), (B–C), (B–D) and (C–D) are dominated by vertices (C–A), (D–A), (C–B), (D–B) and (D–C), respectively. For example, the vertex (A–C) is dominated by vertex (C–A), because all the completion instants of job C (second job in A–C) in all the machines are greater than or equal than the completion instants of the job A (second job in C–A). (vertex (A–C): $e_{1,2}$=9, $e_{2,2}$=12, $e_{3,2}$=15; dominated by vertex (C–A): $e_{1,2}$=9, $e_{2,2}$=12, $e_{3,2}$=14).

(3) At level t=3, the vertices (C–A–B), (C–B–A), (D–A–B) and (D–B–A) are removed, because all of them presents a LB greater than or equal to Z_0=25.

(4) At level t=3, the vertices (C–A–D), (C–B–D), (D–A–C) and (D–B–C) are dominated by vertices (D–A–C), (D–C–B), (D–C–A) and (D–C–B), respectively.

(5) At level t=4, the vertex (D–C–B–A) with LB=24 is dominated by vertex (D–C–A–B) with LB=23, because all the completion instants of job A (fourth job in D–C–B–A) in all the machines are greater than or equal than the completion instants of the job B (fourth job in D–C–A–B). (vertex (D–C–B–A): $e_{1,2}$=19, $e_{2,2}$=22, $e_{3,2}$=24; dominated by vertex (D–C–A–B): $e_{1,2}$=19, $e_{2,2}$=22, $e_{3,2}$=23).

(6) At level t=4, the vertex (D–C–A–B) represents an optimal sequence with value C_{max}=23. The shortest path in the graph in Fig. 2 shows highlighting in black, the arcs between the vertices.

Table 1
Processing times ($p_{i,k}$); where A, B, C and D corresponds to i=1 to 4 (n=4) and m_1, m_2 and m_3 correspond to k=1 to 3 (m=3)

	A	B	C	D
m_1	4	6	5	4
m_2	3	3	3	4
m_3	2	1	3	2

Please cite this article as: Bautista, J., et al., Solving the $Fm|block|C_{max}$ problem using Bounded Dynamic Programming. Engineering Applications of Artificial Intelligence (2011), doi:10.1016/j.engappai.2011.09.001

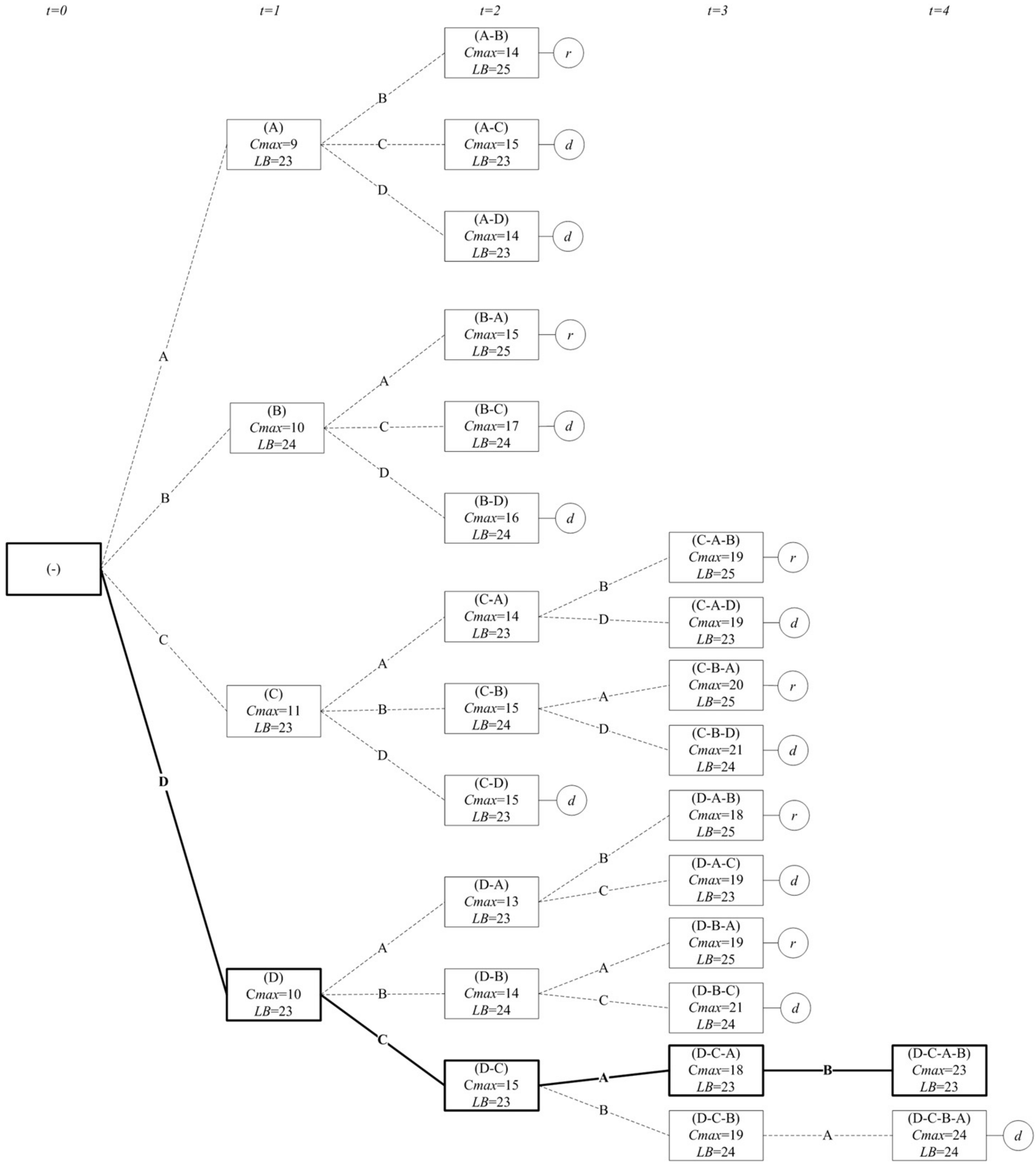

Fig. 2. Graph for the example. In each vertex, the following quantities can be found: the subsequence of jobs, the value of partial C_{max} associated with the subsequence (C_{max}) and the lower bound of the total C_{max} (*LB*). The abbreviations “d” and “r” symbolize “dominated” and “removed”, respectively.

6. Computational experiment

We have performed an operation test with the 12 sets from Taillard's benchmark instances (Taillard, 1993). Taillard's benchmark instances consist in 120 instances, grouped in 12 sets. Each set has 10 instances, each of them with the same number of jobs and machines. The number of jobs goes from 20 (set 1) to 500 (set 12) and the number of machines goes from 5 (set 1) to 20 (set 12).

To obtain solutions, we have used two variants of BDP programmed in C++, compiled with gcc v. 4.01, running on an Apple Macintosh iMac computer with an Intel Core i7 2.93 GHz processor and 8 GB RAM using MAC OS X 10.6.4. Neither the

Table 2
Solutions offered by BDP for each window width (from H_1 to H_8). Best values for *RPD* obtained.

	Ins.	*Best lit.*	$H=1$ C_{max}	$H=10$ C_{max}	$H=50$ C_{max}	$H=100$ C_{max}	$H=250$ C_{max}	$H=500$ C_{max}	$H=750$ C_{max}	$H=1000$ C_{max}	*Best found*	*Best RPD*
Set 1	1	1374	1640	1513	1441	1423	1390	1380	1380	1380	1380^{+}	0.44
	2	1408	1725	1549	1474	1459	1450	1442	1432	1431	1431^{+}	1.63
	3	1280	1667	1471	1353	1330	1326	1314	1302	1302	1302^{+}	1.72
$n=20$	4	1448	1563	1562	1543	1466	1456	1456	1456	1456	1456	0.55
$m=5$	5	1341	1512	1424	1400	1369	1367	1357	1350	1350	1350	0.67
	6	1363	1515	1470	1395	1393	1393	1385	1385	1385	1385	1.61
	7	1381	1467	1407	1407	1396	1396	1396	1395	1393	1393	0.87
	8	1379	1608	1524	1392	1392	1386	1386	1386	1386	1386	0.51
	9	1373	1461	1432	1410	1410	1403	1403	1389	1389	1389	1.17
	10	1283	1421	1311	1307	1307	1293	1293	1293	1293	1293	0.78
Set 2	11	1698	2006	1807	1768	1762	1741	1741	1731	1731	1731	1.94
	12	1833	2116	1974	1974	1909	1909	1897	1895	1883	1883	2.73
	13	1659	1781	1728	1695	1687	1687	1684	1684	1683	1683	1.45
$n=20$	14	1535	1791	1714	1640	1587	1587	1579	1579	1576	1576	2.67
$m=10$	15	1617	1978	1780	1738	1707	1707	1667	1667	1667	1667	3.09
	16	1590	1830	1710	1611	1611	1610	1610	1610	1610	1610	1.26
	17	1622	1818	1740	1725	1722	1691	1691	1681	1681	1681	3.64
	18	1731	2133	1859	1796	1777	1761	1760	1752	1749	1749^{+}	1.04
	19	1747	1962	1854	1854	1768	1755	1755	1755	1755	1755	0.46
	20	1782	2100	1933	1922	1890	1829	1829	1829	1829	1829	2.64
Set 3	21	2436	2772	2644	2640	2622	2567	2551	2551	2551	2551	4.72
	22	2234	2760	2544	2429	2367	2350	2326	2315	2315	2315	3.63
	23	2479	2813	2730	2705	2665	2663	2651	2644	2627	2627	5.97
$n=20$	24	2348	2733	2480	2440	2429	2419	2403	2388	2388	2388	1.70
$m=20$	25	2435	2886	2740	2621	2602	2553	2534	2534	2534	2534	4.07
	26	2383	2744	2532	2492	2492	2492	2461	2461	2461	2461	3.27
	27	2390	2956	2715	2672	2603	2603	2550	2518	2497	2497^{+}	4.48
	28	2328	2792	2596	2574	2543	2522	2522	2522	2522	2522	8.33
	29	2363	3036	2570	2545	2494	2494	2483	2483	2483	2483	5.08
	30	2323	2698	2561	2442	2404	2404	2367	2367	2360	2360	1.59
Set 4	31	3002	3276	3146	3124	3096	3078	3066	3066	3066	3066	2.13
	32	3201	3481	3341	3267	3267	3253	3253	3253	3251	3251	1.56
	33	3011	3235	3146	3108	3081	3081	3081	3081	3077	3077	2.19
$n=50$	34	3128	3554	3261	3261	3215	3187	3181	3181	3181	3181	1.69
$m=5$	35	3166	3471	3257	3226	3226	3226	3226	3216	3216	3216	1.58
	36	3169	3530	3365	3360	3317	3317	3284	3284	3279	3279	3.47
	37	3013	3383	3188	3124	3098	3096	3096	3096	3090	3090	2.56
	38	3073	3480	3180	3125	3125	3125	3125	3125	3125	3125	1.69
	39	2908	3256	3107	3076	3005	3004	2986	2971	2971	2971	2.17
	40	3120	3434	3277	3217	3182	3171	3171	3163	3163	3163	1.38
Set 5	41	3638	4139	3911	3837	3744	3744	3730	3730	3730	3730	2.53
	42	3507	3914	3701	3701	3647	3624	3624	3585	3571	3571	1.82
	43	3488	3982	3848	3754	3754	3672	3642	3623	3623	3623	3.87

Table 2 (*continued*)

	Ins.	*Best lit.*	$H=1$ C_{max}	$H=10$ C_{max}	$H=50$ C_{max}	$H=100$ C_{max}	$H=250$ C_{max}	$H=500$ C_{max}	$H=750$ C_{max}	$H=1000$ C_{max}	*Best found*	*Best RPD*
$n=50$	44	3656	4050	4024	3869	3869	3869	3869	3869	3840	3840	5.03
$m=10$	45	3629	4109	3836	3761	3761	3761	3720	3712	3706	3706	2.12
	46	3621	3997	3845	3743	3743	3743	3692	3692	3692	3692	1.96
	47	3696	4204	3940	3890	3814	3814	3814	3814	3814	3814	3.19
	48	3572	4113	3986	3867	3718	3718	3718	3718	3709	3709	3.84
	49	3532	3871	3742	3682	3660	3660	3636	3619	3619	3619	2.46
	50	3624	4455	4011	3940	3883	3859	3811	3796	3780	3780^{+}	4.30
Set 6	51	4500	5213	4899	4844	4753	4753	4741	4705	4705	4705	4.56
	52	4276	5163	4960	4811	4720	4700	4700	4668	4658	4658	8.93
	53	4289	5258	4879	4553	4553	4553	4553	4553	4553	4553	6.16
$n=50$	54	4377	5010	4663	4649	4643	4615	4572	4572	4572	4572	4.46
$m=20$	55	4268	5291	4888	4669	4669	4624	4594	4542	4542	4542	6.42
	56	4280	5039	4876	4689	4651	4628	4596	4596	4594	4594	7.34
	57	4308	5110	4853	4636	4636	4573	4505	4505	4505	4505	4.57
	58	4326	5395	4836	4689	4627	4590	4544	4494	4494	4494	3.88
	59	4316	5261	5044	4780	4780	4780	4732	4687	4680	4680	8.43
	60	4428	5160	4887	4831	4719	4719	4663	4663	4639	4639	4.77
Set 7	61	6151	6764	6417	6329	6270	6230	6225	6225	6225	6225	1.20
	62	6022	6537	6236	6113	6113	6108	6108	6034	6034	6034	0.20
	63	5927	6368	6207	5975	5975	5975	5942	5942	5928	5928	0.02
$n=100$	64	5772	6190	5926	5808	5805	5782	5782	5782	5755	***5755***	− ***0.29***
$m=5$	65	5960	6453	6089	6070	6050	6050	6050	6016	5979	5979	0.32
	66	5852	6471	6034	5945	5945	5876	5876	5876	5876	5876	0.41
	67	6004	6471	6220	6111	6081	6056	6056	6050	6046	6046	0.70
	68	5915	6397	6056	6002	5916	5916	5882	5882	5879	***5879***	− ***0.61***
	69	6123	6647	6255	6255	6255	6201	6201	6172	6164	6164	0.67
	70	6159	6741	6274	6274	6244	6180	6154	6154	6154	***6154***	− ***0.08***
Set 8	71	7042	7790	7404	7374	7283	7246	7231	7231	7103	7103	0.87
	72	6791	7547	7097	6957	6895	6866	6814	6814	6814	6814	0.34
	73	6936	7728	7293	7165	7157	7065	7050	7050	7050	7050	1.64
$n=100$	74	7187	7925	7701	7553	7521	7482	7466	7405	7405	7405	3.03
$m=10$	75	6810	7424	7110	7008	6962	6932	6932	6932	6932	6932	1.79
	76	6666	7427	7046	6971	6971	6934	6878	6855	6855	6855	2.84
	77	6801	7681	7322	7117	7117	7071	6983	6983	6983	6983	2.68
	78	6874	7415	7257	6998	6998	6998	6998	6972	6965	6965	1.32
	79	7055	7955	7453	7344	7281	7216	7216	7216	7216	7216	2.28
	80	6965	7705	7344	7225	7129	7129	7125	7123	7058	7058	1.34
Set 9	81	7844	9309	8982	8673	8560	8551	8479	8395	8395	8395	7.02
	82	7894	9234	8540	8380	8309	8248	8232	8232	8232	8232	4.28
	83	7794	9016	8664	8434	8434	8382	8334	8334	8303	8303	6.53
$n=100$	84	7899	8891	8609	8604	8416	8244	8244	8240	8225	8225	4.13
$m=20$	85	7901	9024	8378	8378	8225	8225	8203	8203	8185	8185	3.59
	86	7888	9241	8765	8553	8340	8340	8318	8318	8318	8318	5.45
	87	7930	8936	8620	8457	8457	8364	8364	8364	8241	8241	3.92
	88	8022	9386	8794	8681	8577	8534	8487	8449	8268	8268	3.07
	89	7969	8995	8626	8525	8357	8357	8205	8205	8205	8205	2.96
	90	7993	9275	8886	8771	8655	8655	8564	8432	8432	8432	5.49
Set 10	91	13,406	14,678	14,089	13,674	13,674	13,674	13,557	13,537	13,468	13,468	0.46
	92	13,313	14,472	13,832	13,592	13,537	13,311	13,287	13,287	13,263	***13,263***	− ***0.38***
	93	13,416	14,463	13,944	13,727	13,691	13,615	13,503	13,475	13,475	13,475	0.44
$n=200$	94	13,344	14,641	13,981	13,662	13,644	13,618	13,550	13,435	13,435	13,435	0.68

$m=10$	95	13,360	14,344	13,962	13,694	13,566	13,471	13,426	13,319	13,319	***13,319***	***−0.31***
	96	13,192	14,115	13,754	13,484	13,339	13,304	13,304	13,273	13,169	***13,169***	***−0.17***
	97	13,598	15,270	14,161	13,866	13,805	13,805	13,728	13,728	13,675	13,675	0.57
	98	13,504	14,831	13,909	13,782	13,678	13,629	13,626	13,626	13,607	13,607	0.76
	99	13,310	14,590	13,735	13,511	13,468	13,386	13,386	13,358	13,298	***13,298***	***−0.09***
	100	13,439	14,861	13,914	13,617	13,600	13,535	13,523	13,480	13,456	13,456	0.13
Set 11	101	14,912	16,617	16,006	15,558	15,475	15,331	15,263	15,263	15,235	15,235	2.17
	102	15,002	16,919	15,860	15,721	15,718	15,580	15,501	15,411	15,351	15,351	2.33
	103	15,186	16,484	15,838	15,568	15,409	15,324	15,318	15,318	15,318	15,318	0.87
$n=200$	104	15,082	16,533	15,762	15,515	15,506	15,506	15,430	15,296	15,296	15,296	1.42
$m=20$	105	14,970	17,064	15,823	15,623	15,569	15,351	15,351	15,307	15,307	15,307	2.25
	106	15,101	16,713	15,968	15,781	15,781	15,482	15,482	15,453	15,453	15,453	2.33
	107	15,099	16,746	16,075	15,846	15,840	15,691	15,567	15,567	15,522	15,522	2.80
	108	15,141	16,430	15,897	15,594	15,566	15,534	15,405	15,388	15,358	15,358	1.43
	109	15,034	16,452	15,896	15,540	15,540	15,361	15,361	15,351	15,351	15,351	2.11
	110	15,122	16,385	15,796	15,522	15,464	15,400	15,370	15,370	15,294	15,294	1.14
Set 12	111	36,609	38,636	37,081	36,662	36,576	36,367	36,366	36,135	36,135	***36,135***	***−1.29***
	112	36,927	39,389	37,636	37,253	37,067	36,906	36,906	36,726	36,703	***36,703***	***−0.61***
	113	36,646	38,694	37,426	36,806	36,582	36,473	36,408	36,255	36,174	***36,174***	***−1.29***
$n=500$	114	36,641	38,878	37,508	36,915	36,856	36,673	36,506	36,506	36,416	***36,416***	***−0.61***
$m=20$	115	36,583	39,016	37,536	36,916	36,567	36,567	36,430	36,289	36,123	***36,123***	***−1.26***
	116	36,917	39,207	37,959	37,053	36,985	36,663	36,618	36,575	36,501	***36,501***	***−1.13***
	117	36,518	38,606	37,348	36,658	36,603	36,395	36,256	36,256	36,164	***36,164***	***−0.97***
	118	36,837	38,727	37,642	37,064	36,801	36,561	36,523	36,503	36,415	***36,415***	***−1.15***
	119	36,641	38,761	37,314	36,791	36,426	36,426	36,402	36,235	36,094	***36,094***	***−1.49***
	120	36,866	38,866	37,336	36,772	36,751	36,485	36,485	36,418	36,390	***36,390***	***−1.29***

implementation nor the compiler used threads or any type of parallel code; therefore, the computer can be considered a single 2.93 GHz processor. The 12 sets were solved using eight correlative window widths, from H_1 to H_8 with values 1, 10, 50, 100, 250, 500, 750 and 1000, respectively.

For the initial solution Z_0, we used the value of the solution obtained with the previous width $H_{\alpha-1}$ for each window width H_α ($\alpha=1,\ldots,8$), except for the case with width $H_1=1$ in which Z_0 was fixed at ∞.

To analyze the experimental results, we used the relative percentage deviation (*RPD*) calculated as follows:

$$RPD = \frac{BDP_{Best} - Best_{solution}}{Best_{solution}} \times 100 \quad (28)$$

Table 2 shows:

(1) Column "*Best lit*", shows, for each instance (*Ins*. from 1 to 120) in sets from 1 to 12 from Taillard, the best results reported in the literature (see Ribas et al., 2011).
(2) Column "*Best found*" shows the best solutions found for each instance using Variants 1 and 2. For most instances, Variant 1 offered the best solution and only in exceptional cases (marked with the symbol +), the Variant 2 offered the best solution between them.
(3) Columns headed from $H=1$ to $H=1000$ show the solutions for C_{max} obtained for each window width.
(4) Finally, column "*Best RPD*" shows the best *RPD* value obtained by BDP for each instance.

In Table 2 we can observe that values of *RPD* are between −1.49% (instance 119) and 8.93% (instance 52) in all the instances, considering the best value between the two BDP variants. *RPD* negative values (marked in italics and bold face) indicate an improvement to the best solution reported in the literature; these improvements occur in 17 instances: instance 70 (with a window width $H=500$), instance 95 (with a window width $H=750$) and instances 64, 68, 92, 96, 99, 111, 112, 113, 114, 115, 116, 117, 118, 119 and 120 (with a window width $H=1000$, although some of these instances improved less in previous windows widths).

The average *RPD* for the 120 instances is 2.18%. The average *RPD* for each set (from 1 to 12) and variant (1 and 2) are reported in Table 3. Table 3 also shows average *CPU* time (in s) for both variants of the BDP and windows widths $H=500$, $H=750$ and $H=1000$ (for the 12 sets). The *CPU* times related to $H \leq 250$ have been removed from Table 3 because that can be considered negligible compared to the *CPU* times exposed.

Table 3 shows that the *CPU* times grow if we increase the window width and the dimension of the sets. Regarding to times according to the variant used, in sets from 1 to 4 the times are similar for both variants. However, from sets 5 to 12, the Variant 1 is faster.

In order to study the impact of increase the window width *H* to improve the solutions found, we have performed an additional computational experiment using the window widths $H=1250$, 2500, 5000 and 10,000 in the instances corresponding to the sets from 1 to 4 from Taillard.

The results of this experiment are shown in Tables 4 and 5. The solutions have been improved in 38 instances of 40 (except for instances 2 and 19), comparing the values for *RPD* of column "*Best RPD*" and column "*RPD* $H=1000$", or alternatively, comparing the values for C_{max} of columns "C_{max} $H=1000$" and "*Best found*". The optimal solution is reached in the instances 4, 5, 9 and 10. The average *CPU* times for each window width ($H=1250$ to 10,000) are shown in Table 5, in addition to "*% average RPD*", which improve over the values obtained for $H=1000$, in 0.64%, 0.98%, 1.2% and 0.63% for sets from 1 to 4, respectively.

Table 3
Average *RPD* and *CPU* times (in s) for the 12 sets.

Sets	1	2	3	4	5	6	7	8	9	10	11	12
% Average RPD 1	1.20	2.13	4.42	2.04	3.20	5.95	0.25	1.81	4.65	0.21	1.88	−1.11
% Average RPD 2	2.03	4.30	5.80	3.25	5.24	9.59	3.42	4.22	8.66	4.31	6.61	6.20
% Average RPD (both)	0.99	2.09	4.28	2.04	3.11	5.95	0.25	1.81	4.65	0.21	1.88	−1.11
Average CPU 1/500	3.2	3.6	4.6	36.3	44.2	62.7	191.7	249.0	378.3	1625.2	2630.1	37,046.8
Average CPU 2/500	2.4	3.2	4.4	39.8	46.8	63.9	251.8	299.3	425.2	2156.9	2989.2	43,280.7
Average CPU 1/750	6.9	7.6	9.2	79.1	92.8	128.1	420.3	509.1	719.1	3190.0	4793.4	64,481.6
Average CPU 2/750	4.9	6.3	8.4	83.5	98.6	129.7	545.0	619.7	823.6	4415.5	5554.8	–
Average CPU 1/1000	12.1	12.6	14.8	134.6	159.2	209.4	743.9	888.3	1147.0	5150.9	7389.7	97,577.4
Average CPU 2/1000	8.0	10.6	13.7	146.6	171.3	215.6	935.1	1061.8	1345.5	7219.8	8780.3	–

Table 4
Solutions offered by BDP for the sets from 1 to 4 and window width $H=1250$, 2500, 5000 and 10,000, for each instance.

	Ins.	*Best lit.*	C_{max} $H=1000$	*RPD* $H=1000$	$H=1250$ C_{max}	$H=2500$ C_{max}	$H=5000$ C_{max}	$H=10,000$ C_{max}	*Best found*	*Best RPD*
Set 1	1	1374	1380$^+$	0.44	1379	1379	1379	1379	1379$^+$	0.36
	2	1408	1431$^+$	1.63	1431	1431	1431	1431	1431$^+$	1.63
	3	1280	1302$^+$	1.72	1302	1302	1290	1284	1284$^+$	0.31
$n=20$	4	1448	1456	0.55	1456	1456	1448	1448	1448	0.00
$m=5$	5	1341	1350	0.67	1350	1349	1341	1341	1341	0.00
	6	1363	1385	1.61	1385	1371	1371	1371	1371	0.59
	7	1381	1393	0.87	1393	1393	1391	1386	1386	0.36
	8	1379	1386	0.51	1386	1386	1386	1382	1382	0.22
	9	1373	1389	1.17	1389	1378	1378	1373	1373	0.00
	10	1283	1293	0.78	1293	1283	1283	1283	1283	0.00
Set 2	11	1698	1731	1.94	1730	1730	1730	1712	1712	0.82
	12	1833	1883	2.73	1879	1860	1852	1847	1847	0.76
	13	1659	1683	1.45	1683	1682	1682	1678	1678	1.15
$n=20$	14	1535	1576	2.67	1572	1567	1557	1557	1557	1.43
$m=10$	15	1617	1667	3.09	1667	1667	1664	1652	1652	2.16
	16	1590	1610	1.26	1610	1610	1606	1606	1606	1.01
	17	1622	1681	3.64	1654	1654	1651	1636	1636	0.86
	18	1731	1749$^+$	1.04	1749	1749	1743	1740	1740$^+$	0.52
	19	1747	1755	0.46	1755	1755	1755	1755	1755	0.46
	20	1782	1829	2.64	1829	1829	1817	1817	1817	1.96
Set 3	21	2436	2551	4.72	2551	2537	2537	2537	2537	4.15
	22	2234	2315	3.63	2306	2284	2284	2270	2270	1.61
	23	2479	2627	5.97	2598	2598	2598	2592	2592	4.56
$n=20$	24	2348	2388	1.70	2388	2388	2388	2386	2386	1.62
$m=20$	25	2435	2534	4.07	2534	2515	2508	2486	2486	2.09
	26	2383	2461	3.27	2458	2444	2444	2438	2438	2.31
	27	2390	2497$^+$	4.48	2497	2485	2479	2471	2471$^+$	3.39
	28	2328	2522	8.33	2521	2512	2508	2502	2502	7.47
	29	2363	2483	5.08	2476	2458	2443	2432	2432	2.92
	30	2323	2360	1.59	2360	2360	2338	2338	2338	0.65
Set 4	31	3002	3066	2.13	3066	3064	3063	3044	3044	1.40
	32	3201	3251	1.56	3251	3251	3251	3247	3247	1.44
	33	3011	3077	2.19	3077	3072	3047	3047	3047	1.20
$n=50$	34	3128	3181	1.69	3181	3181	3181	3168	3168	1.28
$m=5$	35	3166	3216	1.58	3216	3212	3205	3204	3204	1.20
	36	3169	3279	3.47	3269	3266	3254	3252	3252	2.62
	37	3013	3090	2.56	3090	3077	3077	3061	3061	1.59
	38	3073	3125	1.69	3125	3122	3098	3098	3098	0.81
	39	2908	2971	2.17	2971	2958	2958	2956	2956	1.65
	40	3120	3163	1.38	3163	3150	3150	3148	3148	0.90

Table 5
Average *RPD* and *CPU* times (in s) for the sets from 1 to 4 and window width $H=1250$, 2500, 5000 and 10,000.

Sets	1	2	3	4
% Average RPD	0.35	1.11	3.08	1.41
Average CPU $H=1250$	17.17	19.92	23.56	231.62
Average CPU $H=2500$	70.24	88.28	100.74	1094.28
Average CPU $H=5000$	274.58	350.21	408.39	4484.20
Average CPU $H=10,000$	1058.99	1395.36	1590.45	19,074.25

7. Conclusions

In this paper a *Bounded Dynamic Programming* (BDP) procedure has been proposed for solving the permutation flow shop problem with blocking ($Fm|block|C_{max}$). This type of procedure has been used to solve sequencing in mixed assembly lines and assembly line balancing problems but, to the best of our knowledge, it has not been used to solve the problem here considered.

The BDP combines features of dynamic programming with features of branch and bound algorithms. The main elements that define the efficiency of the BDP procedure are the graph associated to the problem, the initial solution, the bounding scheme used to prune the graph and the window width used. The window width limits the maximum number of partial solutions retained in each level, therefore it is also necessary to define the rules to decide which vertices are pruned. In our implementation two different variants has been used. The best behavior has been obtained with BDP Variant 1, when the priority rule keeps those vertices with a best bound of C_{max} (i.e. $LB1(t,j)$) and in case of ties those with the best partial C_{max} (i.e. $e_m(t,j)$) of a built subsequence. Even though we have set the initial solution (Z_0) to infinite and we have used a simple bounding scheme, we have improved the best known solutions for 17 of Taillard's benchmark instances. Improved instances are: instance 70 with a window width of 500, instance 95 with a window width of 750 and instances 64, 68, 92, 96, 99, 111, 112, 113, 114, 115, 116, 117, 118, 119 and 120 with a window width of 1000 (although some of these instances improved less in previous windows widths), in a competitive time.

Future research will focus on using an improved bounding scheme more adapted to the characteristics of the problem which, combined with a better initial solution as the MME2 proposed in Ribas et al. (2011), could help to improve the efficiency of the procedure. The procedure will include a dynamic rule, which has the ability to widen or to shrink the window width based on the potential for improvement of the solution.

Acknowledgment

The authors greatly appreciate the collaboration of Nissan Spanish Industrial Operations (NSIO) as well as the Nissan Chair UPC for partially funding this research. This work was also partially funded by project PROTHIUS-III, DPI2010-16759 including EDRF funding from the Spanish government.

References

Abadi, I.N., Hall, N.G., Sriskandarajah, C., 2000. Minimizing cycle time in a blocking flowshop. Oper. Res. 48 (1), 177–180.

Bautista, J., Cano, A., 2011. Solving mixed model sequencing problem in assembly lines with serial workstations with work overload minimisation and interruption rules. Eur. J. Oper. Res. 210 (3), 495–513.

Bautista, J., Pereira, J., 2009. A dynamic programming based heuristic for the assembly line balancing problem. Eur. J. Oper. Res. 194 (3), 787–794.

Bautista, J., Companys, R., Corominas, A., 1996. Heuristics and exact algorithms for solving the Monden problem. Eur. J. Oper. Res. 88, 101–113.

Caraffa, V., Ianes, S., Bagchi, T.P., Sriskandarajah, C., 2001. Minimizing makespan in a blocking flowshop using genetic algorithms. Int. J. Prod. Econ. 70 (2), 101–115.

Carraway, R.L., Schmidt, R.L., 1991. An improved discrete dynamic programming algorithm for allocating resources among interdependent projects. Manage. Sci. 37 (9), 1195–1200.

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

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

Gilmore, P.C., Lawler, E.L., Shmoys, D.B., 1991. Well-solved special cases. In: Lawler, E.L., Lenstra, K.L., Rinooy Kan, A.H.G., Shmoys, D.B. (Eds.), The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, Wiley, pp. 87–143.

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

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discrete Math. 5, 287–326.

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

Lageweg, B.J., Lenstra, J.K., Rinnooy Kan, A.H.G., 1978. A general bounding scheme for the permutation flow-shop problem. Oper. Res. 26 (1), 53–67.

Leisten, R., 1990. Flowshop sequencing problems with limited buffer storage. Int. J. Prod. Res. 28 (11), 2085.

Levner, E.V., 1969. Optimal planning of parts machining on a number of machines. Autom. Remote Control 12 (12), 1972–1978.

Liu, B., Wang, L., Jin, Y., 2008. An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. Comput Oper. Res. 35 (9), 2791–2806.

Marsten, R.E., Morin, Th.L., 1978. A hybrid approach to discrete. Math. Programming 14, 21–40.

McCormick, S.T., Pinedo, M.L., Shenker, S., Wolf, B., 1989. Sequencing in an assembly line with blocking to minimize cycle time. Oper. Res. 37, 925–936.

Morin, Th.L., Marsten, R.E., 1976. Branch-and-bound strategies for dynamic programming. Oper. Res. 24 (4), 611–627.

Nawaz, M., Enscore Jr, E.E., Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11 (1), 91–95.

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

Qian, B., Wang, L., Huang, D.X., Wang, X., 2009. An effective hybrid DE-based algorithm for flow shop scheduling with limited buffers. Int. J. Prod. Res. 47 (1), 1–24.

Reddi, S.S., Ramamoorthy, B., 1972. On the flow-shop sequencing problem with no wait in process. Oper. Res. Q. 23 (3), 323–331.

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

Ronconi, D.P., 2004. A note on constructive heuristics for the flowshop problem with blocking. Int. J. Prod. Econ. 87 (1), 39–48.

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

Ronconi, D.P., Armentano, V.A., 2001. Lower bounding schemes for flowshops with blocking in-process. J. Oper. Res. Soc. 52, 1289–1297.

Suhami, I., Mah, R.S.H., 1981. An implicit enumeration scheme for the flowshop problem with no intermediate storage. Comput. Chem. Eng. 5, 83–91.

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

Wang, L., Zhang, L., Zheng, D., 2006. An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. Comput. Oper. Res. 33 (10), 2960–2971.

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

Please cite this article as: Bautista, J., et al., Solving the $Fm|block|C_{max}$ problem using Bounded Dynamic Programming. Engineering Applications of Artificial Intelligence (2011), doi:10.1016/j.engappai.2011.09.001