J Sched (2010) 13: 597–607
DOI 10.1007/s10951-009-0160-6

Minimizing total tardiness in a stochastic single machine scheduling problem using approximate dynamic programming

Débora P. Ronconi · Warren B. Powell

Published online: 5 February 2010

Abstract This paper addresses the non-preemptive single machine scheduling problem to minimize total tardiness. We are interested in the online version of this problem, where orders arrive at the system at random times. Jobs have to be scheduled without knowledge of what jobs will come afterwards. The processing times and the due dates become known when the order is placed. The order release date occurs only at the beginning of periodic intervals. A customized approximate dynamic programming method is introduced for this problem. The authors also present numerical experiments that assess the reliability of the new approach and show that it performs better than a myopic policy.

Keywords Tardiness · Approximate dynamic programming · Single machine · Scheduling

1 Introduction

The purpose of this paper is to present a method based on approximate dynamic programming to solve the non-preemptive single machine scheduling problem with the objective of minimizing the total tardiness with relation to due dates. It should be observed that the term "single machine" can be interpreted as a production cell or a single bottleneck machine in a complicated machine environment. Furthermore, the results that can be obtained for a single machine provide a basis for heuristics that are applicable to more complicated machine environments.

The optimization criterion considered here is of great importance in manufacturing systems because when a job is not completed by its due date, certain costs are incurred. These costs include: penalty clauses in the contract, if applicable; loss of goodwill resulting in an increased probability of losing the customer for some or all future jobs; and a damaged reputation which will turn other customers away (Sen and Gupta 1984). The difficulty level of this problem can be assessed by means of a particular case whose solution is undoubtedly complex, namely, the deterministic problem when all jobs are available at the same time with one machine is NP-hard for the criterion of minimizing total tardiness (Du and Leung 1990).

This paper addresses the online version of the single machine scheduling problem. According to Pinedo (2002), online scheduling problems are important for several reasons. In practice, often the information is not fully available before the first decision is made. From a theoretical point of view, online scheduling is important because it builds a bridge between deterministic and stochastic scheduling. In this version of the problem, jobs arrive at the system at random times and they have to be scheduled without the knowledge of what jobs will come afterwards. The processing times and the due dates become known when the order is placed.

When there are jobs with nonidentical ready times, as in the problem considered, the insertion of idle time in the schedule can be advantageous; see Kanet and Sridharan (2000) for a literature review about scheduling with inserted

D.P. Ronconi was supported by CNPq (Grants 486124/2007-0 and 307399/2006-0) and FAPESP (Grants 06/03496-3 and 06/53440-4).

D.P. Ronconi (✉)
Departamento de Engenharia de Produção, Escola Politécnica, Universidade de São Paulo, Av. Prof. Almeida Prado, 128, Cidade Universitária, 05508-900, São Paulo, SP, Brazil
e-mail: dronconi@usp.br

W.B. Powell
Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ 08544, USA
e-mail: powell@princeton.edu

Table 1 Processing times, due dates, order arrival times, and release dates of an example with 5 jobs

Job	Processing time	Due date	Order arrival	Release date
1	80	300	20	100
2	70	300	60	100
3	10	210	110	200
4	10	210	140	200
5	10	210	170	200

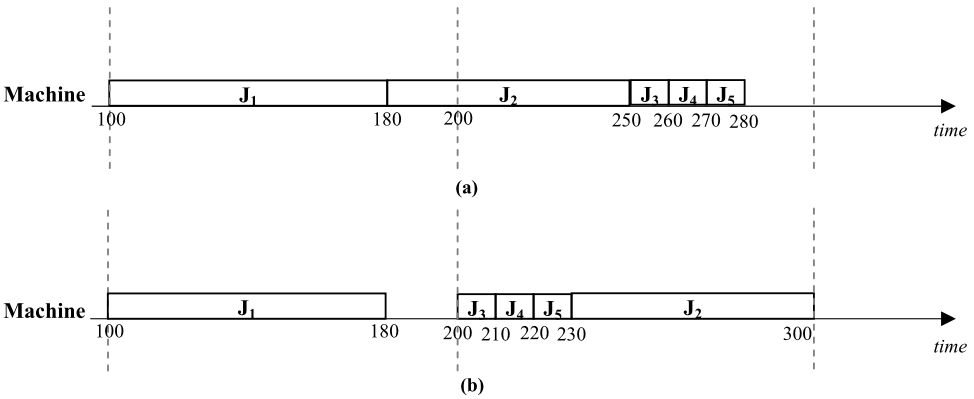

Fig. 1 Schedule generated (**a**) by myopic policy and (**b**) with the insertion of idle time

idle time (IIT). To illustrate this fact, consider the following problem with five jobs. Table 1 shows their order arrival time, release dates, due dates, and processing times. A discrete timing convention (Bergamaschi et al. 1997) under which an order release procedure may occur only at the beginning of periodic intervals is being considered in this example. As can be observed in this example, jobs 1 and 2 have long processing times and loose due dates, while jobs 3, 4, and 5 have short processing times and tight due dates. Moreover, the arrival times of jobs 1 and 2 are such that their release date is defined as 100, while the arrival times of jobs 3, 4 and 5 are such that their release date is defined as 200.

Consider now the application of a myopic policy that defines the schedule considering the jobs available every time new jobs are released. At time 100, jobs 1 and 2 will be available and the myopic policy will choose and fix the subsequence with the smallest total tardiness between subsequences **1 2** and **2 1**. At time 200, jobs 3, 4, and 5 will be available and the myopic policy will choose, among the subsequences including jobs 3, 4, and 5, the subsequence with the best value of the objective function. This subsequence will be fixed after the previous established subsequence (starting at time 250). The final sequence is shown in Fig. 1a. Note that no artificial idle time can be generated by this class of policy.

Finally, let us assume that the scheduler has some information related to the forthcoming orders and that, with this information and in order to obtain a better schedule, forced idle times can be added. Figure 1b shows an alternative schedule that could have been obtained for such a scheduler. In short, knowing that some jobs with tight due dates would arrive, an idle time was inserted to have the machine ready to process the forthcoming jobs.

In both schedules the tardiness of jobs 1 and 2 is zero. On the other hand, the tardiness of jobs 3, 4, and 5 is 0, 10, and 20 units of time in the second schedule, respectively, while in the first schedule the same jobs present the following tardiness values: 50, 60, and 70. These high tardiness values are due to the fact that the fraction (50 units) of the processing time of job 2 that exceeds time 200 generates an additional tardiness of the same value on every tardy job after job 2. As a consequence, the total tardiness of the first solution (generated by the myopic approach) is 180, while the total tardiness of the second solution is 30. Note that a significant improvement in the total tardiness was obtained with a small delay in the completion time of the sequence, illustrating that, in some cases, the insertion of idle time can be significantly advantageous.

A similar statement was made by Sridharan and Zhou (1996). These authors developed a branch-and-bound algorithm and a heuristic with a look-ahead feature for the single machine problem to minimize total tardiness. In the environment considered by those authors, not every job is available at the beginning of the horizon, but it is known a priori when they will arrive. They conclude that, when the shop load is not high, a high percentage of the optimal solutions are schedules with inserted idle time. Kanet and Li (2004) also

mention that future research might use arrival time information to deliberately insert idle time to wait for the arrival of a high priority job.

In this paper we will address the scheduling problem in which jobs arrive at the system at unknown arbitrary times. The insertion of forced idle time will be considered by estimating future states of the system. This problem can be treated as a dynamic assignment problem where one resource (single machine) has to be allocated to a sequence of tasks (jobs). According to Spivey and Powell (2004), the dynamic assignment problem is tremendously rich, posing the challenge of balancing completeness with simplicity. A successful approach to deal with this kind of problem is approximate dynamic programming (ADP), which combines simulation and function approximation to alleviate the "curse of dimensionality" associated with the traditional dynamic programming approach (de Farias and Van Roy 2003; Lee and Lee 2006).

This paper is organized as follows. The next section provides a description of the problem with a mathematical model. Section 3 describes the proposed customized ADP method that is able to consider forthcoming information while minimizing the total tardiness in a finite horizon. In Sect. 4, a comparison with a myopic policy is presented to evaluate the performance of the method in problems up to 200 jobs. Section 5 outlines some of the conclusions and lines for future research.

2 A dynamic scheduling model

In the problem addressed here, there are q jobs to be processed with unequal release dates. Associated with each job j there is an integer processing time, p_j, and an integer due date, d_j. The attributes of each order become known at the order, or arrival, time o_j. Assuming a discrete timing convention (Bergamaschi et al. 1997), the release of orders may occur only at the beginning of periodic intervals. Time is divided into a set of discrete points $\Gamma = 0, 1, .., t, .., T^{\text{horiz}}$ and if the call-in time for job j occurs between instants $t-1$ and t, then it will be available to be served at instant t. Let u be the length of each time interval between two consecutive instants. So, if a time interval is $u = 100$ minutes, time instant $t = 5$ corresponds to 500 minutes from the beginning. The objective is to find a schedule of the jobs that minimizes the total tardiness of the system. Note that jobs that have been released on or before instant t (and which are still available to be scheduled at instant t) may be scheduled at different points in time during the time interval between t and $t+1$, but decisions are made only at the discrete time instances in Γ. No preemption is allowed.

The problem presented above can be described through a dynamic scheduling model. We do this by presenting the state variable, decision variables, exogenous information, transition function, and objective function.

Let S_t be the state of the system at instant t defined as $S_t = \{J_t, A_t, s_t^{\text{start}}\}$, where $J_t = \{\text{set of unserved jobs at instant } t\}$, $A_t = \{a_j = (o_j, p_j, d_j) \mid j \in J_t\}$, and s_t^{start} is the time at which the machine will be available to process jobs from J_t ($s_t^{\text{start}} \geq tu$). Note that the case $s_t^{\text{start}} > tu$ corresponds to a situation where a job started at an earlier time interval is completed after the instant t.

The decision consists of the definition of the subset of jobs in J_t that start to be processed between instants t and $t+1$, henceforth called J_t^C, and the sequence for those jobs represented by the following variables:

$$x_{tjk} = \begin{cases} 1 & \text{if job } j \in J_t \text{ is scheduled as the } k\text{th job in the sequence of jobs that will start to be processed at time } s_t^{\text{start}}; \\ 0 & \text{otherwise,} \end{cases}$$

where for jobs in $J_t \setminus J_t^C$ we have that $x_{tjk} = 0\ \forall k$.

According to a policy $\pi \in \Pi$ (family of policies), at each instant t, the values of the binary decision variables x_{tjk} are defined considering the constraints that establish that each position is occupied by only one job and each job occupies only one position. Let X_t^π be the policy that determines a decision. The completion time C_k of each scheduled job in position k, $k = 1, .., |J_t^C|$, can be calculated as

$$C_k\left(S_t, X_t^\pi\right) = s_t^{\text{start}} + \sum_{b=1}^{k} \sum_{j \in J_t} p_j x_{tjb}.$$

Note that according to the definition of the subset J_t^C an idle time can exist at the end of the time interval. The tardiness T_k of a job $j \in J_t^C$ scheduled as the kth job in the sequence can be computed as

$$T_k\left(S_t, X_t^\pi\right) = \max\left(C_k\left(S_t, X_t^\pi\right) - \sum_{j \in J_t} d_j x_{tjk}, 0\right).$$

The total tardiness of these jobs is given by

$$T_t^{\text{ttard}}\left(S_t, X_t^\pi\right) = \sum_{k=1}^{|J_t^C|} T_k\left(S_t, X_t^\pi\right).$$

The only source of exogenous information is the information about new jobs arriving to the system. Let ω be the sample path that represents a sequence of exogenous information and let $J(\omega)$ be the set that contains all jobs in this path. Assuming that the exogenous information arriving in the time interval between $t-1$ and t is given by $W_t = (\hat{J}_t(\omega), \hat{A}_t(\omega))$, then $(W_1, \ldots, W_{T^{\text{horiz}}})$ is a sample realization of our process, where $\hat{J}_t(\omega) = \{j \mid u(t-1) \leq o_j \leq ut, j \in J(\omega)\}$ is the set of new jobs and $\hat{A}_t(\omega)$ their associated attributes. Following

standard conventions in probability, we assume that $\omega \in \Omega$, where $\mathcal{F}$ is a sigma-algebra on Ω defining the set of events, and $\mathcal{P}$ is a probability measure on $(\Omega, \mathcal{F})$ giving us a standard probability space $(\Omega, \mathcal{F}, \mathcal{P})$.

The dynamics of the system are given by

$$S_{t+1} = S^M\big(S_t, X_t^\pi, W_{t+1}\big)$$

where $S^M(\cdot)$ denotes the transition function and the superscript M stands for "model." The transition function for our problem is given by

$$J_{t+1} = \big(J_t \backslash J_t^C\big) \cup \hat{J}_{t+1},$$
$$A_{t+1} = \big\{a_j = (o_j, p_j, d_j) \mid j \in J_{t+1}\big\} \quad \text{and}$$
$$s_{t+1}^{\text{start}} = \max\big(C_{|J_t^C|}, (t+1)u\big).$$

The goal is to find a policy π that minimizes the expected value of the total tardiness:

$$\min_{\pi \in \Pi} E\left\{\sum_{t=1}^{T^{\text{horiz}}} T_t^{\text{ttard}}\big(S_t, X_t^\pi\big)\right\}.$$

In theory, we can characterize an optimal policy using Bellman's equation:

$$V_t(S_t) = \min_{x_t \in X_t^\pi}\big(T_t^{\text{ttard}}(S_t, x_t) + E\big\{V_{t+1}(S_{t+1}) \mid S_t\big\}\big),$$

where $V_t(S_t)$ is the value of being in state S_t. Here, the expectation is defined with respect to our probability space, but our solution strategy does not require explicitly computing the expectation.

In practice, solving Bellman's equation exactly is computationally infeasible due to the classic curse of dimensionality. For this problem class, there are actually three curses of dimensionality: the state variable, the exogenous information, and the decision vector (see Powell 2007, Chap. 4). Next, we describe how it can be solved using the methods of ADP.

3 Approximate dynamic programming approach

ADP has evolved since the 1950s when Bellman first realized that the practical solution of his optimality equations was limited when the state variable was a vector. The field matured significantly in the 1990s with the appearance of Bertsekas and Tsitsiklis (1996) and Sutton and Barto (1998).

ADP is based on an algorithmic strategy that steps *forward* through time, using iterative algorithms that help to estimate the value function for future stages. In several ADP methods, the value table is constructed only for a small subset of states, and a function is used to interpolate among the stored values. The sampled points are typically determined by running simulations with some known heuristics and/or random actions. The rationale is that, by simulating the conditions one can expect during real operations, one can sample the regions of the state space that are most relevant for constructing well-controlled trajectories (Lee and Lee 2006). In this section, an ADP-based procedure is proposed as a solution method for the scheduling problem described in the previous sections.

Powell (2007) appears to be the first book to recognize all three curses of dimensionality and propose practical solution methods. A major problem occurs when the decision x_t is a vector, requiring the use of classical optimization algorithms. A practical issue is the presence of the expectation in Bellman's equation. This can be solved by using the concept of a post-decision state variable S_t^x (Powell 2007; Powell and Van Roy 2004), which is the state immediately after a decision has been made, but before any new information has arrived.

For this problem, the post-decision state would be given by

$$S_t^x = \big\{J_t^x, A_t^x, s_{t+1}^{\text{start}}\big\},$$
$$J_t^x = \big(J_t \backslash J_t^C\big),$$
$$A_t^x = \big\{a_j = (o_j, p_j, d_j) \mid j \in J_t^x\big\}.$$

The next pre-decision state would then be given as a function of the previous post-decision state using

$$J_{t+1} = J_t^x \cup \hat{J}_{t+1},$$
$$A_{t+1} = A_t^x \cup \big\{a_j = (o_j, p_j, d_j) \mid j \in \hat{J}_{t+1}\big\}.$$

This allows us to break Bellman's equation into two steps:

$$V_t(S_t) = \min_{x_t}\big(T_t^{\text{ttard}}(S_t, x_t) + V_t^x\big(S_t^x\big)\big), \tag{1}$$

$$V_t^x\big(S_t^x\big) = E\big\{V_{t+1}(S_{t+1}) \mid S_t^x\big\}, \tag{2}$$

where $V_t^x(S_t^x)$ is the value of being in state S_t^x immediately after we made a decision.

At the heart of this strategy is that the decision is made with (1), which is a deterministic optimization problem. Furthermore, it is typically a fairly small deterministic optimization problem, because it only works with the jobs that are known at time t, plus the value function. This is how we handle the fact that x is a vector (the third curse of dimensionality).

In general (and certainly for this problem), the value function, and the expectation in (2), cannot be computed exactly. For this reason, we replace it with a statistical approximation obtained by the proposed algorithm through a training phase executed for N iterations with a sequence of sample paths. This statistical approximation is represented

as $\bar{V}_t^{n-1}(S_t^x \mid \bar{\theta}_t^{n-1})$ where $\bar{\theta}_t^{n-1}$ is a vector of parameters estimated after $n-1$ iterations of our algorithm. At each iteration a sample path from past history or from a known probability model should be available.

This allows us to make a decision by solving

$$\hat{v}_t^n = \min_{x_t \in \chi_t^n} \left(T_t^{\text{ttard}}\left(S_t^n, x_t\right) + \bar{V}_t^{n-1}\left(S_t^x \mid \bar{\theta}_t^{n-1}\right)\right).$$

We let x_t^n be the value of x_t that solves the minimization problem at iteration n. The key is to choose a functional approximation for the value function that works with a search algorithm that is appropriate to the problem. Finally, we use $\hat{v}_t^n$ to update $\bar{V}_{t-1}^{n-1}$, which is the value function at the previous instant $(t-1)$, around the previous post-decision variable. This is how we approximate the expectation in (2). The precise updating mechanism U^V depends on the value function approximation, so we may represent the statistical updating using

$$\bar{V}_{t-1}^n \leftarrow U^V\left(\bar{V}_{t-1}^{n-1}, S_{t-1}^{x,n}, \hat{v}_t^n\right).$$

In this paper, we assume that the value function has a linear form with unknown parameters θ, and we update our estimate of the vector θ using recursive statistics. The overall strategy works as follows. At each instant t we solve a static scheduling problem that consists of the currently available jobs with their known processing times and due dates. The currently available jobs can be jobs which first became known to the system between instants $t-1$ and t, or they can be jobs that were not started before instant t. Several different schedules are generated, and the one with the smallest estimate of the total tardiness at the end of the planning horizon is chosen. In that selected schedule, each job that started to be processed in the time interval between t and $t+1$ will be fixed, and its contribution to the total tardiness can be exactly evaluated. However, the tardiness that is caused by the known jobs that will be processed after instant $t+1$ and by the future jobs must be estimated. This estimate is made with the help of a parameterized value function, $\bar{V}_t$, which is updated iteratively.

At instant $t+1$, new jobs will arrive, and the set of available jobs will be updated. A schedule for the time interval between $t+1$ and $t+2$ will be selected, and the actual tardiness for jobs scheduled in $(t+1, t+2)$ can be calculated. With this information, the parameters of the value function applied at instant t can be updated. The updated value function will be used in the next iteration of the algorithm. For each instant t a different value function is established.

The algorithm is trained by running it for N iterations. This parameter is empirically defined based on previous experiments. We assume that, at each iteration, data of the exogenous information of the problem is available, and arbitrary initial values for the parameters are assigned. Our solution approach requires only that we have the ability to generate data in an appropriate way that reproduces the behavior of the system. At *Step 1* of the algorithm, we can generate sample paths from past history or from an explicit probability model (if it is known). When the function is finally established (at the end of iteration N), the method can be applied to the actual current data from the problem that we want to solve. Therefore, Algorithm 3.1 below runs $N+1$ times, neglecting the steps associated with the establishment of $\bar{V}_t$ in its last execution. The algorithmic description of the ADP approach is shown below.

Algorithm 3.1: Approximate dynamic algorithm approach

Step 0. Initialization:

Step 0a. Initialize $\bar{\theta}_t^0 = 0\ \forall t \in \Gamma$.

Step 0b. Initialize $n = 1$ and S_0^1.

Step 1. Choose a sample path ω^n.

Step 2. For $t = 0, 1, \ldots, T^{\text{horiz}}$:

Step 2a. Solve (perhaps heuristically):

$$\hat{v}_t^n = \min_{x_t \in \chi_t^n} \left(T_t^{\text{ttard}}\left(S_t^n, x_t\right) + \bar{V}_t^{n-1}\left(S_t^x \mid \bar{\theta}_t^{n-1}\right)\right) \tag{3}$$

and let x_t^n be the value of x_t that solves (3).

Step 2b. If $t > 0$, update the vector of parameters θ_{t-1}^n and consequently the value function approximation using:

$$\bar{V}_{t-1}^n \leftarrow U^V\left(\bar{V}_{t-1}^{n-1}, S_{t-1}^{x,n}, \hat{v}_t^n\right).$$

Step 2c. Update the state of the system:

$$S_{t+1}^n = S^M\left(S_t^n, X_t^\pi, W_{t+1}(\omega^n)\right).$$

Step 3. Increment n by one. If $n \leq N$ go to Step 1.

Step 4. Return the value functions $\bar{V}_t^N$ for $t = 0, 1, 2, \ldots, T^{\text{horiz}}$.

For our problem the selection of a sequence of exogenous information (ω^n) at *Step 1* consists of sampling the release dates, processing times, and due dates of the jobs. The implementation of this step can be made at the beginning of each iteration or inside the loop of t (*Step 2*). The initial state of the system is empty, i.e., $S_0^1 = \emptyset$.

We note that with this strategy, we do not, at instant t, plan decisions for future time intervals. The impact of decisions now on the future is captured (approximately) through the value function, which is estimated from prior sample paths. This also means that the problem at instant t is relatively small, which means it is easier to solve. We are able to quickly solve a sequence of small problems, following a sample path of random events and using information from solving these problems in the future to update the value function.

At *Step 2a* the set χ_t^n of feasible schedules at instant t and iteration n is the set of all possible schedules considering all

the available jobs at instant $t(J_t)$. In order to obtain good solutions within a reasonable time, the problem stated at this step is solved by heuristic methods. Initially we apply a dispatching rule that provides an initial solution considering all available jobs. This sequence is used as the seed sequence for a local search. For each sequence obtained by the local search we evaluate the total tardiness of the jobs which are started between instants t and $t+1(T_t^{\text{ttard}})$ and the estimate of the tardiness that is caused by all jobs (known and unknown) that will be processed after instant $t+1(\bar{V}_t^n)$. The jobs which are started between instants t and $t+1$ in the best solution found are fixed.

Some dispatching rules were tested to generate the initial solution. The best results were obtained through the use of the modified due date (MDD) rule proposed by Baker and Bertrand (1982). The MDD rule is known to be an efficient heuristic that can minimize total tardiness in the single machine environment (Sen et al. 2003). At each time τ^{mdd} for each job j, this rule evaluates the following measure:

$$\max\bigl(d_j, C_j^{\text{midd}}(D)\bigr)$$

where D represents the job sequence already established at time $\tau^{\text{mdd}} = s_t^{\text{start}} \sum_{i\in D} P_i$ and $C_j^{\text{mdd}}(D)$ represents the completion time of job j, $j \notin D$, if this job is sequenced just after sequence D. The job with the smallest measure is chosen to be sequenced at time τ^{mdd}. This procedure is repeated until all available jobs at instant $t(J_t)$ belong to D.

The local search requires the definition of its components, namely, the type of move that generates the neighborhood and the search strategy. In this algorithm, the neighborhood is defined by the shift move, which consists of removing a job from its original position and inserting it in the remaining positions. This move generates a neighborhood of size $(|J_t|-1)^2$. In addition, for each sequence obtained by the local search, the insertion of idle time is considered. This idle time is inserted by moving the last job that starts before the instant $t+1$ until it starts at instant $t+1$. With this additional consideration, the new size of a neighborhood is $2(|J_t|-1)^2$. The search technique applied is the *best improving* strategy, where the entire neighborhood is analyzed and the current solution is replaced by the best one.

For each different sequence obtained by the local search, the values of T_t^{ttard} and $\bar{V}_t^n$ in (3) must be evaluated. T_t^{ttard} is computed as

$$T_t^{\text{ttard}}\bigl(S_t^n, x_t\bigr) = \sum_{k=1}^{|J_t^C|} T_k\bigl(S_t^n, x_t\bigr).$$

We then need to evaluate the impact of the decision x_t, which we do using a value function approximation around the state of the system immediately after a decision is made (the post-decision state). This step requires that we design a functional approximation to provide this estimate. A classical strategy is to use a series of *basis functions* $\phi_f(S_t^x)$, $f \in \mathcal{F}$, where $\mathcal{F}$ is sometimes referred to as a set of features. A generic linear-in-the-parameters value function would be written:

$$V_t^n\bigl(S_t^x|\theta_t^n\bigr) = \sum_{f\in F} \theta_f \phi_f\bigl(S_t^x\bigr).$$

The art of ADP is designing the basis functions. We experimented with a number of alternatives. The best results were obtained through the use of the following linear function of the tardiness and the fraction of the processing time of the job in position $|J_t^C|$ that exceeds the instant $t+1$, p^{excess}:

$$\bar{V}_t^n\bigl(S_t^x|\bar{\theta}_t^n\bigr) = \bar{\theta}_{t0}^n + \bar{\theta}_{t1}^n\left(\sum_{k=|J_t^C|+1}^{|J_t|} T_k\right) + \bar{\theta}_{t2}^n p^{\text{excess}}.$$

Observe that, when idle time is inserted, $p^{\text{excess}} = 0$. We note that this approximation should work well for scheduling problems in this same class, but different approximations may be needed for different problem classes. For example, a parallel machine scheduling problem with setups might benefit from a feature that captures the number of different job classes in the post-decision state (which hints at the number of machine setups that might be required).

The problem of finding a good schedule requires choosing a candidate solution x_t, computing the tardiness $T_t^{\text{ttard}}(S_t^n, x_t)$, and then computing $\bar{V}_t^n(S_t^x|\bar{\theta}_t^{n-1})$ where S_t^x captures the jobs that were not covered in the current time interval. The important feature of ADP is that $\bar{V}_t^n(S_t^x|\bar{\theta}_t^{n-1})$ is very simple to compute, and as a result is easy to embed within a local search heuristic.

In order to update the value function approximation (or, in our implementation, the vector of parameters $\bar{\theta}_{t-1}^n$) at *Step 2b*, the following problem is solved:

$$\min_{\theta} \sum_{m=1}^{n} \lambda^{n-m}\bigl(\hat{v}_t^m - \bar{V}_{t-1}^{m-1}\bigl(\bar{\theta}_{t-1}^{m-1}, S_{t-1}^x\bigr)\bigr)^2, \tag{4}$$

where $\lambda \in (0,1]$ is a discount factor that is used to assign lower weight to older observations. At each iteration a new observation, i.e., a new value of $\hat{v}_t^n$, is obtained and a new solution of (4) should be found.

If $\lambda = 1$, we face a classical least-squares problem. Let Φ^n be a matrix where there are n rows (corresponding to the n observations) where the ith row corresponds to $(\phi_0(S_t^{x,i}), \phi_1(S_t^{x,i}), \phi_2(S_t^{x,i}))$ where $\phi_0(S_t^{x,i}) = 1$, $\phi_1(S_t^{x,i}) = \sum_{k=|J_t^C|+1}^{|J_t|} T_k$ and $\phi_2(S_t^{x,i}) = p^{\text{excess}}$. Classical statistical theory tells us that the optimal parameter vector is given by

$$\theta^n = \bigl[\Phi^n\bigl(\Phi^n\bigr)^T\bigr]^{-1}\Phi^n\hat{v}^n,$$

where $\hat{v}^n$ is a column vector of all the observations of the value of being in the observed states. The problem is that this equation is clumsy to compute at each iteration. Instead, we use recursive least squares (see Powell 2007, and Bertsekas and Tsitsiklis 1996). Let ϕ^n be a column vector of the basis functions $(\phi_0(S_t^{x,i}), \phi_1(S_t^{x,i}), \phi_2(S_t^{x,i}))$, and let $B^n = [\Phi^n(\Phi^n)^T]^{-1}$. B^n and θ^n can be computed recursively. Start with $B_t^0 = I\ \forall t$, and some initial value for θ^0 (say zero). The steps for recursive least squares require executing:

$$\text{(i)} \quad \gamma^n = \lambda + \left(\phi^n\right)^T B_t^{n-1}\phi^n,$$

$$\text{(ii)} \quad H^n = \frac{1}{\gamma^n} B_t^{n-1},$$

$$\text{(iii)} \quad \varepsilon^n = \hat{v}_t^n - \bar{V}_{t-1}^{n-1}\left(S_{t-1}^x|\bar{\theta}_{t-1}^{n-1}\right),$$

$$\text{(iv)} \quad \bar{\theta}_{t-1}^n = \bar{\theta}_{t-1}^{n-1} + H^n\phi^n\varepsilon^n,$$

$$\text{(v)} \quad B_t^n = \frac{1}{\lambda}\left(B_t^{n-1} - \frac{1}{\gamma^n}\left(B_t^{n-1}\phi^n\left(\phi^n\right)^T B_t^{n-1}\right)\right),$$

where H and B are 3×3 matrices and γ and ε are scalar.

The matrix B_t^{n-1} captures $[\Phi^n(\Phi^n)^T]^{-1}$ in the normal equations for linear regression. λ plays the role of a discount factor. If we use $\lambda = 1$, then we are putting equal weight on all the previous errors. Because the observations $\hat{v}_t^n$ come from a nonstationary series, it tends to be better to use $\lambda < 1$, which is the same as putting a discount of λ^{n-m} on observations made m iterations ago. The updating equation for B_t^n (equation v), is derived from the well-known Sherman-Morrison formulas for updating the inverse of a matrix (see Sect. 7.6 of Powell 2007).

According to Powell (2007) (Sect. 7.3.3), a relationship between λ and the convergence of the method can be heuristically established. By defining, for each iteration, α_n as a step size and λ_n as the discount factor, this relationship can be described as follows:

$$\lambda_n = \begin{cases} 1 & \text{if } n = 1, \\ \alpha_{n-1}\left(\frac{1-\alpha_n}{\alpha}{}_n\right) & \text{if } n > 1. \end{cases} \tag{5}$$

The value of the adaptive step size was adjusted according to George and Powell (2006) as

$$\alpha_n = 1 - \frac{(\sigma^n)^2}{\delta^n}, \tag{6}$$

where σ^2 is the variance in the observations and δ is the expected value of the squared prediction errors. In order to estimate σ^2 and δ in each iteration n, these authors presented approximate formulas and computed them recursively. The detailed formulas for this calculation can be found in the Appendix.

4 Numerical experiments

Aiming to evaluate the performance of the proposed ADP approach, we implemented Algorithm 3.1 and performed a comparison with its myopic counterpart on a set of 750 problems. Codes were written in the C programming language and tests were run on a Pentium IV with a 2.66 GHz processor and 3.25 GB RAM.

As with steady state problems, a myopic policy can work quite well; the ADP approach developed in this paper applies to the case in which jobs have heterogeneous characteristics (as illustrated in the example of Fig. 1) and nonuniform arrivals. Given this aspect of the problem, the numerical experiments were conducted as follows. For each problem, half of the jobs have short processing times and tight due dates (type 1), while the other half have long processing times with loose due dates (type 2). The data generation is done at each instant t. For type 1 jobs, we consider a processing time p_j in the range [1,10] (discrete uniform distribution) and a due date d_j defined as $d_j = tu + gp_j$ with $g \in [1, 2]$ (discrete uniform distribution). We also consider five possibilities for type 2 jobs, setting $p_j \in [10, 40]$ and $d_j = tu + gp_j$ with $g \in [10, G]$ and $G \in \{60, 70, \ldots, 100\}$. The increased arbitrariness of the due dates creates more opportunities to distinguish between jobs, thus increasing the need to hold the machine idle for accommodating a soon-to-arrive hot job (Sridharan and Zhou 1996; Kanet 1986). Problems were generated varying the average number of jobs $q \in \{100, 125, 150, 175, 200\}$.

We assume that the time horizon is divided into $T^{\text{horiz}} = 70$ time intervals and the length of each interval is $u = 100$ units of time. The jobs arrive to the system in the following way. The number of arrivals at each time interval between instants $t-1$ and t follows a Poisson distribution with parameter $\lambda_t^{\text{arrival}}$. The total number of jobs that arrive within the horizon is also a Poisson distribution with parameter q. Note that we have a different parameter $\lambda_t^{\text{arrival}}$ for each interval. The values of these parameters were selected in order to have busy intervals in the middle of the horizon. The idea is to simulate situations where seasonal surge may occur (e.g., candy canes in Christmas season, ice cream in summer or, in small horizons, a mid-week rush). For the reproducibility of our results, parameters $\lambda_t^{\text{arrival}}$ are available at http://www.poli.usp.br/pro/docentes/dronconi/.

In the training phase, for each combination of q and G (these sets will be called *classes* from now on), the algorithm was run N times. After running a few problems with several values, N was set equal to 500. Once the adjustments of the parameters of the value function were made, a set of 30 instances was generated for each class. So, overall in this first set of experiments, the ADP approach was applied to 25 different classes with a total of 750 problems.

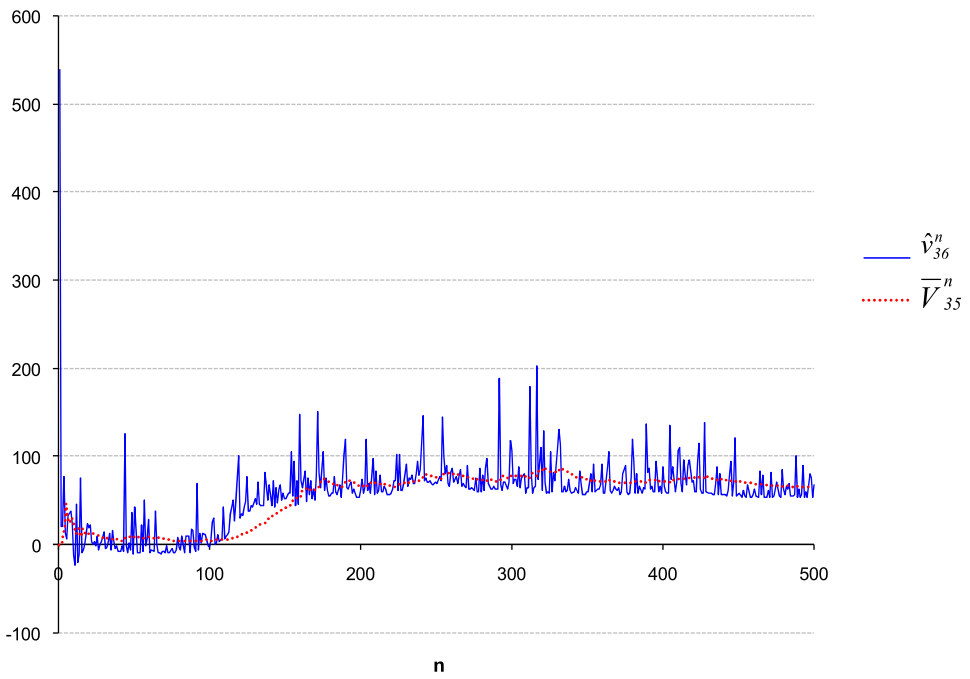

Fig. 2 ADP algorithm: training phase

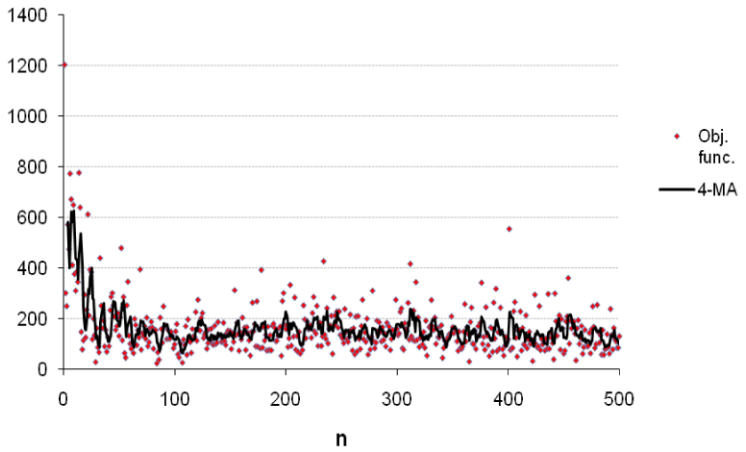

Fig. 3 Objective function value during the training phase

Initially, to illustrate how the ADP algorithm works during the training phase, Fig. 2 shows the values of $\bar{V}^n_{35}$ (contribution to the objective function of the decision taken at instant 35 from time s^{start}_{36} to the end of the planning horizon) and $\hat{v}^n_{36}$ (contribution to the objective function of the decision taken at instant 36 from s^{start}_{36} to the end of the planning horizon) in a problem with 100 jobs. After the initial phase ($n > 100$), the parameters of the value function converge to stable values and $\bar{V}^n_{35}$ is able to estimate $\hat{v}^n_{36}$, as expected.

Figure 3 shows the value of the objective function during the training phase. It can be noted that this value decreases in the beginning of this phase, as shown by the 4-iteration moving average curve. This behavior indicates that the method learns quickly how to improve the quality of the schedules that were created.

In order to evaluate the quality of the solutions generated by the proposed method, we compare it against its myopic counterpart. At each instant t the myopic approach initially applies the MDD rule to generate an initial solution considering all available jobs (J_t). Next, this sequence is used as the seed sequence for a local search with the same components as ADP. However, for each sequence obtained only the total tardiness of all available jobs was evaluated. Note that no estimate about the future is made and that the algorithm will choose, at the beginning of each time interval, the best sequence based on the total tardiness minimization of all available jobs. As a consequence, no idle times will be inserted in the schedule, since that will lead to higher total tardiness values. At the end of the search, the jobs which are started between instants t and $t + 1$ in the best solution found are fixed.

Table 2 Percentage differences of the ADP and the myopic policy

Mean number of jobs	G									
	100		90		80		70		60	
	%Dif	σ	*%Dif*	σ	*%Dif*	σ	*%Dif*	σ	*%Dif*	σ
100	**63.3**	4.0	**63.4**	4.0	**63.4**	4.1	**63.4**	4.0	**63.9**	4.0
125	**68.5**	2.3	**68.7**	2.3	**67.9**	2.3	**68.3**	2.3	**69.0**	2.4
150	**68.0**	1.2	**65.8**	1.2	**67.8**	1.1	**67.7**	1.0	**65.6**	2.1
175	**66.4**	1.2	**66.5**	0.9	**66.0**	0.9	**36.1**	6.5	14.3	17.4
200	**58.6**	1.4	**56.1**	1.6	28.0	7.5	−17.7	16.0	−42.4	12.8

Table 3 Average CPU time (ms)

Mean number of jobs	G									
	100		90		80		70		60	
	ADP	mp	ADP	mp	ADP	mp	ADP	mp	ADP	mp
100	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.0	1.0	0.5
125	3.7	1.0	3.7	0.5	3.1	0.5	3.1	1.1	3.6	0.5
150	8.8	2.1	8.8	2.1	9.4	2.1	9.4	1.6	9.4	2.1
175	22.4	4.7	23.4	4.7	22.9	4.7	21.9	4.7	21.9	4.7
200	48.9	10.9	48.4	10.9	47.9	11.5	48.4	10.9	47.9	13.0

The percentage difference of the solution value obtained by ADP (F_a) when compared to the value of the myopic policy (F_{mp}) is calculated as follows:

$$\%Dif = 100\frac{F_{mp} - F_a}{F_{mp}}.$$

Table 2 presents the average value of *%Dif* and the standard deviation (σ) obtained for each class consisting of 30 problems. It can be noted that significant improvements were achieved for almost every class. The numbers in bold indicate classes where the ADP strategy outperforms the myopic policy by over 35%. The reason for the good performance of ADP when different jobs are mixed can be attributed to the fact that the myopic policy cannot learn from the past. Thus, this kind of policy cannot realize that in the case of jobs with longer processing times and loose due dates, the insertion of forced idle times can be very advantageous. Conversely, the ADP function can estimate the future state of the system and help the algorithm to choose actions that minimize the total tardiness of the whole system; these actions can include the insertion of idle time or not. In this experiment almost all solutions obtained through ADP presented forced idle times. On average, for a given problem, this inclusion occurred in approximately 28% of its intervals.

In a few classes, which correspond to scenarios with little flexibility for the insertion of idle time ($n = 200$ and $G \leq 70$), the ADP approach produces worse results. This is probably due to the fact that, although ADP considers that some large jobs must be postponed to process arriving jobs with tight due dates, their due dates may not be loose enough to avoid their tardiness. Moreover, when a job is tardy, the tardiness is composed of the processing time of all jobs scheduled between its due date and its completion time. Consequently, if larger jobs are processed in this period of time, tardiness can be high.

Table 3 presents the average CPU time of each class consisting of 30 problems with the application of the ADP approach and the myopic policy (mp). Note that the myopic policy is approximately 2–4 times faster than the ADP; however, much better solutions can be found by ADP, as presented in Table 2. This result was expected since in the proposed ADP approach a larger neighborhood is considered.

In order to gain more insight into the performance of the suggested procedure, an additional analysis is presented in Table 4. The number of successful cases is used to evaluate ADP by comparing it with the myopic policy for the 750 problems tested. We say that a method has successfully solved a problem when it is able to find a better solution value than its competitor. There were just five tie cases. As can be observed, ADP presents better results in 698 problems. Furthermore, as the due dates of the larger jobs become tight and the utilization of the system increases, the number of successful cases is reduced.

With the purpose of showing that the ADP algorithm can also have a good performance with problems that have ho-

Table 4 Number of successful cases of ADP

Mean number of jobs	G				
	100	90	80	70	60
100	29	29	29	29	29
125	30	30	30	30	30
150	30	30	30	30	30
175	30	30	30	29	24
200	30	30	25	16	9

Table 5 Comparison of ADP and the myopic policy in a homogeneous set of problems

Mean number of jobs	*%Dif*	σ	Number of successful cases of ADP	Even
100	0.0	0.0	0	30
125	0.0	0.0	0	30
150	0.0	0.1	1	28
175	0.0	0.1	1	27
200	−0.4	0.2	3	20

mogeneous characteristics, a second set of instances is considered. In this experiment all jobs are of type 1.

Table 5 shows the average value of *%Dif* and the standard deviation (σ) obtained for each class. The number of successful cases of ADP and the number of even cases are also presented. Note that, in 135 of 150 problems, both algorithms found the same solution value and that, in the other problems, the percentage difference is very small. It can be observed that, when the jobs have homogeneous characteristics and most of the available jobs can be processed within the present interval, the schedules generated by the ADP approach have the same quality as the ones generated by the myopic approach. In this experiment only 8.7% of the solutions obtained by ADP presented forced idle times. This inclusion occurred only in one or two intervals per solution. The running times of both analyzed strategies did not present significant differences and in the largest case, with 200 jobs, the CPU times were shorter than 1.5 ms.

5 Conclusions and future research

This research addressed a scheduling problem where jobs arrive at random times in a single machine environment. A method based on approximate dynamic programming that explores information in advance, especially by inserting idle time, was presented to solve this problem. A comparison with a myopic policy was carried out with a set of nonhomogeneous jobs, and the best performance was obtained by the proposed approach. This good performance can be attributed to the fact that the myopic policy is not able to realize that, in future stages of the system, it could be valuable to insert forced idle time or a different sequence of jobs that might not be locally advantageous. ADP was also evaluated with a homogeneous set of jobs, and this strategy showed the same behavior of the myopic policy. These computational experiments indicate that the classes of problems for which ADP is most well suited are precisely those where the jobs have heterogeneous characteristics.

As future research, the performance of the proposed method can be evaluated in more generic environments as, for example, in flowshop scheduling problems. Also, further research should focus on developing an efficient stopping criterion for the training phase.

Acknowledgements The authors would like to thank the anonymous associate editor and the referee whose comments greatly helped to improve this paper. This research has been partially supported by "Fundação de Amparo à Pesquisa do Estado de São Paulo"—FAPESP and by "Conselho Nacional de Desenvolvimento Científico e Tecnológico"—CNPq. The work of the second author was supported by the U.S. Air Force Office of Scientific Research, grant FA9550-08-1-0195.

Appendix: Approximate formulas for σ^2 and δ (George and Powell 2006)

In order to estimate σ^2 and δ to calculate α_n in (6), George and Powell (2006) made an approximation of δ^n by smoothing the squared instantaneous errors. Here are the steps implemented in this paper to recursively compute the approximations $\bar{\delta}^n$ and $\bar{\sigma}^n$ at each iteration.

Step 0. Assign an initial step size $\alpha_1 = 1$ and initial and target values for error step size $\eta^0 = 1$ and $\bar{\eta} = 0$. Assign initial values to the parameters $\bar{\beta}^0 = 0$, $\bar{\lambda}_0^{gp} = 1$ and $\bar{\delta}^0 = 0$.

Step 1. With the new observation $\hat{v}_t^n$ and the related error ε, evaluate the following expressions:

$$\eta^n = \frac{\eta^{n-1}}{1 + \eta^{n-1} - \bar{\eta}},$$

$$\bar{\delta}^n = \left(1 - \eta^n\right)\bar{\delta}^{n-1} + \eta^n \varepsilon^2,$$

$$\bar{\beta}^n = \left(1 - \eta^n\right)\bar{\beta}^{n-1} - \eta^n \varepsilon,$$

$$\left(\bar{\sigma}^n\right)^2 = \frac{\bar{\delta}^{n-1} - (\bar{\beta}^n)^2}{1 + \bar{\lambda}_{n-1}^{gp}}.$$

Step 2. Evaluate the step size for the next iteration (if $n > 1$):

$$\alpha_n = 1 - \frac{(\bar{\sigma}^n)^2}{\bar{\delta}^n}.$$

Step 3. Update the coefficient for the variance of the smoothed estimate:

$$\bar{\lambda}_n^{gp} = \begin{cases} (\alpha_n)^2, & \text{if } n = 1, \\ (1-\alpha_n)^2 \bar{\lambda}_{n-1}^{gp} + (\alpha_n)^2, & \text{if } n > 1. \end{cases}$$

An advantage of the proposed methodology is the small number of parameters. To apply the proposed formulas, the only tuned parameter is the target error step size. In this paper, after running the generated problems with several values, the target error step size was set to zero. Furthermore, these steps can be easily implemented. For additional details, see George and Powell (2006).

References

Baker, K. R., & Bertrand, J. W. M. (1982). A dynamic priority rule for scheduling against due-dates. *Journal of Operations Management*, *3*, 37–42.

Bergamaschi, D., Cigolini, R., Perona, M., & Portioli, A. (1997). Order review and release strategies in a job shop environment: a review and a classification. *International Journal of Production Research*, *35*, 399–420.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont: Athena Scientific.

Du, J., & Leung, J. Y.-T. (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, *15*, 483–495.

de Farias, D. P., & Van Roy, B. (2003). The linear programming approach to approximate dynamic programming. *Operations Research*, *51*, 850–865.

George, A., & Powell, W. B. (2006). Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine Learning*, *65*, 167–198.

Kanet, J. J. (1986). Tactically delayed versus non-delay scheduling: an experimental investigation. *European Journal of Operational Research*, *24*, 99–115.

Kanet, J. J., & Li, X. (2004). A weighted modified due date rule for sequencing to minimize weighted tardiness. *Journal of Scheduling*, *7*, 261–276.

Kanet, J. J., & Sridharan, V. (2000). Scheduling with inserted idle time: problem taxonomy and literature review. *Operations Research*, *48*, 99–110.

Lee, J. H., & Lee, J. M. (2006). Approximate dynamic programming based approach to process control and scheduling. *Computer and Chemical Engineering*, *30*, 1603–1618.

Powell, W. B., & Van Roy, B. (2004). Approximate dynamic programming for high dimensional resource allocation problems. In J. Si, A. G. Barto, & W. B. Powell II (Eds.), *Handbook of learning and approximate dynamic programming*. New York: IEEE Press.

Powell, W. B. (2007). *Approximate dynamic programming: solving the curses of dimensionality*. Hoboken: Wiley.

Pinedo, M. (2002). *Scheduling: theory, algorithms, and systems*. Upper Saddle River: Prentice-Hall.

Sen, T., & Gupta, S. K. (1984). State-of-art survey of static scheduling research involving due dates. *OMEGA*, *12*, 63–76.

Sen, T., Sulek, J. M., & Dileepan, P. (2003). Static scheduling research to minimize weighted and unweighted tardiness: a state-of-the-art survey. *International Journal of Production Economics*, *83*, 1–12.

Sridharan, V., & Zhou, Z. (1996). Dynamic non-preemptive single machine scheduling. *Computers and Operations Research*, *23*, 1183–1190.

Spivey, M. Z., & Powell, W. B. (2004). The dynamic assignment problem. *Transportation Science*, *38*, 339–419.

Sutton, R., & Barto, A. (1998). *Reinforcement learning*. Cambridge: MIT Press.