





Total productive maintenance of make-to-stock production-inventory systems via artificial-intelligence-based iSMART

Angelo Encapera^a, Abhijit Gosavi ^b and Susan L. Murray ^b

^aGarmin International, Kansas City, MO, USA; ^bDepartment of Psychological Science, Missouri University of Science and Technology, Rolla, MO, USA

ABSTRACT

Total Productive Maintenance (TPM) is a critical activity that significantly reduces lead times and uncertainty in Make-To-Stock (MTS) production systems, thereby increasing the efficiency and profit margins of the associated firm. TPM problems can be set up as semi-Markov decision processes (SMDPs) and solved optimally using classical dynamic programming (DP) on small-scale problems. However, on large industrial-scale problems, DP breaks down, and one must then resort to an artificial intelligence (AI) technique called reinforcement learning (RL). This work presents a new AI algorithm for solving SMDPs, called iSMART, an acronym for imaging Semi-Markov Average Reward Technique. iSMART requires a significantly lower modelling and computational effort than classical DP, where estimating the transition probabilities can itself be very time-consuming and mathematically challenging for large-scale problems. Further, unlike previous RL algorithms for SMDPs, iSMART does *not* need exploration decay. This means iSMART eliminates an additional parameter that requires significant tuning in the traditional RL-based solution approach. Modern AI based in deep learning seeks to reduce dependence on tuning parameters. iSMART is designed in this spirit for solving TPM problems in MTS production systems, where it is shown to deliver optimal solutions on small-scale problems and near-optimal ones on large-scale problems.

ARTICLE HISTORY

Received 3 April 2019
Accepted 16 December 2019

KEYWORDS

Total productive maintenance; make-to-stock systems; reinforcement learning; constant exploration

1. Introduction

Total Production Maintenance (TPM) originated from the fields of reliability and maintenance (Jardine & Tsang, 2005; Lewis, 1994). These closely related fields have together created many innovative engineering practices resulting in savings worth millions of dollars in industry by reducing downtime, lead times, and work-in-process inventory and increasing profits (Ahmed, Hj. Hassan, & Taha, 2005; Ireland & Dale, 2001; Manzini, Regattieri, Pham, & Ferrari, 2009; McCall, 1965; McKone and Weiss, 1998). TPM takes a systematic look at production systems in order to incorporate scheduled maintenance into standard industry practices. In Make-To-Stock (MTS) production-inventory (PI) systems, machines tend to be utilised more heavily than in Make-To-Order (MTO) systems, as unlike in MTO systems where machines are utilised only *after* an order is placed, products are produced on more continuous schedules in MTS systems. Therefore, unlike in MTO systems which have natural breaks in production, one has to set aside time for preventive maintenance in MTS systems.

In any system, a lack of maintenance causes machines to fail in catastrophic ways, which typically disrupts production patterns and leads to failures in meeting customer demand. These events often significantly increase operating costs. When performed in a systematic manner, TPM can reduce the frequency of such disruptions and deliver significant cost savings. Further, as the frequency of machine stoppages due to failures is minimised by TPM, the mean product lead time and the associated variance are reduced, which in turn improves the organisation's cost and schedule performance and raises profits. Reduction of mean lead times by TPM leads to a direct reduction in work-in-process inventory (Askin & Goldberg, 2001); reduction in lead time variability (variance) is a golden objective of supply chain management (Christensen, Germain, & Birou, 2007). TPM is therefore a key industry practice used in production systems due to its favourable impact on profits. See Ahuja and Khamba (2008) for a broad review of some of the recent relevant literature on TPM.

The key to TPM is statistically determining the *maintenance interval*, i.e. the time interval between successive

maintenance operations. In general, regular preventative maintenance activities result in lowering the frequency of costly repairs due to equipment failure. However, these planned maintenance operations do cause some downtime of their own and clearly cost money, albeit less than that resulting from unplanned emergency repairs. An excessive amount of preventative maintenance increases costs and reduces the advantages of avoiding repairs. A PI system seeks to make profits from selling products to customers. The goal of TPM in a PI system is hence to optimise the frequency of the maintenance interval, in order to maximise the profits or *net revenues*, i.e. the total revenues obtained from selling the finished products to the customers minus the sum of the costs of maintenance and the costs of repairs. As a system ages and since failure rates increase with time, preventative maintenance becomes an activity of critical importance from the perspective of productivity and profitability (Lewis, 1994).

The most accurate method for solving theoretical TPM problems is the Markov Decision Process (MDP), which is a special case of its more powerful variant, the semi-MDP (SMDP). Because of the large volume of state spaces in large-scale industrial MDPs/SMDPs and the intractability of constructing the associated transition probabilities, classical solution methods, called dynamic programming (Bellman, 1957; Howard, 1960), break down on real-world applications of TPM. In recent times, methods based on artificial intelligence (AI) have proven to be very powerful in solving numerous problems from manufacturing, especially via a reduction of the *dimensionality* of the problem (Wuest, Weimer, Irgens, & Thoben, 2016). Reinforcement Learning (RL), which is a branch of AI, significantly reduces the dimensionality of the underlying MDP/SMDP by bypassing the construction of the transition probabilities of the associated MDPs/SMDPs. For instance, an MDP with N states and K actions in each state would need to store for each action, a matrix of the size: $N \times N$. For this problem, KN^2 values would be needed in the computer for dynamic programming. This number starts exceeding a million when $N = 1000$, which is common of large-scale TPM problems. An RL method such Q-Learning, on the other hand, would require only KN values for the same problem, thereby bringing about a significant reduction (of a factor of N) in the dimensionality of the solution approach. E.g. in a problem with 1000 states and two actions, dynamic programming would require 2 million values to be stored, while Q-Learning would need only 2000. As such, RL methods have simplified the solution of many industrial problems (Bertsekas & Tsitsiklis, 1996; Gosavi, 2015; Sutton & Barto, 1998).

In this paper, a new RL algorithm called iSMART (short for imaging-Semi-Markov Average Reward

Technique) is developed to solve the SMDP underlying a large-scale TPM problem. A special feature of the new algorithm is that unlike many RL algorithms, it does not need a reduction of the so-called exploration rate. A truly robust RL algorithm should ideally work under constant exploration, where one does not have to adjust the exploration as the algorithm makes progress. The new algorithm is designed to meet this goal for SMDPs that seek to minimise average costs. The algorithm is first tested on small SMDPs, where the optimal solutions can be derived via dynamic programming. Thereafter, it is tested on medium-scale and large-scale TPM problems in MTS PI systems, where its numerical performance is very encouraging and the storage data required to solve the problem is greatly reduced.

The remainder of this paper is organised as follows. Section 2 provides a background for the advantages of iSMART and for applying RL to solve TPM problems. Section 3 defines the maintenance problem studied here, along with a discussion on the relevant literature on TPM and MTS systems. Section 4 motivates the need for the new algorithm and presents its step-by-step details. Section 5 is devoted to numerical results obtained from using the new algorithm and its benchmark on the TPM problem. Section 6 concludes this paper with a summary of the work done and a discussion on potential future research.

2. Background

Algorithms of classical RL, also called *shallow* RL, require *tuning* of many parameters, such as (a) learning rates, (b) features of state representation, and (c) exploration rates, in order to deliver good performance. Tuning of parameters means determining their optimal values and/or adjusting their values as the algorithm progresses – via time-consuming trials and errors. The field of *deep learning* seeks to develop algorithms that can work *automatically* without any tuning/adjustment of these critical parameters, saving significant computational time and also delivering robust behaviour. See the recent insightful review in Bertsekas (2019) for a good discussion on a subset of the main issues on the interface of deep learning and reinforcement learning. Constant learning rates that make learning-rate-decaying schedules redundant have been proposed in Beck and Srikant (2012). The recent work of Silver et al. (2016) seeks to use multiple neural networks for automatic feature representation, within the Alpha-Zero programme that attains super-human performance in playing chess, as opposed to hand-tuned (or hand-crafted) features used in shallow RL. Their automatic approach makes it unnecessary to handcraft features for the function approximation. Finally, constant

exploration rates have been mathematically analysed in Rokhlin (2018) for standard Q-Learning. But other than this work, to the best of the authors' knowledge, constant exploration rates have not been studied extensively in the literature. It is well-known, however, that inappropriate tuning of exploration parameters can lead to very disappointing behaviour in many RL algorithms, including in approximate policy iteration (Bertsekas & Tsitsiklis, 1996). Therefore, the new algorithm seeks to make advances in this direction – in particular, in the domain of average cost SMDPs.

RL algorithms have been used for solving problems in manufacturing, supply chains, and robotics (Aissani, Beldjilali, & Trentesaux, 2009; Chaharsooghi, Heydari, & Zegordi, 2008; Kober, Bagnell, & Peters, 2013; Mortazavi, Khamseh, & Azimi, 2015). The work in Das, Gosavi, Mahadevan, and Marchallick (1999) was one of the first pieces of research that sought to solve TPM problems in PI systems via shallow RL and developed an algorithm called SMART (Semi-Markov Average Reward Technique), which has led to a family of algorithms over the years: λ -SMART (Gosavi, Bandla, & Das, 2002), R-SMART (Gosavi, 2004), and R-MART (Zhu & Ukkusuri, 2014) to name a few. All the algorithms in the SMART family require reduction (or decaying) of its so-called exploration parameter (or rate) appropriately. Decaying means reducing the value of the exploration rate as the learning progresses. iSMART is a new algorithm in the SMART family, and as stated above, a prominent feature of iSMART is that it does *not* require any decay of the exploration parameter, but instead works under constant exploration.

The new algorithm thus removes the exploration rate parameter from the training process, thereby saving computational time and the effort of trials and errors needed to determine optimal exploration rates. Further, the new algorithm is developed keeping in mind problems in which state transitions are of a *cyclical* nature and the system transitions through one or more unique states *repeatedly*. In TPM problems, the system regularly cycles through the failed state where production comes to a halt and as such is characterised by the cyclical nature. Finally, it should be noted that unlike the model-based approach in Das and Sarkar (1999), which requires first setting up the transition-probability model and then using a non-linear optimisation procedure, the RL approach requires only a simulator and delivers results in a shorter computer time. The reason for the elongated times in the model-based approach is that evaluating the transitional probabilities first requires setting up expressions with multiple integrals, which can take a long time (offline), and then needs computing of the transition probabilities via numerical integration, which

is also a time-consuming process within the computer. An additional advantage of the RL approach is that it is *not* restricted to the distributions used in model-based approaches and can be used for any given distribution for the inputs. Model-based approaches are often severely restricted in use to the distributions employed in creating the model and break down when the distributions found in the real-world system do not match those used in the model. In the real world, the PI system studied here is known to have a variety of distributions for its inputs, and hence this makes the algorithm especially suitable for studying PI systems. The next section delves into details of the PI system and the underlying stochastic model for the TPM problem.

3. Problem definition

The literature offers two models for determining the maintenance-interval: renewal theory (Ross, 2014) and MDPs/SMDPs (Bertsekas, 2014). Determining the optimal maintenance interval in the MTS production-inventory (PI) system requires using the *age* of the machine as well as the *inventory level* in the warehouse. Renewal-theoretic models can account for the age but disregard the inventory level; the latter is critical when one considers PI systems, as shutting down the machine for maintenance is not appropriate when the inventory is running low. MDP/SMDP models, on the other hand, are more detailed and can capture the state of the system in terms of its age as well as the finished product inventory in the warehouse, thereby developing a model that can adjust its maintenance policy according to not only the age, but also the level of the inventory. One of the earliest applications of the MDP to solve the preventive maintenance problem can be found in Marcellus and Dada (1991). Van der Duyn Schouten and Vanneste (1995) and Das and Sarkar (1999) study TPM in the context of production-inventory systems. The model in Das and Sarkar (1999) is more versatile in the sense that it employs the very flexible gamma distribution for production, repair, and inter-failure times. The gamma distribution carries the critical property of increasing failure rates for their time between failures which is typical of most systems as equipment ages. Finally, their model uses the exponential distribution for inter-arrival times of demand. This leads to a Poisson rate of arrival, which can be well-approximated by the normal distribution demand patterns that are known to exist widely in production systems (Askin & Goldberg, 2001). As such, we employ the generally applicable model of Das and Sarkar (1999) for testing our new algorithm. However, our algorithm will be simulation-based and will hence be capable of using any given distribution for the

input parameters. In other words, our algorithm is flexible enough for usage with other MDP/SMDP models for studying TPM in PI systems.

In general, MDPs are problems of sequential decision-making in discrete-event systems that are controlled by the so-called Markov chains, which can capture the state of the system as it changes dynamically. Using a controller/manager to specify the action selected in a given state of the system, one can optimise system performance via consideration of a quantifiable metric, namely, maximising the net revenues (rewards) obtained over an infinite time horizon, where the net revenues contain the profits from selling the finished product and the costs incurred in repair (replacing a belt or a broken bearing) and maintenance (e.g. an oil change). Maximising the net revenues is mathematically equivalent to minimising the net costs. Semi-MDPs (SMDPs) are more generalised versions of MDPs, which take time into consideration, meaning the time of transition between states does not have to be constant, which is the case for the PI system considered here. In SMDPs, the time of transitions is a random variable, which allows for added flexibility in capturing the behaviour of real-world systems, and is explicitly a part of the objective function. In this paper, the problem of identifying the optimal maintenance interval in an MTS PI system is set up as one that maximises the average reward (net revenues) in an SMDP. The details of the system are now presented in detail.

Consider the MTS PI system as shown in Figure 1. The PI system produces a batch of finished products with the goal of meeting the external demand, which is typical of many real-world systems. The production system in the figure could represent a single machine/workstation or an assembly line made of numerous workstations. The production system is defined by its age, while the warehouse or inventory buffer is characterised by its inventory level. More details on these aspects will be presented below, but it should be noted that the simulation model should be detailed enough to capture the system's age and the inventory buffer level, so the RL algorithm can be employed in conjunction with the simulation model to deliver useful optimisation results.

In these types of production systems, when a manager decides that maintenance needs to occur, instead of triggering a new production cycle, the system is shut down to carry out the maintenance activity. If a production cycle is initiated and the system does not fail during the production activity, a batch of finished products is delivered to the warehouse in MTS systems. On the other hand, the system can break down during the production cycle, i.e. a so-called failure occurs. At that point, the production process is halted and a repair is performed. When a

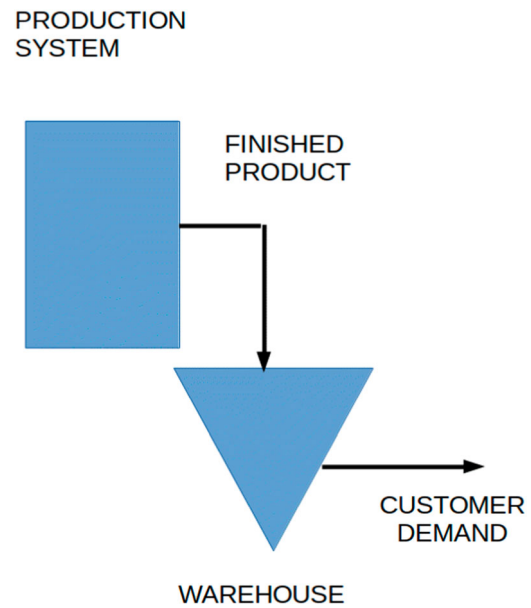


Figure 1. The production-inventory system under MTS.

machine is down for maintenance or repair, it does not produce any parts. The machine production cycle ends after each batch of the product has been manufactured. With every consecutive completed production cycle that has occurred without any maintenance or repairs, the probability of system failure generally increases, which is known as increasing failure rate. With increasing failure rates, maintenance is known to save money (Lewis, 1994). In this work, the number of consecutive production cycles completed without maintenance or repairs is used to mathematically represent the *age* of the machine, while the number of finished products in the warehouse is used to mathematically denote the *level of inventory*. Together, the age of the machine and the inventory level in the warehouse will define the state of the system. More formally, the state of the PI system in our model will be defined, following Das and Sarkar (1999), by the 2-tuple: (φ, c) , where φ denotes the number of consecutive production cycles completed since the last repair or maintenance (or age), and c denotes the number of units of finished products in the warehouse (or inventory level). In what follows, the nature of the transitions of the system and how the revenues and costs are captured in the simulation model are described.

After every successful production cycle, a decision must be made to either produce a product (i.e. initiate a production cycle) or maintain the machine. Thus, there are two actions that can be taken: either (1) produce or (2) maintain. The revenue/cost structure of the above-described transitions in the simulator can be defined as follows. Let C_m and C_r denote the cost of one maintenance and cost of one repair (in dollars), respectively;

the revenues earned from selling one unit of the product (batch) will be denoted by P (dollars). The production action can only be chosen when the inventory buffer is below the upper limit (U). The production action can only be chosen when the inventory buffer is below the upper limit (U). The machine can either progress from $(\varphi = m)$ to $(\varphi = m + 1)$ by selecting the action of production and successfully completing the production cycle, or it can transition from $(\varphi = m)$ to $(\varphi = 0)$ if the machine fails during the production cycle. When the machine fails it incurs a cost of Cr (dollars). When the maintenance action is chosen, the machine goes from $(\varphi = m)$ to state $(\varphi = 0)$ and a cost of Cm (dollars) is incurred. In other words, the age of the machine is reset to 0 after a failure (which is followed by a repair) or a maintenance, while the age increases by 1 after every successful production cycle is completed. Upper and lower limits are assumed on the finished product inventory buffer; this implies that when the inventory buffer reaches its upper limit (U), production is stopped, the system is said to go on 'vacation', and the production is not started again until the buffer reaches its lower limit (L) due to customer demand. As a result, c , the inventory level can never exceed U . Backordering is not allowed in the model of Das and Sarkar (1999), and hence the lowest value of c will be 0. When no backordering is allowed, any demand that arrives when the inventory level equals zero is assumed to be lost.

When a production cycle is successful, the inventory level goes from $(c = s)$ to $(c = s + 1)$ if no demand arrives during the production cycle; during this transition, no profits are earned. If z demand arrivals occur during a successful production cycle, the inventory level goes from $(c = s)$ to $(c = \max(s + 1 - z, 0))$; if the production cycle is unsuccessful and ends up in a failure, the inventory level goes from $(c = s)$ to $(c = \max(s - z, 0))$. The expressions above can be explained as follows. Consider the case of the successful production first. With a successful production, the inventory level goes from s to $(s + 1)$ at the end of the production cycle, but the z demands that arrive during the production process deplete the inventory, and hence the final inventory level equals $(s + 1 - z)$ if backordering is allowed; but since backordering is not permitted in this model, the lowest value of the inventory should equal zero and therefore the expression for the final value of the inventory level is $\max(s + 1 - z, 0)$. The case of the unsuccessful production can be explained similarly with the ending state as $\max(s - z, 0)$. Every time a demand arrives and the inventory level is positive, the system earns a profit of P (dollars).

The stochastic model will be assumed to have the following inputs: the production times (time to take to

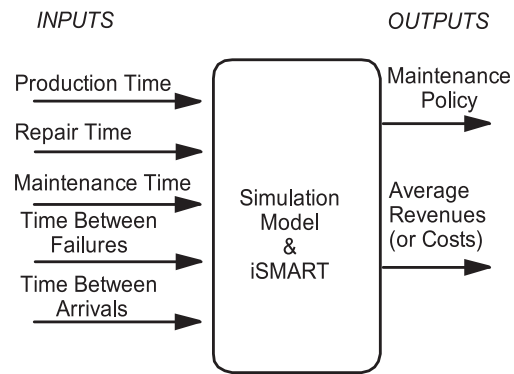


Figure 2. A schematic for the inputs and the outputs of the simulation model of the production-inventory system when combined with the RL algorithm.

produce one batch of the finished product), the repair times, the maintenance times, the time between demand arrivals (TBD), and the time between failures (TBF). Within a simulator, state transitions and the time between state transitions are generated on the basis of the inputs named above via random number generation (Law & Kelton, 2000). As stated above, the production time, the time between failures, and the repair time for this model will be assumed to belong to the very flexible gamma distribution. Production times in industry are known to carry the gamma distribution (Benjaafar, Kim, & Vishwanadham, 2004), while repair times tend to be variable as all repairs are not the same and include the time needed for investigation, which introduces additional variability into the duration of the repair times. The gamma distribution is very customisable and allows the user to alter both the shape and scale (magnitude) in such a way that it can fit a variety of probability distribution functions. The TBD will be assumed to follow the exponential distribution. The maintenance time will be assumed to follow the uniform distribution, as it is less variable due to its planned nature. Note that these are the assumptions made in Das and Sarkar (1999), although the simulation model can work with any given distribution. The simulation model, in combination with the RL algorithm, will deliver the optimal maintenance policy and the expected revenues or reward delivered by the optimal maintenance policy; Figure 2 depicts the inputs with known distributions and outputs from the simulation-based RL model.

4. The new algorithm: iSMART

This section is devoted to describing the iSMART algorithm in detail. Key notation that will be needed for the algorithm is now presented:

- i, j, l : Indices for states in the system
- a : Index for an action
- $\mathcal{A}(i)$: Set of actions allowed in state i
- $|\mathcal{A}(i)|$: Cardinality of $\mathcal{A}(i)$, i.e. the number of actions permitted in state i
- S : Set of states in the system
- N : The number of states in the system
- k : Iteration number in the RL algorithm
- $Q(i, a)$: Q-factor for state-action pair (i, a)
- $R(i, a)$: R-factor for state-action pair (i, a)
- $T(i, a)$: T-factor for state-action pair (i, a)
- $q(i, a)$: The probability of selecting action a in state i in the exploration scheme
- $p(i, a, j)$: The probability of going from state i to state j under the influence of action a
- $r(i, a, j)$: The constant immediate reward obtained in going from state i to state j under the influence of action a
- $t(i, a, j)$: The average time consumed in going from state i to state j under the influence of action a
- α, β : Step sizes or learning rates
- d : The policy, which is an n -tuple that specifies the action to be chosen in each state; thus $d(l)$ will denote the action specified by policy d in state l .
- $\langle d(1), d(2), \dots, d(N) \rangle$: The policy d defined in terms of its actions for each state from 1 through N
- $L_d(i)$: The steady-state probability of state i under policy d
- ρ : The average reward; ρ^* will denote the *optimal* average reward and $\rho(d)$ will denote the average reward when policy d is pursued.

The algorithm iSMART is a Q-Learning-type algorithm for average reward SMDPs in which, in addition to the classical Q-factors, the average reward is estimated via supplementary state-action values called R- and T-factors. These R- and T-factors essentially seek to serve as the *image* of the greedy policy contained in the Q-factors; the notion of greediness in Q-factors will be explained below. Thus, the key idea here is that Q-factors will denote the state-action values based on the traditional Bellman equation update, while the R-factors and T-factors will estimate the state-action values for the one-step transition (immediate) reward and the same for one-step transition time, respectively, for the so-called *greedy* policy contained in the Q-factors.

Very importantly, unlike R-SMART, the algorithm will work under fixed exploration. In other words, the probability of selecting any given action will *not* be changed during the course of the algorithm. Thus, for instance, if there are K actions, each action could be chosen with a uniform probability of $1/K$. Each action could alternatively be chosen with some other constant probability

such that the sum of the probabilities of all actions equals one. It is important to point out that the decaying of the exploration is typically performed via a rule, whose parameters have to be tuned.

We now describe how reducing of exploration is typically done in RL. In what follows, $q(i, a)$ denotes the probability of selecting action a in state i and k denotes the number of iterations in the algorithm, where $k \geq 1$. The following rules, shown via Equations (1) and (2), are commonly used in the literature to reduce exploration:

$$q(i, a) = AB^{k-1} \quad (1)$$

where for instance $A = 0.5$ in case there are two actions and $B = 0.9999$;

$$q(i, a) = A/(B + k - 1) \quad (2)$$

where for instance $A = 5,000$ and $B = 10,000$. Appropriate values of A and B depend on the application at hand and have to be determined via trial and error; in other words, they need tuning. If the latter is not done properly, algorithms such as R-SMART fail to converge to optimal solutions even on small-scale problems. Thus, an important advantage of this new algorithm is its ability to work with *fixed* exploration and deliver convergent behaviour.

As stated before, like any other Q-learning algorithm, iSMART is based on value iteration. This allows the Bellman optimality equation to be the underlying foundation for determining the optimal solution using Q-learning algorithms. In other words, the solutions generated by iSMART are expected to reach those of the Bellman optimality equation, which is known to generate the optimal solution (Bertsekas, 2014). The optimal solution (or policy) delivers the optimal average reward, ρ^* . Clearly, the goal is hence to determine the policy that produces the optimal average reward. The average reward of the policy d is the net revenue per unit time when that policy is pursued and is technically defined as follows (Bertsekas, 2014):

$$\rho(d) = \frac{\sum_{i=1}^N L_d(i) \sum_{j=1}^N p(i, d(i), j) r(i, d(i), j)}{\sum_{i=1}^N L_d(i) \sum_{j=1}^N p(i, d(i), j) t(i, d(i), j)}. \quad (3)$$

Thus, the goal in solving the SMDP is to identify the optimal policy, d^* , whose average reward acquires the highest possible value, ρ^* , i.e.

$$\rho^* \stackrel{\text{def}}{=} \text{Max}_d \rho(d). \quad (4)$$

As stated above, to determine the optimal solution, the Q-factor version of the Bellman optimality equation for

average reward SMDPs is needed, which is as follows:

$$Q(i, a) = \sum_{j=1}^N p(i, a, j) [r(i, a, j) - \rho * t(i, a, j) + \max_{b \in \mathcal{A}(j)} Q(j, b)] \quad (5)$$

for all $i \in S$ and $a \in \mathcal{A}(i)$.

Using the above equation, a Q-learning update can be derived as follows:

$$Q^{k+1}(i, a) \leftarrow [1 - \alpha^k] Q^k(i, a) + \alpha^k [r(i, a, j) - \rho * t(i, a, j) + \max_{b \in \mathcal{A}(j)} Q^k(j, b)] \quad (6)$$

in a transition from state i to state j under the influence of action a , where $Q^k(i, a)$ denotes the value of the Q-factor for (i, a) , i.e. $Q(i, a)$, in the k th iteration, and α^k denotes the value of the step size or learning rate in the k th iteration. However, it should be noted that the above equation is difficult to use in practice because ρ^* is not known at the start. Due to this, iSMART not only updates Q-factors, but also needs to update the values of ρ . The update of ρ will be performed using the so-called 'mirror image' discussed above. The mirror image will constitute of the R - and T - factors that will essentially pursue the *greedy* action stored in the Q-factors. The idea of the greedy action in Q-factors can be illustrated as follows. Suppose for state 2, we have the following current values of the Q-factors: $Q(2, 1) = 95.9$ and $Q(2, 2) = 109.8$. Since we are maximising the objective function (average reward), the action 2 will be the greedy action for state 2 at this time, because for this set of values: $\max\{Q(2, 1), Q(2, 2)\} = Q(2, 2)$. As the algorithm progresses, the greedy actions change, converging to the optimal actions in the limit.

The central idea underlying iSMART is that the mirror image leads to the solution of the Bellman optimality equation. To be more specific, if (i^*, a^*) represents a distinguished state-action pair frequently visited in the system, then $R(i^*, a^*)/T(i^*, a^*)$, which is derived from the mirror image, should ideally converge to ρ^* in the limit, as the number of algorithm iterations converges to infinity. This allows the algorithm to generate Q-factors that solve Equation (5), which is the Bellman optimality equation guaranteed to deliver the optimal solution. As an example on how this distinguished state-action pair is selected, consider this: assume that state 1 is visited frequently in the system and action 1 is chosen regularly in that state, then one could set $(i^*, a^*) = (1, 1)$. Clearly, identifying this state-action pair requires some knowledge of the system dynamics. In the step-by-step description of the algorithm that follows, the inputs to the

algorithm are initialised in Step 1 and the outputs are generated in the final step: Step 7. The algorithm runs within a discrete-event simulator of the PI system and cycles through the sequence of iterative steps: Steps 2 through 6.

Steps in iSMART:

A step-by-step explanation of the seven steps in iSMART algorithm will now be presented.

Step 1 (Inputs): Set the number of iterations, k , to 1. Set $\rho^k = 0$, where ρ^k is an estimate of the optimal average reward in the k^{th} iteration. Set $Q^k(i, a)$, $R^k(i, a)$, and $T^k(i, a)$ to 0 for all $i \in S$ and all $a \in \mathcal{A}(i)$. Set k_{\max} to a large number that will allow the algorithm to successfully explore all states and actions. Set (i^*, a^*) to any state-action pair in $S \times \mathcal{A}$ (preferably a state-action pair that is visited *frequently*.)

Step 2: Start the simulation at an arbitrary state i . Select an action with a probability of $\frac{1}{|\mathcal{A}(i)|}$. Note that this probability is the exploration rate that was discussed before. This probability is never changed during the course of the algorithm, but may have to be set to a value other than $\frac{1}{|\mathcal{A}(i)|}$, *depending on the nature of the problem*. This will be discussed in more detail later in the work.

Step 3: Simulate action a . Let the next state be denoted by j . Let $r(i, a, j)$ denote the immediate reward from state i to state j under action a . Also let $t(i, a, j)$ denote the time spent under the same state-action transition.

Step 4: Update the Q-factors as shown below. The term α denotes the step size or learning rate for the Q-factors.

$$Q^{k+1}(i, a) \leftarrow [1 - \alpha^k] Q^k(i, a) + \alpha^k [r(i, a, j) - \rho^k t(i, a, j) + \max_{b \in \mathcal{A}(j)} Q^k(j, b)] \quad (7)$$

Step 5: If $a \in \arg \max_{c \in \mathcal{A}(i)} Q^k(i, c)$ (i.e. the action selection in Step 3 was in fact a greedy one), update $R^k(i, a)$, $T^k(i, a)$, and ρ^{k+1} as follows:

$$R^{k+1}(i, a) \leftarrow [1 - \beta^k] R^k(i, a) + \beta^k [r(i, a, j) - R^k(i^*, a^*) + \max_{b \in \mathcal{A}(j)} R^k(j, b)]; \quad (8)$$

$$T^{k+1}(i, a) \leftarrow [1 - \beta^k] T^k(i, a) + \beta^k [t(i, a, j) - T^k(i^*, a^*) + \max_{b \in \mathcal{A}(j)} T^k(j, b)]; \quad (9)$$

$$\rho^{k+1} = R^{k+1}(i^*, a^*)/T^{k+1}(i^*, a^*). \quad (10)$$

Step 6: If $k < k_{\max}$, set $i \leftarrow j$ and $k \leftarrow k + 1$ and return to Step 2. Otherwise, continue to Step 7.

Step 7 (Outputs): For each $l \in S$, compute $d^*(l) \in \arg \max_{b \in \mathcal{A}(l)} Q^k(l, b)$. The policy returned by the algorithm is d^* , where the action in state l is given by $d(l)$.

5. Numerical results

This section provides numerical results from experiments with the new algorithm. The first subsection is devoted to results on small-scale SMDPs with four state-action pairs, while the second subsection presents results on the TPM problem for the MTS PI system discussed above with several thousand state-action pairs.

5.1. Small-scale SMDP systems

The four small-scale SMDPs discussed here will have two states and two actions allowed in each state, which leads to 4 state-action pairs. These are thus small-scale problems whose transition probabilities and rewards are known explicitly, and consequently the optimal policy in each problem is known with certainty. The larger problems that will be studied in the next section will have significantly larger state-action spaces, and at best their transition probabilities are estimated via numerical integration, which carries its own error. While a convincing test of any RL algorithm is on large-scale problems, it is typical in the literature to first test a new RL algorithm on small-scale problems, such as mountain car or pole balancing (see Sutton and Barto (1998) and references therein), to establish that the algorithm is known to never fail on small-scale problems. Thus, the first set of experiments is a demonstration of the algorithm's performance on small-scale problems; the latter can also be viewed as miniature versions of the large-scale versions described later.

For the small-scale problems, some additional notation is needed: P_a , TRM_a , and TTM_a will denote the transition probability matrix, transition reward matrix, and transition time matrix for action a , respectively. Note that $P_a(i, j) = p(i, a, j)$, where $P_a(i, j)$ denotes the element in the i th row and j th column of the matrix P_a . Similarly, $TRM_a(i, j) = r(i, a, j)$ and $TTM_a(i, j) = t(i, a, j)$. Input data for the four small-scale systems tested are provided next.

System 1:

$$P_1 = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}; \quad P_2 = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix};$$

$$TRM_1 = \begin{bmatrix} 6 & -5 \\ 7 & 12 \end{bmatrix}; \quad TRM_2 = \begin{bmatrix} 10 & 17 \\ -14 & 13 \end{bmatrix};$$

$$TTM_1 = \begin{bmatrix} 10 & 5 \\ 120 & 60 \end{bmatrix}; \quad TTM_2 = \begin{bmatrix} 50 & 75 \\ 7 & 2 \end{bmatrix}.$$

System 2:

$$P_1 = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}; \quad P_2 = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix};$$

$$TRM_1 = \begin{bmatrix} 6 & 5 \\ 7 & 12 \end{bmatrix}; \quad TRM_2 = \begin{bmatrix} 10 & 17 \\ 14 & 13 \end{bmatrix};$$

$$TTM_1 = \begin{bmatrix} 10 & 5 \\ 120 & 60 \end{bmatrix}; \quad TTM_2 = \begin{bmatrix} 5 & 75 \\ 7 & 20 \end{bmatrix}.$$

System 3:

$$P_1 = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}; \quad P_2 = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$$

$$TRM_1 = \begin{bmatrix} 6 & -5 \\ 70 & 12 \end{bmatrix}; \quad TRM_2 = \begin{bmatrix} 12 & 17 \\ 6 & 13 \end{bmatrix};$$

$$TTM_1 = \begin{bmatrix} 10 & 5 \\ 120 & 60 \end{bmatrix}; \quad TTM_2 = \begin{bmatrix} 50 & 75 \\ 7 & 20 \end{bmatrix}.$$

System 4:

$$P_1 = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}; \quad P_2 = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix};$$

$$TRM_1 = \begin{bmatrix} 16 & 5 \\ 75 & 120 \end{bmatrix}; \quad TRM_2 = \begin{bmatrix} 80 & 10 \\ 6 & 1 \end{bmatrix};$$

$$TTM_1 = \begin{bmatrix} 10 & 5 \\ 120 & 60 \end{bmatrix}; \quad TTM_2 = \begin{bmatrix} 50 & 75 \\ 7 & 20 \end{bmatrix}.$$

Simulators for these systems and the embedded algorithm were coded in MATLAB and run using a 2.60 GHz Intel Core i7-6700HQ processor on a 64-bit Windows operating system. Each system was run for 10,000 time units using a probability of 0.5 for each action, and the distinguished state-action pair was chosen to be: $(i^*, a^*) = (1, 1)$. The following rules were used for the learning rates:

$$\alpha^k = 150/(300 + k). \quad (11)$$

$$\beta^k = 10/(300 + 3k) \quad (12)$$

The results obtained are presented in Table 1. The optimal policy and the optimal average reward ρ^* , were obtained via policy iteration. We explain the results obtained for System 3 for illustration. For state 1: $Q(1, 1) = 511.08 > Q(1, 2) = 485.85$, which implies that action 1 is the optimal action for state 1 according to the algorithm. Similarly, for state 2: $Q(2, 1) = 528.66 > Q(2, 2) = 520.59$,

Table 1. Results on small-scale SMDPs.

System	iSMART Outputs				Policy Iteration Outputs		
	$Q(1,1)$	$Q(1,2)$	$Q(2,1)$	$Q(2,2)$	$\frac{R(1,1)}{T(1,1)}$	ρ^*	Optimal Policy
1	104.47	70.44	-60.97	108.00	2.0700	2.1045	< 1,2 >
2	174.76	175.84	111.75	170.78	0.8409	0.8357	< 2,2 >
3	511.08	485.85	528.66	520.59	0.7169	0.7442	< 1,1 >
4	461.93	464.59	436.29	421.59	1.3450	1.3401	< 2,1 >

which means the optimal action is 1 for state 2 as well. Thus, policy d^* delivered by the algorithm for this system is $\langle 1, 1 \rangle$, which coincides with the optimal policy. Further, note that the optimal average reward's estimate from the algorithm is $\frac{R(1,1)}{T(1,1)} = 0.7169$, which closely approximates the value provided by policy iteration: $\rho^* = 0.7442$. It is clear from a similar inspection of the Q-factors in the table that for every system, the iSMART algorithm generates the optimal policy.

5.2. TPM results from medium- and large-scale systems

Results of using iSMART and RSMART on the TPM problem are presented here. The gamma distribution, as discussed above, is used for the time between failures, production time, and repair times, while the exponential distribution is used for the time between customer (demand) arrivals and the uniform distribution is used for the maintenance time. The mean of the gamma distribution parametrised by (n, λ) is given by n/λ , and its variance by n/λ^2 . The uniform distribution for the maintenance time will be characterised by (a, b) , where a is the lower limit and b the upper limit. The exponential distribution for the time between arrivals will be defined by its single parameter μ , such that the mean is $1/\mu$. The inventory limit is defined by the upper and lower bounds (L, U) .

The mean time between failures (MTBF) is used to classify problems into (i) medium-scale problems and (ii) large-scale problems. The medium-scale cases have MTBF of 100 time units or less, while the large-scale cases have MTBF exceeding 100 time units. The cost values used for the model, as described above, are C_m and C_r , which are the cost of one maintenance and cost of one repair, respectively; the revenues earned from selling one unit are denoted by P . Table 2 details the data for two sets of cost/revenue parameters used in our experiments. Table 3 provides the inputs for the first 10 cases studied, which are medium-scale problems, while Table 4 provides the input data for the four large-scale cases (the last four cases). In all the experiments, $L = 2$ and $U = 3$.

For all large-scale cases, the maintenance time parameters are: $a = 5$ and $b = 20$. For all cases, (i^*, a^*) was set to equal $(0, 1)$ i.e. the state where the age is zero and inventory level equals 1.

Like in the small-scale systems of the previous subsection, the simulator of the MTS PI system and the embedded algorithm were coded in MATLAB and run using a 2.60 GHz Intel Core i7-6700HQ processor on a 64-bit Windows operating system. The system for each case was run for 1000,000 time units, and this is called the learning phase in which the optimal policy is being generated (learned). The learning rates defined in Equations (11) and (12) were used with iSMART for these cases; these are standard rules used widely in other experiments in the literature (Gosavi, 2015). In other words, no additional tuning or tinkering of these rates was needed with iSMART. Further, it should be noted that the same rules worked for all the cases.

The policy generated here is of a threshold nature, where there is an upper limit on the threshold beyond which no production must be performed and maintenance should be conducted when that threshold is

Table 2. Cost and profit parameters.

Cost/Revenue Structure	CS1	CS2
C_m	2	2
C_r	5	10
P	1	0.5

Table 3. Input parameters for the medium-scale cases: these cases use the cost structure CS1.

Case	Time between demands ($1/\mu$)	Time between failures (n, λ)	Time for production (n, λ)	Maintenance Time (a, b)	Repair Time (n, λ)	Cost Revenue Structure
1	10	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	CS1
2	5	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	CS1
3	7	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	CS1
4	15	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	CS1
5	20	(8, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	CS1
6	10	(4, 0.1)	(8, 0.8)	(5, 20)	(2, 0.01)	CS1
7	10	(4, 0.08)	(8, 0.8)	(5, 20)	(2, 0.01)	CS1
8	10	(8, 0.08)	(4, 0.4)	(5, 20)	(2, 0.01)	CS1
9	10	(8, 0.08)	(8, 0.8)	(2, 10)	(2, 0.01)	CS1
10	10	(8, 0.08)	(8, 0.8)	(5, 20)	(1, 0.05)	CS1

Table 4. Input parameters for the large-scale cases: maintenance time for all the cases was set at (5, 20).

Case	Time between demands ($1/\mu$)	Time between failures (n, λ)	Time for production (n, λ)	Repair Time (n, λ)	Cost/Revenue Structure
11	10	(4, 0.01)	(8, 0.8)	(2, 0.01)	CS1
12	10	(8, 0.008)	(8, 0.8)	(2, 0.01)	CS1
13	10	(4, 0.01)	(8, 0.8)	(2, 0.01)	CS2
14	10	(8, 0.008)	(8, 0.8)	(2, 0.01)	CS2

Optimal Maintenance Policy Based on Production Cycles and Inventory Levels Example

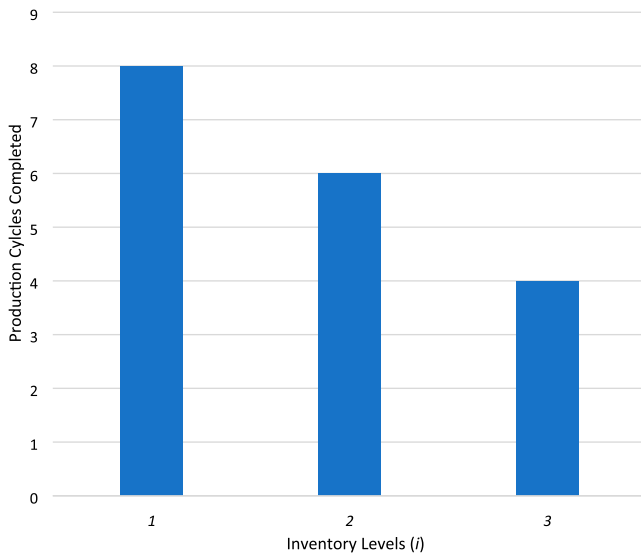


Figure 3. Optimal maintenance policies based on age and inventory levels.

reached. As such, it is economical and also more insightful to describe the optimal policy in terms of the thresholds, rather than in terms of actions prescribed for every state. The optimal policy for each system is hence denoted by 3 integers: (i_1, i_2, i_3) . This notation implies that if the inventory level is c , the optimal action is to produce until the production count is less than i_c and to maintain when the production count equals i_c for $c = 1, 2$, and 3. This can also be explained via Figure 3 where the y -axis is the production count at which maintenance should be performed, and the x -axis is the inventory level. In this example, the optimal solution was recorded as $(8, 6, 4)$. This means that when the inventory level equals 1, the system should be maintained after 8 production cycles are complete; if the inventory level equals 2, the system should be maintained after 6 production cycles are complete; if the inventory level equals 3, the system should be maintained after 4 production cycles are complete. The optimal policy description given here and displayed in this work omits inventory levels of 0 because when the inventory level is 0, the optimal action has shown to be to produce regardless of the production count (Das & Sarkar, 1999). Presenting thresholds of this nature also make practical sense on shop floors where implementation is typically done from charts, such as the one shown in Figure 3.

The simulator for each case was then run again using the optimal policy for 100,000 time units with 8 replications. This is called the frozen phase. The learning and frozen phases typically took 45 and 27 s, respectively.

Table 5. Numerical results from iSMART benchmarked against those from the numerical optimisation in Das and Sarkar (1999).

Case	Results from Optimisation			Results from iSMART	
	Policy	ρ^*	Pr	Policy	ρ^{iSMART}
1	(6, 5, 5)	0.0296	0.65	(4, 5, 5)	0.0296 ± 0.0005
2	(6, 6, 5)	0.0237	0.65	(6, 8, 7)	0.0236 ± 0.00062
3	(6, 5, 5)	0.0273	0.65	(5, 5, 5)	0.0283 ± 0.0009
4	(6, 6, 5)	0.0267	0.65	(5, 5, 5)	0.0284 ± 0.0005
5	(6, 6, 6)	0.0232	0.75	(5, 5, 4)	0.0239 ± 0.0004
6	(4, 4, 4)	-0.0054	0.65	(5, 3, 5)	-0.0055 ± 0.0006
7	(4, 4, 4)	-0.0001	0.65	(4, 3, 4)	-0.0007 ± 0.006
8	(6, 6, 5)	0.0287	0.65	(5, 5, 6)	0.0296 ± 0.0011
9	(7, 6, 5)	0.0261	0.65	(7, 5, 7)	0.0272 ± 0.0009
10	(8, 8, 6)	0.0413	0.65	(6, 5, 5)	0.0422 ± 0.0008
11	(21, 19, 13)	0.0616	0.91	(16, 16, 31)	0.0621 ± 0.00018
12	(63, 59, 41)	0.0754	0.95	(56, 63, 53)	0.0751 ± 0.00028
13	(19, 18, 15)	0.0235	0.89	(17, 19, 21)	0.0240 ± 0.00084
14	(57, 54, 42)	0.0354	0.95	(50, 57, 70)	0.0356 ± 0.00013

Table 5 presents the results from running iSMART on all the medium-scale and large-scale cases for TPM on the MTS PI system. The table shows the optimal policy, as obtained from the numerical optimisation performed in Das and Sarkar (1999), along with the average reward delivered by it in the simulator. The table then shows the policy delivered by iSMART and its average reward from running the frozen phase. Pr in the table denotes the probability of selecting the action produce, which is never changed, thereby leading to fixed exploration. In most cases, it has to be held above 0.5, due to the nature of the problem. The upper and lower bounds for 95% confidence intervals (CI) on the average reward (ρ) are also provided for the policy delivered by iSMART. In all cases, iSMART's performance is very close to that of the optimal policy. It should be noted that in some cases, the optimal policy is slightly outperformed, and there are two reasons for that: (i) the average reward is computed from discrete-event simulations, which introduce approximations and (ii) the optimal policy in Das and Sarkar (1999) uses numerical integrations to determine the transition probabilities, which can introduce approximations of their own and hence slightly different results.

The performance of iSMART was benchmarked against that of R-SMART, as iSMART has been derived from R-SMART. As stated above, it is expected that RSMART needs tuning of the exploration rate, since the exploration rate has to be decayed properly. Our experiments on the MTS PI system showed that with R-SMART, not only does one have to tune the exploration rate properly, but one also has to tune the learning rate for the average reward (ρ) i.e. β . In other words, the rule used in Equation (12) did not always generate satisfactory performance. The reason for this is that the convergence of R-SMART relies critically on the trajectory of ρ that the algorithm's path generates (Gosavi, 2004). On the other

Table 6. Results from using R-SMART for cases 1:10 where the exploration rate is given by $q(i, a) = (\text{Pr})^k$ and the learning rate for the average reward ρ is given by β^k .

Case	β^k	Pr	Policy	$\rho^{R-SMART}$
1	$\frac{100}{300+k}$	0.9999	(5, 3, 4)	0.0297 ± 0.0014
2	$\frac{105}{300+k}$	0.9999	(4, 4, 6)	0.0234 ± 0.00064
3	$\frac{1000}{2000+k}$	0.99999	(5, 4, 5)	0.0284 ± 0.0008
4	$\frac{1000}{2000+k}$	0.99999	(4, 4, 4)	0.0261 ± 0.0005
5	$\frac{1500}{3000+k}$	0.99999	(6, 5, 4)	0.0236 ± 0.0004
6	$\frac{1500}{3000+k}$	0.99999	(3, 4, 5)	-0.0055 ± 0.0004
7	$\frac{1200}{2400+k}$	0.99999	(4, 5, 4)	0.00015 ± 0.0007
8	$\frac{1100}{2300+k}$	0.99999	(4, 5, 4)	0.0283 ± 0.0006
9	$\frac{1300}{2500+k}$	0.99999	(5, 5, 5)	0.0266 ± 0.0011
10	$\frac{1500}{3000+k}$	0.99999	(6, 7, 7)	0.0417 ± 0.0005

hand, in iSMART, ρ is estimated via the R - and T -factors, which makes the process of estimating ρ more stable, and hence a variety of rules can work for β . Tables 6 and 7 present the results of experiments with R-SMART; Table 6 is for the first 10 medium-scale cases and Table 7 is for the large-scale cases (Cases 11 through 14). These tables show how for each case the exploration rate and the learning rate β may have to be selected separately in R-SMART in order to generate near-optimal solutions. In the experiments with R-SMART, the following exploration rate based on Equation (1) was used: For every $i \in S$,

$$q(i, 1) = (\text{Pr})^k, \quad (13)$$

where $(\text{Pr})^k$ denotes the scalar Pr raised to the power k and $q(i, 2) = 1 - q(i, 1)$ for every $i \in S$; note that $a = 1$ in $q(i, a)$ denotes the production action, while $a = 2$ denotes the maintenance action. The value of Pr chosen for each case is shown in Tables 6 and 7. Also, shown in these tables is the learning rate β chosen for each case. For all the cases, the following rules was used for the learning rate α :

$$\alpha^k = 1500/(3000 + k) \quad (14)$$

As is clear from examining Tables 6 and 7, R-SMART also generates good policies, albeit with the experimentation needed to identify the appropriate learning rate β and the appropriate exploration rate. This is further explained next.

Computational Time Savings with iSMART: The actual running times of the two algorithms, iSMART and R-SMART, are almost identical. The critical difference lies

Table 7. Results from using R-SMART for Cases 11:14 where the exploration rate is given by $q(i, a) = (\text{Pr})^k$ and the learning rate for the average reward ρ is given by β^k .

Case	β^k	Pr	Policy	$\rho^{R-SMART}$
11	$\frac{1300}{2500+k}$	0.999999	(19, 22, 23)	0.0620 ± 0.00033
12	$\frac{1300}{300+k}$	0.999999	(54, 98, 93)	0.0758 ± 0.00029
13	$\frac{1300}{2500+k}$	0.999999	(19, 22, 23)	0.0231 ± 0.0013
14	$\frac{1300}{2500+k}$	0.999999	(57, 89, 93)	0.0353 ± 0.00017

in the tuning the exploration rate parameter and the learning rate β parameter. This is the time taken to identify parameters that work and deliver good behaviour from the algorithm. This kind of tuning means that one must run the system simulation with a set of parameters and if these parameters do not work, one must try another set of parameters. Thus, this involves time spent offline. *This offline time period can be significantly long.* The small-scale cases took about 5–10 min of experimentation, while the large-scale cases needed about 30 min. It is to be further noted that a key goal of deep learning and artificial intelligence is to have robust and automatic parameters that require little tuning, and it is here that the advantage of iSMART is truly obvious. We must note, however, that the storage burden of iSMART is somewhat heavier than that of R-SMART. This is because of the additional R - and T -factors. Thus, for R-SMART with 100 states and two actions, one needs 200 Q -factors and the scalar ρ , i.e. 201 scalars. On the other hand, iSMART requires in addition to the 200 Q -factors, 200 R -factors and 200 T -factors, i.e. 600 scalars. Increased computational burden is becoming less of an issue with modern computers used by companies such as Google. Nonetheless, ideally future research should also consider avenues to more robust behaviour without adding to the computational burden, as additional computational burden appears to be hurting the environment (Strubell, Ganesh, & McCallum, 2019).

6. Conclusions

TPM is a critical practice in MTS PI systems, which can save large firms millions of dollars annually, and is as such an important component of effective management strategy. The problem becomes complicated in large-scale systems where the number of states is very large. This paper considered the TPM problem in MTS PI systems under some very general assumptions on the input parameters, such as the time between failures, the repair time, the production time, the maintenance time, and the time between successive customer (demand)

arrivals. The paper also proposed a new version of an RL algorithm called iSMART to solve this challenging problem. The new algorithm is based on a Q-factor version of the Bellman optimality equation that uses a mirror imaging principle to seek optimality and is designed to overcome a critical deficiency of R-SMART, specifically the need for decaying exploration. Finding the correct rate for exploration adds an extra layer to the computational burden of the R-SMART algorithm. iSMART works with an exploration rate that is never changed during the course of the algorithm and thus removes the need for tuning of the exploration parameter – thereby saving a significant amount of time in the computational exercise associated with running the algorithm on large-scale problems. In the experiments conducted, iSMART was able to generate optimal solutions on every small-scale problem tested and delivered very robust performance on larger-scale problems from the TPM domain.

A preliminary version of this paper with a small subset of results presented here has appeared in a conference (Encapera & Gosavi, 2017). In future work, mathematical convergence of the iSMART algorithm should be studied. Further, studying non-Poisson arrivals and multiple machines will lead to other exciting avenues for future research, where fluid-based approaches may be more suitable.

Acknowledgements

The authors wish to thank the editors and the reviewers for their careful reading of the manuscript that has led to a significantly improved paper and in particular the presentation of additional numerical results for benchmarking of the new algorithm.

Disclosure statement

No potential conflict of interest was reported by the authors.

Notes on contributors

Angelo Encapera obtained an M.S. in Systems Engineering from Missouri University of Science and Technology, Rolla, MO, USA in Dec, 2017. He obtained a B.S. in Petroleum Engineering in May 2016, also from Missouri University of Science and Technology, Rolla, MO, USA. He is currently employed at Garmin Inc. as a Systems Engineer. His research interests include machine learning and systems engineering.

Abhijit Gosavi obtained a Ph.D. in Industrial Engineering from the University of South Florida, Tampa, FL, USA. He obtained a B.S. and M.S., both in Mechanical Engineering from Jadavpur University, Kolkata, India, and the Indian Institute of Technology, Madras, India, respectively. He currently serves as an Associate Professor in the Department of Engineering Management and Systems Engineering at Missouri University of Science and Technology, Rolla, MO. His research interests include

simulation-based optimisation, reinforcement learning, and total productive maintenance.

Susan L. Murray obtained a Ph.D. in Industrial Engineering from Texas A&M University, College Station, TX, USA. She obtained a M.S. in Industrial Engineering from the University of Texas at Arlington, TX, USA and a B.S. in Industrial Engineering from Texas A&M University, College Station, TX, USA. She currently serves as Chair and Professor of Industrial Psychology at Missouri University of Science and Technology, Rolla, MO. Her research interests include safety, total productive maintenance, and human factors.

ORCID

Abhijit Gosavi  <http://orcid.org/0000-0002-9703-4076>

Susan L. Murray  <http://orcid.org/0000-0003-0985-3854>

References

- Ahmed, S., Hj. Hassan, M., & Taha, Z. (2005). TPM can go beyond maintenance: Excerpt from a case implementation. *Journal of Quality in Maintenance Engineering*, 11(1), 19–42.
- Ahuja, I. P. S., & Khamba, J. S. (2008). Total productive maintenance: Literature review and directions. *International Journal of Quality & Reliability Management*, 25(7), 709–756.
- Aissani, N., Beldjilali, B., & Trentesaux, D. (2009). Dynamic scheduling of maintenance tasks in the petroleum industry: A reinforcement approach. *Engineering Applications of Artificial Intelligence*, 22(7), 1089–1103.
- Askin, R. G., & Goldberg, J. B. (2001). *Design and analysis of lean production systems*. New York, NY: John Wiley & Sons.
- Beck, C. L., & Srikant, R. (2012). Error bounds for constant step-size Q-learning. *Systems and Control Letters*, 61(12), 1203–1208.
- Bellman, R. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.
- Benjaafar, S., Kim, J. S., & Vishwanadham, N. (2004). On the effect of product variety in production-inventory systems. *Annals of Operations Research*, 126(1–4), 71–101.
- Bertsekas, D. (2014). *Dynamic programming and optimal control* (4th ed.). Belmont: Athena.
- Bertsekas, D. (2019). Feature-based aggregation and deep reinforcement learning: A survey and some new implementations. *IEEE/CAA Journal of Automatica Sinica*, 6(1), 1–31.
- Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Belmont: Athena Press.
- Chaharsooghi, S. K., Heydari, J., & Zegordi, S. H. (2008). A reinforcement learning model for supply chain ordering management: An application to the beer game. *Decision Support Systems*, 45, 949–959.
- Christensen, W. J., Germain, R. N., & Birou, L. (2007). Variance vs average: Supply chain lead-time as a predictor of financial performance. *Supply Chain Management: An International Journal*, 12(5), 349–357.
- Das, T. K., Gosavi, A., Mahadevan, S., & Marchallick, N. (1999, April). Solving semi-markov decision problems using average reward reinforcement learning. *Management Science*, 45(4), 560–574.
- Das, T., & Sarkar, S. (1999). Optimal preventive maintenance in a production inventory system. *IIE Transactions*, 31, 537–551.

- Encapera, A., & Gosavi, A. (2017, June 4–8). *A new reinforcement learning algorithm with fixed exploration for semi-markov control in preventive maintenance*. Proceedings of the ASME 2017, 12th International Manufacturing Science and Engineering Conference MSEC2017 (Vol. 3), Los Angeles, CA, USA.
- Gosavi, A. (2004). Reinforcement learning for long-run average cost. *European Journal of Operational Research*, 155, 654–674.
- Gosavi, A. (2015). *Simulation-based optimization: Reinforcement learning and parametric optimization techniques* (2nd ed.). New York: Springer Science.
- Gosavi, A., Bhandla, N., & Das, T. K. (2002). A reinforcement learning approach to airline seat allocation for multiple fare classes with overbooking. *IIE Transactions*, 34, 729–742.
- Howard, R. (1960). *Dynamic programming and markov processes*. Cambridge, MA: MIT Press.
- Ireland, F., & Dale, B. G. (2001). A study of total productive maintenance implementation. *Journal of Quality in Maintenance Engineering*, 7(3), 183–191.
- Jardine, A. K., & Tsang, A. H. (2005). *Maintenance, replacement, and reliability: Theory and applications*. Boca Raton, FL: CRC Press.
- Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11), 1238–1274.
- Law, A. M., & Kelton, W. D. (2000). *Simulation modeling and analysis* (Vol. 3). New York: McGraw-Hill.
- Lewis, E. E. (1994). *Introduction to reliability engineering* (2nd ed.). Hoboken, NJ: John Wiley and Sons.
- Manzini, R., Regattieri, A., Pham, H., & Ferrari, E. (2009). *Maintenance for industrial systems*. New York, NY: Springer Science & Business Media.
- Marcellus, R. L., & Dada, M. (1991). Interactive Process Quality Improvement. *Management Science*, 37(11), 1365–1376.
- McCall, J. J. (1965). Maintenance policies for stochastically failing equipment: A survey. *Management Science*, 11(5), 493–524.
- McKone, K. E., & Weiss, E. (1998). TPM: Planned and autonomous maintenance: Bridging the gap between practice and research. *Production and Operations Management*, 7(4), 335–351.
- Mortazavi, A., Khamseh, A. A., & Azimi, P. (2015). Designing of an intelligent self-adaptive model for supply chain ordering management system. *Engineering Applications of Artificial Intelligence*, 37, 207–220.
- Rokhlin, D. B. (2018). *Robbins-Monro conditions for persistent exploration learning strategies*. arXiv preprint arXiv:1808.00245.
- Ross, S. M. (2014). *Introduction to probability models* (10th ed.). Cambridge, MA: Academic Press.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., . . . Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587), 484–489.
- Strubell, E., Ganesh, A., & McCallum, A. (2019). *Energy and policy considerations for deep learning in NLP*. arXiv preprint arXiv:1906.02243.
- Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.
- Van der Duyn Schouten, F. A., & Vanneste, S. (1995). Maintenance optimization of a production system with buffer capacity. *European Journal of Operational Research*, 82(2), 323–338.
- Wuest, T., Weimer, D., Irgens, C., & Thoben, K.-D. (2016). Machine learning in manufacturing: Advantages, challenges, and applications. *Production & Manufacturing Research*, 4(1), 23–45.
- Zhu, F., & Ukkusuri, S. V. (2014). Accounting for dynamic speed limit control in a stochastic traffic environment: A reinforcement learning approach. *Transportation Research Part C: Emerging Technologies*, 41, 30–47.