Availability and performability analysis for a service degradation process with condition-based preventive maintenance II – estimation and simulation

Tadashi Dohi

Department of Information Engineering, Hiroshima University, 1-4-1 Kagamiyama, Higashi-Hiroshima 739-8527, Japan
E-mail: dohi@rel.hiroshima-u.ac.jp

Abstract: The preventive maintenance is very useful to improve effectively the service availability for software systems with service degradation. In this paper, we present a stochastic model to describe an operational software, which consists of one operating system and multiple applications and provides a service in continuous time. In this paper we examine adaptive estimation problems using a reinforcement learning algorithm which is called the $Q$-learning. We give the $Q$-learning algorithms to estimate the optimal software rejuvenation policies that minimise the service availability and maximise the expected service reward per unit time, respectively. Illustrative numerical examples are presented to investigate asymptotic properties of estimators of the optimal software rejuvenation policies.

Keywords: software service; service degradation; software rejuvenation; condition-based maintenance; semi-Markov decision process; statistics; adaptive estimation; reinforcement learning.


Biographical notes: Tadashi Dohi received his BSc (Engineering), MSc (Engineering) and PhD (Engineering) from Hiroshima University, Japan, in 1989, 1991 and 1995, respectively. In 1992, he joined the Department of Industrial and Systems Engineering, Hiroshima University, Japan as an Assistant Professor. Since 2002, he has been working as a Full Professor in the Department of Information Engineering, Graduate School of Engineering, Hiroshima University. In 1992 and 2000, he was a Visiting Research Scholar in University of British Columbia, Canada and Duke University, USA, respectively, on leave of absence from Hiroshima University. His research areas include software reliability engineering, dependable computing and performance evaluation. He is a regular member of ORSJ, REAJ, IEICE, IPSJ and IEEE.
1 Introduction

The value of service provided by computer systems is increasing day by day more than that of service equipments themselves. Hence, the notions of service reliability and service availability are becoming popular in our highly information-oriented society. In Dohi (2013), we develop a dynamic software rejuvenation model and derive the optimal software rejuvenation policy, where the transition law of the service state is assumed to be known. In general, the model parameters, which are involved in stochastic models, need to be estimated from the underlying data. The commonly used parameter estimation method is based on the fixed sample of data, so that the data have to be given in advance. However, this is not trivial for the operational service systems because it is difficult to observe a sufficient number of data before the service operation. Dohi (2013) consider the Markov and semi-Markov decision process models to trigger the software rejuvenation policy by monitoring the system degradation state, where the underlying transition rates have to be known. In other words, the service degradation process should be estimated in any adaptive ways the service time goes on.

Eto et al. (2008), Okamura et al. (2010) and Okamura and Dohi (2011) concern adaptive estimation problems of the condition-based optimal software rejuvenation policies and apply the reinforcement learning algorithm, which is called the $Q$-learning, to several service degradation models. The $Q$-learning is discussed more specifically within the framework of Markov and/or semi-Markov decision processes. Abounadi et al. (2001) improves the classical $Q$-learning algorithm and show that it can converge to the optimal relative value function in the dynamic programming equation almost surely. Borkar and Meyn (2000) and Konda and Borkar (1999) prove some convergence properties on the $Q$-learning-based algorithms with the ordinary differential equation (O.D.E.) method and the martingale convergence theorem, respectively. Mahadevan (2000) pays his attention to the numerical calculation in the $Q$-learning. For the good survey on the $Q$-learning algorithms in Markov/semi-Markov decision processes, see Bertsekas and Tsitsiklis (1996) and Gosavi (2003).

According to Okamura et al. (2010) and Okamura and Dohi (2011), we also develop the $Q$-learning algorithm for the service availability maximisation and the expected reward minimisation problems in Dohi (2013). The most attractive point is that the $Q$-learning algorithm is statistically consistent, i.e., the estimate of the optimal software rejuvenation time approaches to the real (but unknown) optimal solution as the number of observations increases. While, the $Q$-learning principle is based on a simple try-and-error discipline, so that the convergence speed to the real optimal solution is not so fast in general. We take place the simulation experiments to investigate asymptotic properties of estimators of the optimal software rejuvenation policies. Also we carry aout the sensitivity of the turning parameter on the optimal software rejuvenation schedule as a condition-based preventive maintenance policy for software systems.
2 Service degradation model

To complete the discussion, we consider a software system which consists of one operating system and multiple applications in Dohi (2013). The system can provide a service in continuous time and, at the same time, may deteriorate with time. Suppose that State 0 and State \( s + 1 \) are the normal (highly robust) state and the service down state, respectively. The service starts with State 0 at time 0 and makes a transition to \( s \) degradation levels \( k = 1', 2', \cdots, s' \) stochastically at any random time. At each change of state, the corrective maintenance will be made and the reconfiguration of some applications on the system will be made reactively by listing executing processes. Then, the service state becomes from \( k = 1', 2', \cdots, s' \) to \( j = 1, \cdots, s \) after the corrective maintenance, where the constant set up time, \( k_j \), depending on the state \( j = 0, 1, \cdots, s \) and the corrective maintenance reward per unit time, \( c_1 > 0 \), will be required. However, since the service cannot become as good as new only by the corrective maintenance, where the constant set up time, \( k_j \), depending on the state \( j = 0, 1, \cdots, s \) and the corrective maintenance reward per unit time, \( c_1 > 0 \), will be required. However, since the service cannot become as good as new only by the reconfiguration of applications, the state just after the corrective maintenance makes a transition from \( j = 1, \cdots, s - 1 \) to \( k = 2', \cdots, s' \) 

Suppose that the service level of software at time \( t \) is described by a semi-Markov process (SMP) with \( 2(s + 1) \) states and constant sojourn time \( k_j \ (j = 1, \cdots, s) \). As Dohi (2013) show, since this SMP can be simplified to a right-skip free CTMC by aggregation of states \( k = (1', 2', \cdots, s') \) and \( j = 1, \cdots, s \), it can be reduced to the simpler form depicted in Figure 2. More specifically, let \( \{N(t), t \geq 0\} \) be a right-skip free CTMC with space state \( I = \{0, 1, 2, \cdots, s, s + 1\} \), where the transition rate from \( i \) to \( j \ (i, j = 0, \cdots, s + 1, i < j) \) is given by \( \gamma_{i,j} > 0 \) and \( \sum_{j=i+1}^{s+1} \gamma_{i,j} = \Gamma_i \) for all \( i = 0, 1, \cdots, s \) (see Dohi, 2013). When the service failure occurs, the service is down and the state makes a transition from an arbitrary state \( j = 0, 1, \cdots, s \) to State \( s + 1 \). Then, both the recovery operation and the reinstallation of operating system immediately start, where the time to complete the recovery operation is an independent and identically distributed (i.i.d.) random variable having the cumulative distribution function (c.d.f.) \( H_{i+1}(x) \) and mean \( 1/\omega_{s+1} > 0 \).

On the other hand, one makes a decision whether to trigger the software rejuvenation (reinstallation of operating system) as a preventive maintenance at the time instant when the state of software service changes from \( i = 0, 1, \cdots, s - 1 \) to \( j = i + 1, i + 2, \cdots, s \). If one decides to continue operation (Action 2), the state is monitored until the next change of state, otherwise, the software rejuvenation (Action 1) is preventively triggered, where the time to complete the rejuvenation is also an i.i.d. random variable with the c.d.f. \( H_i(x) \) and mean \( 1/\omega_i > 0 \), depending on the state \( i = 0, 1, \cdots, s \). When the service failure occurs, i.e., the service state becomes \( j = s + 1 \), the recovery operation (Action 3) is taken. Let \( c_2 > 0 \) and \( c_3 > 0 \) be the rejuvenation reward per unit time and the recovery reward per unit time, respectively. In both periods of rejuvenation and recovery operation, the system operation is stopped. Also, it is assumed that the state-dependent reward \( r_i > 0 \) is incurred per unit operation time for \( i = 0, 1, \cdots, s \). Let \( q_{i,j}(\delta) \) denote the probability that the service state changes from \( i \) to \( j \) under Action \( \delta \ (\delta = 1, 2, 3) \). Then it is seen that

\[
q_{i,0}(1) = \int_0^\infty dH_i(t) = 1, \quad i = 0, 1, \cdots, s, \quad (1)
\]
where the mean rejuvenation time (overhead) is given by
\[ h_i = \int_0^\infty t dH_i(t). \] (2)

2. Case 2 (continuation of processing):
\[ q_{i,j}(2) = \gamma_{i,j}/\Gamma_i, \quad i, j = 0, 1, \ldots, s + 1, i < j. \] (3)

3. Case 3 (recovery from failure):
\[ q_{i,0}(3) = \int_0^\infty dH_{s+1}(t) = 1, \] (4)
where the mean recovery time (overhead) is given by
\[ h_{s+1} = \int_0^\infty t dH_{s+1}(t). \] (5)

After completing rejuvenation and recovery operations, the service state becomes as good as new, i.e., \( j = 0 \) in equations (1) and (4), and the same cycle repeats again and again over an infinite time horizon. We define the time interval from the initial point to the completion of rejuvenation or recovery operation whichever occurs first, as one cycle.

3 Dynamic programming formulations

Observing the service state of software, we sequentially determine the optimal timing to trigger the software rejuvenation so as to minimise the steady-state service unavailability. Define the following functions:

- \( \delta_A(i) \): action taken at state \( i \), i.e.,
\[ \delta_A(i) = \begin{cases} 1 : & 0 \leq i \leq s \\ 2 : & 0 \leq i \leq s \\ 3 : & i = s + 1 \end{cases} \] (6)

- \( G_A(i, \delta_A(i)) \): expected service down time between successive two decision points when action \( \delta_A(i) \) is taken at state \( i \),
\[ G_A(i, \delta_A(i)) = \begin{cases} h_i : & \delta_A(i) = 1 \\ k_i : & \delta_A(i) = 2 \\ h_{s+1} : & \delta_A(i) = 3 \end{cases} \] (7)

- \( \pi_A(i, \delta_A(i)) \): total expected time between successive two decision points when action \( \delta_A(i) \) is taken at state \( i \),
\[ \pi_A(i, \delta_A(i)) = \begin{cases} h_i : & \delta_A(i) = 1 \\ 1/\Gamma_i : & \delta_A(i) = 2 \\ h_{s+1} : & \delta_A(i) = 3 \end{cases} \] (8)
U(i): action space at state \( i \).

\( v_A(i) \): value function at state \( i \in I \).

\( UA_\infty \): service unavailability in the steady state, where \( UA_\infty^* \) denotes the minimum one.

From the preliminary above, the Bellman equation based on the principle of optimality (see Tijms, 1994) is given by

\[
v_A(i) = \min_{\delta_A \in U(i)} \left[ G_A(i, \delta_A) - UA_\infty \cdot \pi_A(i, \delta_A) + \sum_{j=0}^{s+1} q_{i,j}(\delta_A) \cdot v_A(j) \right]. \tag{9}
\]

Next, we consider the performability model based on the expected reward function per unit time. Define the following functions:

\( \delta_R(i) \): action taken at state \( i \), i.e.,

\[
\delta_R(i) = \begin{cases} 
1 : 0 \leq i \leq s \\
2 : 0 \leq i \leq s \\
3 : i = s + 1
\end{cases} \tag{10}
\]

\( G_R(i, \delta_R(i)) \): expected service reward between successive two decision points when action \( \delta_R(i) \) is taken at state \( i \),

\[
G_R(i, \delta_R(i)) = \begin{cases} 
-c_2 h_i : \delta_R(i) = 1 \\
r_i (\frac{1}{\tau_i} - k_i) - c_1 k_i : \delta_R(i) = 2 \\
-c_3 h_{s+1} : \delta_R(i) = 3.
\end{cases} \tag{11}
\]

\( \pi_R(i, \delta_R(i)) \): total expected time between successive two decision points when action \( \delta_R(i) \) is taken at state \( i \),

\[
\pi_R(i, \delta_R(i)) = \begin{cases} 
h_i : \delta_R(i) = 1 \\
1/\tau_i : \delta_R(i) = 2 \\
h_{s+1} : \delta_R(i) = 3.
\end{cases} \tag{12}
\]

\( U(i) \): action space at state \( i \).

\( v_R(i) \): value function at state \( i \in I \).

\( UR_\infty \): expected service reward in the steady state, where \( UR_\infty^* \) denotes the maximum one.

From the preliminary above, the Bellman equation based on the principle of optimality (Tijms, 1994) is given by

\[
v_R(i) = \max_{\delta_R \in U(i)} \left[ G_R(i, \delta_R) - UR_\infty \cdot \pi_R(i, \delta_R) + \sum_{j=0}^{s+1} q_{i,j}(\delta_R) \cdot v_R(j) \right]. \tag{13}
\]
4 Reinforcement learning

4.1 Q-learning

The reinforcement learning is a simulation-based optimisation algorithm, where the optimal value function is approximated with the sample or simulation of observations. In general, the reinforcement learning scheme consists of

1 an environment
2 a learning agent with its knowledge base
3 a set of actions taken by the agent
4 responses from the environment to the different actions in different states (see Figure 1).

![Figure 1](Configuration of reinforcement learning algorithm)

That is, the agent learns an interaction from the environment based on try-and error principle, and adapts the environment itself. The agent can receive the information called reward from the environment, and can learn the parameters which govern the environment. In this paper, we focus on the representative reinforcement learning, called Q-learning (see Sutton and Barto, 1998; Watkins and Dayan, 1992), which consists of the following three factors:

- **observation of state**: observe the current state
- **selection of actions**: select the best action from possible ones at the current state, where the best action is taken based on an estimate of reward ($Q$ value)
- **learning from environment**: update $Q$ value with both the current $Q$ value and the reward earned by the selected action.

4.2 Service availability

Define the following notation:

- $Q_A(i, \delta_A(i))$: estimate of future cumulative service down time just after the action $\delta_A(i)$ is taken in state $i$
$t_A(i, \delta_A(i), j)$: transition time to state $j$ just after the action $\delta_A(i)$ is taken in state $i$

$r_A(i, \delta_A(i), j)$: service down time until the state transition to $j$ occurs just after the action $\delta_A(i)$ is taken in state $i$

$REWARD^A_t$: cumulative service down time in the steady state, provided that an agent selects the action $\delta_A(i)$ with probability $1/|U(i)|$ so as to minimise the $Q$ value

$TIME^A_t$: cumulative operation time at each decision point

$\phi$: design parameter in the $Q$-learning algorithm

$k$: number of iterations

$UA_t$: transient (instantaneous) service unavailability, i.e.,

$$UA_t = \frac{REWARD^A_t}{TIME^A_t}, \quad (14)$$

where $\lim_{t \to \infty} UA_t = UA_\infty$.

We derive the $Q$-factor version of the value iteration algorithm mentioned in Dohi (2013). In the first phase of $Q$-learning algorithm (Step 1~Step 5 below), the learning agent learns the $Q$ value as an estimate of future cumulative service down time based on a probabilistic action, and adapts the environment through the update of $Q$ value. In the second phase (Step 6~Step 9 below), the decision maker (DM) regards the first phase as a simulator, and selects the optimal action based on the updated $Q$ value by the agent. Although the DM’s action at each decision point does not influence the agent and the environment, he or she can behave optimally in the sense of minimisation of the $Q$ value, and can estimate the updated $Q$ value, say, estimates of the cumulative service down time and cumulative total operation time from the history. The estimates in this stage is transient, i.e., they can function to check the convergence.

$Q$-learning algorithm

**Step 1:** Agent observes the current state $i$ of software system. Set $k = 0$, $\phi = 1$, $REWARD^A_\infty = 0$, $TIME^A_\infty = 0$, $REWARD^A_t = 0$ and $TIME^A_t = 0$.

**Step 2:** For a sufficient large iteration number $k_z$, if $k \leq k_z$ at each observation point with state $i (= 1, 2, \ldots, s)$, then the agent uses a probabilistic strategy, i.e., takes an action $\delta_A(i)$ [rejuvenation (1) or continuation of process (2)] with probability $1/|U(i)|$. Further, if the action taken by the agent minimises $Q$ value, then $\phi = 0$, otherwise $\phi = 1$. On the other hand, if $k \geq k_z$, then the agent takes the optimal action which minimises $Q$ value and stop the procedure.
Step 3: After observing the transition from state $i$ to $j$, the agent updates the $Q$ value with the probabilistic strategy $\delta$ according to the following formula:

\[
Q_A(i, \delta_A) \leftarrow (1 - \alpha)Q_A(i, \delta_A) + \alpha \left\{ r_A(i, \delta_A, j) - U_A \infty t_A(i, \delta_A, j) + \min_{\delta_A' \in U(j)} Q_A(j, \delta_A') \right\},
\]

where $\alpha \in (0, 1]$ is the learning rate (constant).

Step 4: If $\phi = 0$, then update $REWARD_A \infty$ and $TIME_A \infty$ as shown below:

\[
REWARD_A \infty \leftarrow REWARD_A \infty + r_A(i, \delta_A, j),
\]
\[
TIME_A \infty \leftarrow TIME_A \infty + t_A(i, \delta_A, j).
\]

Step 5: Update the minimum service unavailability in the steady state $U_A \infty$ by

\[
U_A \infty \leftarrow \frac{REWARD_A \infty}{TIME_A \infty}.
\]

Step 6: The DM selects the optimal action at state $i$, minimizing the $Q$ value updated by

\[
\delta_A(i) = \arg\min_{\delta_A(i) \in U(i)} Q_A(i, \delta_A(i)).
\]

Step 7: Update $REWARD_A^t$ and $TIME_A^t$ by

\[
REWARD_A^t \leftarrow REWARD_A^t + r_A(i, \delta_A(i), j),
\]
\[
TIME_A^t \leftarrow TIME_A^t + t_A(i, \delta_A(i), j).
\]

Step 8: Update $U_A^t$ by

\[
U_A^t \leftarrow \frac{REWARD_A^t}{TIME_A^t}.
\]

Step 9: Set $k = k + 1$ and $i \leftarrow j$, and go to Step 2.

In actual implementation of the above algorithm, it should be noted that the action whether to trigger the software rejuvenation or not at each decision point is taken in Step 6. Then, an estimate of the service unavailability in the steady state is equivalent to that estimated by the learning agent in Step 5 and is independent of the DM’s action. On the other hand, when $U_A \infty$ at the maximum iteration number $k_z$, then one can check that the $Q$-learning algorithm converges and as the result the maximum service availability in the steady state is achieved in the software operation with rejuvenation. In the following section, we give an illustrative example and investigate the convergence properties of the $Q$-learning algorithm.
4.3 Service reward

Define the following notation:

- $Q_R(i, \delta_R(i))$: estimate of future cumulative service reward just after the action $\delta_R(i)$ is taken in state $i$
- $t_R(i, \delta_R(i), j)$: transition time to state $j$ just after the action $\delta_R(i)$ is taken in state $i$
- $r_R(i, \delta_R(i), j)$: service reward until the state transition to $j$ occurs just after the action $\delta_R(i)$ is taken in state $i$
- $REWARD_R^\infty$: cumulative service reward in the steady state, provided that an agent selects the action $\delta_R(i)$ with probability $1/|U(i)|$ so as to maximise the $Q$ value
- $TIME_R^\infty$: cumulative operation time in the steady state, provided that an agent selects the action $\delta_R(i)$ with probability $1/|U(i)|$ so as to maximise the $Q$ value
- $REWARD^R_t$: cumulative service reward at each decision point
- $TIME^R_t$: cumulative operation time at each decision point
- $\phi$: design parameter in the $Q$-learning algorithm
- $k$: number of iterations
- $UR_t$: transient (instantaneous) service reward, i.e.,

$$UR_t = \frac{REWARD^R_t}{TIME^R_t},$$

where $\lim_{t \to \infty} UR_t = UR_\infty$.

We derive the $Q$-factor version of the value iteration algorithm mentioned in Section 4. In the first phase of $Q$-learning algorithm (Step 1~Step 5 below), the learning agent learns the $Q$ value as an estimate of future cumulative service reward based on a probabilistic action, and adapts the environment through the update of $Q$ value. In the second phase (Step 6~Step 9 below), the DM regards the first phase as a simulator, and selects the optimal action based on the updated $Q$ value by the agent. Although the DM’s action at each decision point does not influence the agent and the environment, he or she can behave optimally in the sense of maximisation of the $Q$ value, and can estimate the updated $Q$ value, say, estimates of the cumulative service reward and cumulative total operation time from the history. The estimates in this stage is transient, i.e., they can function to check the convergence.

Q-learning algorithm

Step 1: Agent observes the current state $i$ of software system. Set $k = 0$, $\phi = 1$, $REWARD^R_\infty = 0$, $TIME^R_\infty = 0$, $REWARD^R_t = 0$ and $TIME^R_t = 0$.

Step 2: For a sufficient large iteration number $k_z$, if $k \leq k_z$ at each observation point with state $i (= 1, 2, \ldots, s)$, then the agent uses a probabilistic strategy, i.e., takes an action $\delta_R(i)$ [rejuvenation (1) or continuation of process (2)]]
with probability $1/|U(i)|$. Further, if the action taken by the agent maximises $Q$ value, then $\phi = 0$, otherwise $\phi = 1$. On the other hand, if $k \geq k_z$, then the agent takes the optimal action which maximises $Q$ value and stop the procedure.

**Step 3:** After observing the transition from state $i$ to $j$, the agent updates the $Q$ value with the probabilistic strategy $\delta$ according to the following formula:

$$Q_R(i, \delta_R) \leftarrow (1 - \alpha)Q_R(i, \delta_R) + \alpha\left\{r_R(i, \delta_R, j) - UR_\infty t_R(i, \delta_R, j) + \max_{\delta'_R \in U(j)} Q_R(j, \delta'_R)\right\},$$

where $\alpha \in (0, 1]$ is the learning rate (constant).

**Step 4:** If $\phi = 0$, then update $REWARD^R_\infty$ and $TIME^R_\infty$ as shown below:

$$REWARD^R_\infty \leftarrow REWARD^R_\infty + r_R(i, \delta_R, j),$$

$$TIME^R_\infty \leftarrow TIME^R_\infty + t_R(i, \delta_R, j).$$

**Step 5:** Update the maximum service reward in the steady state $UR_\infty$ by

$$UR_\infty \leftarrow REWARD^R_\infty / TIME^R_\infty.$$

**Step 6:** The DM selects the optimal action at state $i$, maximising the $Q$ value updated by

$$\delta_R(i) = \arg\max_{\delta_R(i) \in U(i)} Q_R(i, \delta_R(i)).$$

**Step 7:** Update $REWARD^R_t$ and $TIME^R_t$ by

$$REWARD^R_t \leftarrow REWARD^R_t + r_R(i, \delta_R(i), j),$$

$$TIME^R_t \leftarrow TIME^R_t + t_R(i, \delta_R(i), j).$$

**Step 8:** Update $UR_t$ by

$$UR_t \leftarrow REWARD^R_t / TIME^R_t.$$

**Step 9:** Set $k = k + 1$ and $i \leftarrow j$, and go to Step 2.

In actual implementation of the above algorithm, it should be noted that the action whether to trigger the software rejuvenation or not at each decision point is taken in Step 6. Then, an estimate of the service reward in the steady state is equivalent to that estimated by the learning agent in Step 5 and is independent of the DM’s action. On the other hand, when $UR_\infty \approx UR_t$ at the maximum iteration number $k_z$, then one can check that the $Q$-learning algorithm converges and as the result the maximum service reward in the steady state is achieved in the software operation with rejuvenation. In the following section, we give illustrative examples and investigate the convergence properties of the $Q$-learning algorithm.
5 Simulation experiments

5.1 Availability analysis

Here we give an illustrative example to determine the optimal rejuvenation schedule. We suppose that the number of system states is only 3: normal operation, deterioration (failure probable state) and system down (failure). In this section we consider a generalised Markovian deterioration process with six degradation levels (totally, eight states). We assume the same CTMC transition diagram with absorption representing the deterioration process of the software service (see Dohi, 2013), where each transition rate is assigned on each arc. Suppose that the rejuvenation time and the recovery time from service failure are given by \( \omega_1 = \omega_2 = \omega_3 = \omega_4 = \omega_5 = \omega_6 = 4.0 \text{ (hr}^{-1} \text{)}, \) and \( \omega_7 = 2.0 \text{ (hr}^{-1} \text{)}, \) respectively. Also, it is assumed that \( k_0 = 0.05 \text{ (hr)}, \) \( k_1 = 0.10 \text{ (hr)}, \) \( k_2 = 0.15 \text{ (hr)}, \) \( k_3 = 0.20 \text{ (hr)}, \) \( k_4 = 0.25 \text{ (hr)}, \) \( k_5 = 0.30 \text{ (hr)}, \) \( k_6 = 0.35 \text{ (hr)}. \) In Dohi (2013), we obtain the decision table to characterise the optimal software rejuvenation policy, and show that it is optimal to trigger the software rejuvenation at the first time when the system state reaches to \( i^* = N^* + 1 = 4. \) Then the associated minimum service unavailability in the steady state is given by \( UA_{\infty} = UA_{\infty}(N^*) = UA_{\infty}(3) = 0.0374026. \) Next, we consider more general case where the state transition is governed by a SMP. We also assume the same SMP with \( \omega = 4.0 \) and \( \chi = 2.0 \) similar to the CTMC case (see Dohi, 2013). In this case, the optimal threshold is given by \( N^* = 3 \) and the corresponding service unavailability in the steady state is \( UA_{\infty} = 0.04328 \) (see Dohi, 2013). From this result, the resulting rejuvenation schedule may not be strictly optimal. However, if we pay our attention to only the control-limit policy, this is of course optimal to maximises the service availability in the steady state.

Of our concern is the investigation of convergence properties of the \( Q \)-learning algorithm. We perform the Monte Carlo simulation with pseudo random numbers for the exponential and the Weibull distributions with the same parameters as SMP case, and observe the realisations of state deterioration time and system failure time. At each decision (observation) point, we behave so as to minimise the \( Q \) value and estimate both \( UA_{\infty} \) and \( UA_{t}. \) Throughout this paper, we fix the upper limit of iteration number as \( k_z = 10,000. \) Figure 2 shows the asymptotic behaviour of estimates of the service availability in the steady state for varying learning rate, \( \alpha = 0.01, 0.20, 0.50, \) where the horizontal line denotes the real optimal, \( 1 - UA_{\infty}^* = 0.95672. \) In the figure, we define the unit of a process as the time length of one cycle. From this figure, it is seen that estimates of the service availability in the steady state asymptotically converges to the real optimal solution \( 1 - UA_{\infty}^* = 0.95672 \) as the number of processes increases. Note again that this can be achieved with the probabilistic action by the agent. If the learning rate is given by \( \alpha = 0.10 \) and \( \alpha = 0.50 \) when the number of processes is fixed as 200, the relative error with respect to \( 1 - UA_{\infty}^* = 0.95672 \) becomes 0.14% and 0.77%, respectively. In general, though the smaller \( \alpha \) leads to much more computation, it dose not always guarantee the smaller error. For instance, when the number of processes is 800 with \( \alpha = 0.01, \) the relative error is given by 0.11%. That is, the careful adjustment of the learning rate would be important to realise the effective estimation. In Table 1, we calculate the estimation errors (%) between estimate and the real optimal. As the learning rate decreases, the estimation error decreases and afterward the \( Q \)-learning tends to underestimate the service availability in the steady state.
Next, we examine the asymptotic behaviour of the transient service availability based on the DM’s action. For the example in SMP case, if the complete information on the system deterioration/failure time distributions is available, one can know that the optimal threshold level is given by $N^* = 3$ and would continue applying the control-limit policy with $N^* = 3$. Then, our main interest is to examine the performance of the $Q$-learning characterised by choosing the minimum $Q$ value. That is, if the estimate of transient service unavailability, $UA_t$, is closed to the real optimal solution, $UA^*_\infty$, it can be concluded that the resulting $Q$-learning can provide the similar effect on service availability to the best (but unknown) control limit policy. In Figure 3, we carry out the simulation and compare the $Q$-learning algorithm with the rejuvenation schedule with fixed threshold level $N (= 3, 6)$. In the same parametric circumstance, we plot the behaviour of estimates of $1 - UA_t$. In this simulation experiment, since it is shown analytically that $N^* = 3$ is optimal, it can be easily expected that the simulation result with $N^* = 3$ approaches to $1 - UA^*_\infty = 0.95672$. On the other hand, the rejuvenation schedule based on the $Q$-learning gives the fluctuated results in earlier phase, and latter converges to the real optimal as the number of processes increases. It is evident that the simulation-based algorithm used here can never outperform the really
optimal rejuvenation policy, say, $N^* = 3$. However, in the situation where no statistical information on the system deterioration/failure time distributions is available, this non-parametric estimation scheme would be useful. Figure 4 illustrates the dependence of learning rate on the transient service availability $1 - UA_t$, where the estimate of service availability with smaller learning rate approaches to the service availability in the steady state $1 - UA_\infty$ as the number of processes increases.

**Figure 3** Comparison of $Q$-learning policy with some control-limit ones in simulation (see online version for colours)

![Graph showing comparison of transient system availability](image)

**Figure 4** Asymptotic behaviour of transient service availability based on $Q$-learning for varying learning rate (see online version for colours)

![Graph showing asymptotic behaviour](image)

5.2 **Reward analysis**

Similar to the previous discussion, we consider a generalised CTMC deterioration process with six degradation levels (totally, eight states), where each transition rate is assigned on each arc (see Dohi, 2013). Suppose that the rejuvenation time and the recovery time from service failure are given by $\omega_1 = \omega_2 = \omega_3 = \omega_4 =$
\[ \omega_5 = \omega_6 = 4.0 \text{ (hr}^{-1}) \text{, and } \omega_7 = 2.0 \text{ (hr}^{-1}) \text{, respectively. Also, it is assumed that } k_0 = 0.05 \text{ (hr} \text{), } k_1 = 0.10 \text{ (hr) , } k_2 = 0.15 \text{ (hr), } k_3 = 0.20 \text{ (hr), } k_4 = 0.25 \text{ (hr),} \\
\text{ } k_5 = 0.30 \text{ (hr), } k_6 = 0.35 \text{ (hr), } r_0 = 1.0 \text{ ($), } r_1 = 0.90 \text{ ($), } r_2 = 0.85 \text{ ($),} \\
r_3 = 0.80 \text{ ($), } r_4 = 0.70 \text{ ($), } r_5 = 0.60 \text{ ($), } r_6 = 0.50 \text{ ($), } c_1 = 0.5 \text{ ($), } c_2 = 8.0 \text{ ($),} \\
c_3 = 10.0 \text{ ($). In Dohi (2013), we show that it is optimal to trigger the software} \\
\text{rejuvenation at the first time when the system state reaches to } i^* = N^* + 1 = 4. \\
\text{Then the associated maximum expected service reward in the steady state is given by } UR_{\infty} \text{ in (3) = 0.706035. Also, we consider more general case} \\
\text{where the state transition is governed by a SMP. In this case the optimal threshold is} \\
given by } N^* = 2 \text{ and the corresponding expected service reward in the steady state is } UR_{\infty} = 0.708471 \text{ (see Dohi, 2013). As mentioned before, the resulting rejuvenation schedule may not be strictly optimal. However, if we pay our attention to only the} \\
\text{control-limit policy, this is of course optimal to maximise expected service reward in the} \\
\text{steady state. In Table 2, we calculate the estimation errors (%) between estimate and the real optimal. As the learning rate decreases, the estimation error decreases and} \\
\text{afterward the Q-learning tends to underestimate the expected reward as well. To examine} \\
\text{the effect of the factor } a, \text{ suppose that } r_0 = 1.0, r_1 = a, r_2 = a^2, r_3 = a^3, r_4 = a^4, \text{ } r_5 = a^5, r_0 = a^6, \text{ so we investigate the dependence of factor } a (< 0) \text{ on the optimal} \\
decision. Table 3 presents the dependence of factor } a \text{ on the optimal rejuvenation policy. In this example, as the factor becomes larger, the optimal rejuvenation timing becomes early.} \\

<table>
<thead>
<tr>
<th>\alpha</th>
<th>200</th>
<th>400</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>0.53%</td>
<td>0.33%</td>
<td>0.22%</td>
</tr>
<tr>
<td>0.05</td>
<td>0.64%</td>
<td>0.41%</td>
<td>0.25%</td>
</tr>
<tr>
<td>0.10</td>
<td>1.02%</td>
<td>0.71%</td>
<td>0.46%</td>
</tr>
<tr>
<td>0.20</td>
<td>1.39%</td>
<td>0.99%</td>
<td>0.74%</td>
</tr>
<tr>
<td>0.30</td>
<td>1.57%</td>
<td>1.37%</td>
<td>1.22%</td>
</tr>
<tr>
<td>0.40</td>
<td>2.07%</td>
<td>1.77%</td>
<td>1.71%</td>
</tr>
<tr>
<td>0.50</td>
<td>2.36%</td>
<td>2.03%</td>
<td>2.00%</td>
</tr>
</tbody>
</table>

Table 3 Dependence of factor } a \text{ on the optimal rejuvenation policy with six deterioration states}

<table>
<thead>
<tr>
<th>\alpha \text{ (a)}</th>
<th>0.70</th>
<th>0.75</th>
<th>0.80</th>
<th>0.85</th>
<th>0.90</th>
<th>0.95</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N^* = )</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>( z(N^*) )</td>
<td>0.676826</td>
<td>0.678814</td>
<td>0.686838</td>
<td>0.694862</td>
<td>0.707538</td>
<td>0.735934</td>
</tr>
</tbody>
</table>

We perform the Monte Carlo simulation with pseudo random numbers for the exponential and the Weibull distributions with the same parameters as SMP case, and observe the realisations of state deterioration time and system failure time. At each decision (observation) point, we behave so as to maximise the } Q \text{ value and estimate both } UR_{\infty} \text{ and } UR_t. \text{ Throughout this paper, we fix the upper limit of iteration number}
as $k_z = 10,000$. Figure 5 shows the asymptotic behaviour of estimates of the expected service reward in the steady state for varying learning rate, $\alpha = 0.01, 0.20, 0.50$, where the horizontal line denotes the real maximised value, $UR^*_\infty = 0.708471$. In the figure, we define the unit of a process as the time length of one cycle. From this figure, it is seen that estimates of the expected service reward in the steady state asymptotically converges to the real optimal solution $UR^*_\infty = 0.708471$ as the number of processes increases. Note again that this can be achieved with the probabilistic action by the agent.

If the learning rate is given by $\alpha = 0.10$ and $\alpha = 0.50$ when the number of processes is fixed as 200, the relative error with respect to $UR^*_\infty = 0.708471$ becomes 1.96% and 2.60%, respectively. In general, though the smaller $\alpha$ leads to much more computation, it does not always guarantee the smaller error. For instance, when the number of processes is 800 with $\alpha = 0.01$, the relative error is given by 0.56%. That is, the careful adjustment of the learning rate would be important to realise the effective estimation by taking account of computational effort. In Table 3, we calculate the estimation errors (%) between estimates and the real optimal with high accuracy, where $\frac{|UR^*_t - UR^*_\infty|}{UR^*_\infty}$. As the learning rate decreases, the estimation error decreases and afterward the $Q$-learning tends to estimate the expected service reward in the steady state.

Figure 5  Asymptotic behaviour of expected service reward in the steady state based on $Q$-learning for varying learning rate (see online version for colours)

Next, we examine the asymptotic behaviour of the transient service reward based on the DM's action. For the example, if the complete information on the system deterioration/failure time distributions is available, one can know that the optimal threshold level is given by $N^* = 2$ and would continue applying the control-limit policy with $N^* = 2$. Then, our main concern is to examine the performance of the $Q$-learning characterised by choosing the maximum $Q$ value. That is, if the estimate of transient service reward, $UR_t$, is closed to the real optimal solution, $UR^*_\infty$, it can be concluded that the resulting $Q$-learning can provide the similar effect on service reward to the best (but unknown) control limit policy. In Figure 6, we carry out the simulation and compare the $Q$-learning algorithm with the rejuvenation schedule with fixed threshold level $N^* (= 2, 6)$. In the same parametric circumstance, we plot...
the behaviour of estimates of $UR_t$. In this simulation experiment, since it is shown analytically that $N^* = 2$ is optimal, it can be easily expected that the simulation result with $N^* = 2$ approaches to $UR^*_\infty = 0.708471$. On the other hand, the rejuvenation schedule based on the $Q$-learning gives the fluctuated results in earlier phase, and latter converges to the real optimal as the number of processes increases. It is evident that the simulation-based algorithm used here can never outperform the really optimal rejuvenation policy, say, $N^* = 2$. However, in the situation where no statistical information on the system deterioration/failure time distributions is available, this non-parametric estimation scheme would be useful. Table 4 presents the estimation error of expected service reward in the steady state, where the learning rate remarkably depends on the estimation accuracy. Figure 7 illustrates the dependence of learning rate on the transient service reward $UR_t$, where the estimate of service reward with smaller learning rate approaches to the expected service reward in the steady state $UR_\infty$, as the number of processes increases. In Table 5, we calculate the estimation errors (%) between estimates of transient service reward and the real optimal with high accuracy, where $|UR_\infty - UR_t|$. As the learning rate decreases, the estimation error decreases and afterward the $Q$-learning tends to estimate the service reward in the steady state.

**Table 4** Dependence of learning rate on estimation of expected service reward in the steady state with $Q$-learning

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>200</th>
<th>400</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>1.96%</td>
<td>1.11%</td>
<td>0.56%</td>
</tr>
<tr>
<td>0.05</td>
<td>1.99%</td>
<td>1.36%</td>
<td>0.74%</td>
</tr>
<tr>
<td>0.10</td>
<td>2.02%</td>
<td>1.58%</td>
<td>1.03%</td>
</tr>
<tr>
<td>0.20</td>
<td>2.07%</td>
<td>1.74%</td>
<td>1.41%</td>
</tr>
<tr>
<td>0.30</td>
<td>2.19%</td>
<td>1.88%</td>
<td>1.48%</td>
</tr>
<tr>
<td>0.40</td>
<td>2.33%</td>
<td>2.01%</td>
<td>1.89%</td>
</tr>
<tr>
<td>0.50</td>
<td>2.60%</td>
<td>2.22%</td>
<td>2.05%</td>
</tr>
</tbody>
</table>

**Figure 6** Comparison of $Q$-learning policy with some control-limit ones in simulation (see online version for colours)
Figure 7 Asymptotic behaviour of transient service reward based on $Q$-learning for varying learning rate (see online version for colours)

Table 5 Dependence of learning rate on estimation of transient service reward with $Q$-learning

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Number of processes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>200</td>
</tr>
<tr>
<td>0.01</td>
<td>4.29%</td>
</tr>
<tr>
<td>0.05</td>
<td>5.33%</td>
</tr>
<tr>
<td>0.10</td>
<td>6.94%</td>
</tr>
<tr>
<td>0.20</td>
<td>8.03%</td>
</tr>
<tr>
<td>0.30</td>
<td>9.66%</td>
</tr>
<tr>
<td>0.40</td>
<td>10.98%</td>
</tr>
<tr>
<td>0.50</td>
<td>12.50%</td>
</tr>
</tbody>
</table>

6 Conclusions

In this paper, we have developed an online adaptive algorithm to estimate the optimal software rejuvenation policy, based on the $Q$-learning and investigated its asymptotic properties. The resulting algorithm is a non-parametric estimation one whose estimator is asymptotically optimal. The result can be applied to an estimation problem such as a preventive maintenance problem with garbage collection for an application software, if the degradation level can be quantified by the total amount of memory leak and the temporal behaviour is modelled by a SMP. In fact, it is not so difficult to implement the $Q$-learning algorithm on real computer systems like the web server.

In future, the improvement of the $Q$-learning in terms of the convergence speed of $z_t$ to the steady-state solution should be studied. Then, the other type of $Q$-learning algorithm called actor-critic algorithm (Konda and Borkar, 1999) would be considered as an alternative one. Also, it is well known that the value iteration algorithm for the semi-Markov decision processes is simple but does not often function well in some cases. Then, the other type of $Q$-learning based on the policy iteration should be examined for the online estimation of the optimal software rejuvenation scheduling.
Availability and performability analysis

References