A Experimental Results

A.1 Empirical results for Proposition 5.1

We evaluated the performance of lazy-agents trained in environments with random delays, where $o_{\max} = 10$, and compared them to normal agents trained in those with constant delays, where $o = o_{\max}$, as illustrated in Fig. 5. The empirical results demonstrate that the performance of lazy-agents is comparable to that of normal agents. This supports our argument that RDMDPs can be transformed into equivalent CDMDPs through our lazy-agents, thereby enabling conventional methods designed for handling constant delays to be naturally extended to environments with random delays.

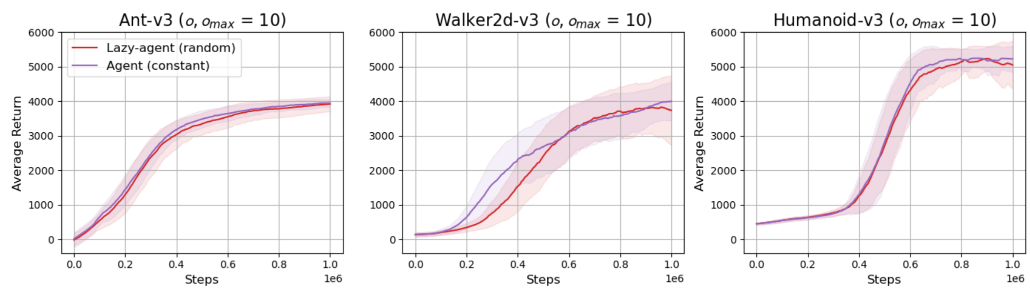


Figure 5: Performance curves of lazy-agents trained in environments with random delays of $o_{\max} = 10$ and normal agents trained in environments with constant delays of $o = o_{\max}$ for continuous control tasks in the MuJoCo benchmark. All tasks were conducted with five different seeds for one million time-steps. The shaded regions represent the standard deviation of average returns.

A.2 Plots of Performance comparison

In this section, we present the performance curves of each algorithm on the MuJoCo tasks with random delays of $o_{\max} \in \{5, 10, 20\}$. All tasks were conducted with five different seeds for one million time-steps. The shaded regions represent the standard deviation of average returns.

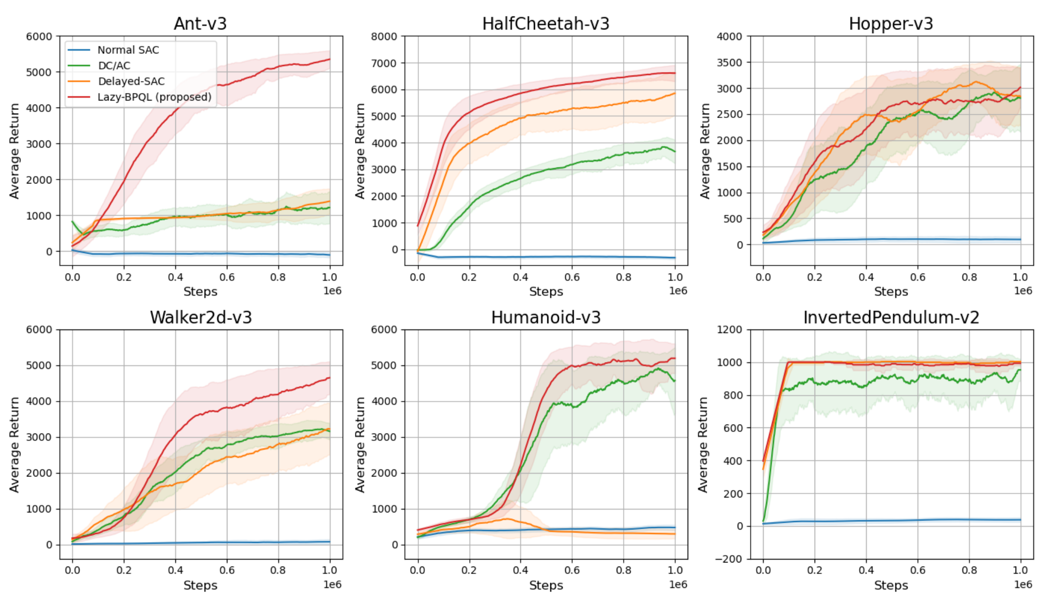


Figure 6: Performance curves of each algorithm on the MuJoCo tasks with $o_{\max} = 5$.

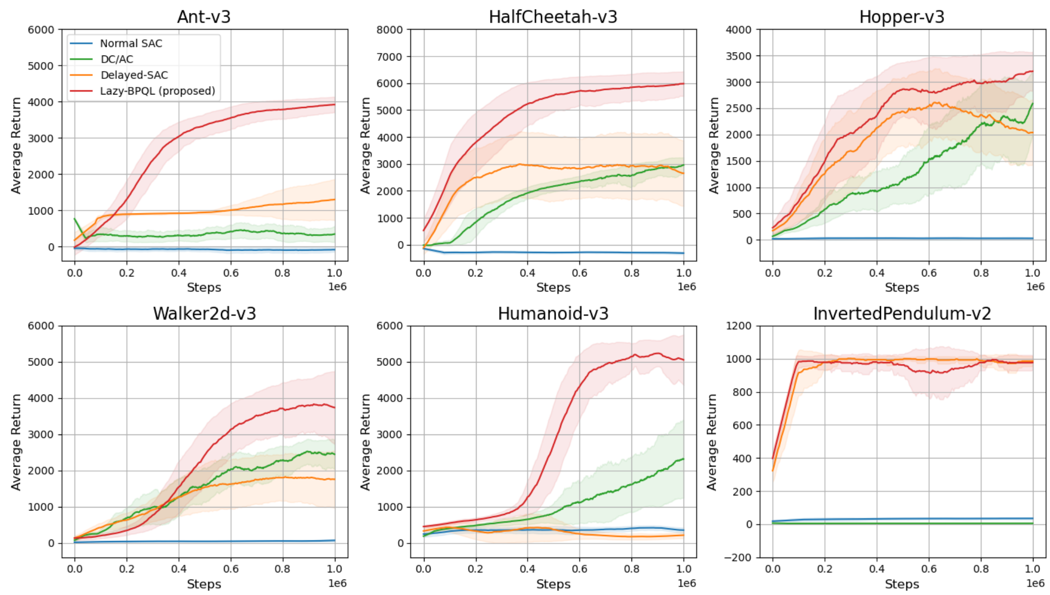


Figure 7: Performance curves of each algorithm on the MuJoCo tasks with $o_{\max} = 10$.

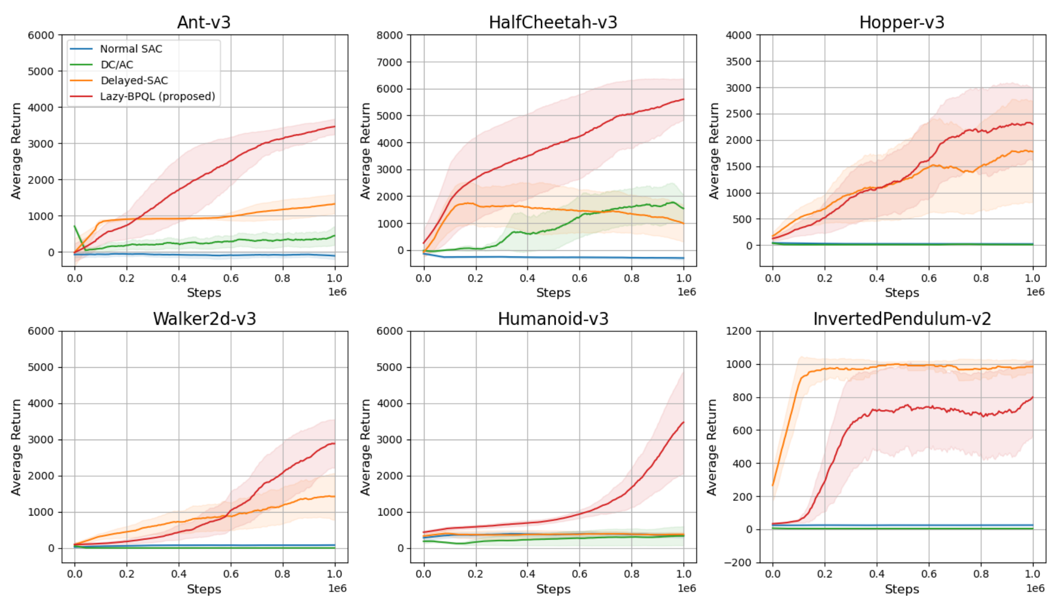


Figure 8: Performance curves of each algorithm on the MuJoCo tasks with $o_{\max} = 20$.

A.3 STATE-SPACE EXPLOSION ISSUE

In this section, we present the performance curves of lazy-augmented-SAC and lazy-BPQL on the MuJoCo tasks with random delays of $o_{\max} \in \{5, 10, 20\}$. As shown in Fig. 9, the proposed lazy-BPQL outperforms lazy-augmented-SAC in terms of asymptotic performance and sample efficiency. Notably, lazy-augmented-SAC completely fails to learn for the tasks even with the random delay of $o_{\max} = 5$, highlighting the importance of avoiding the state-space explosion issue.

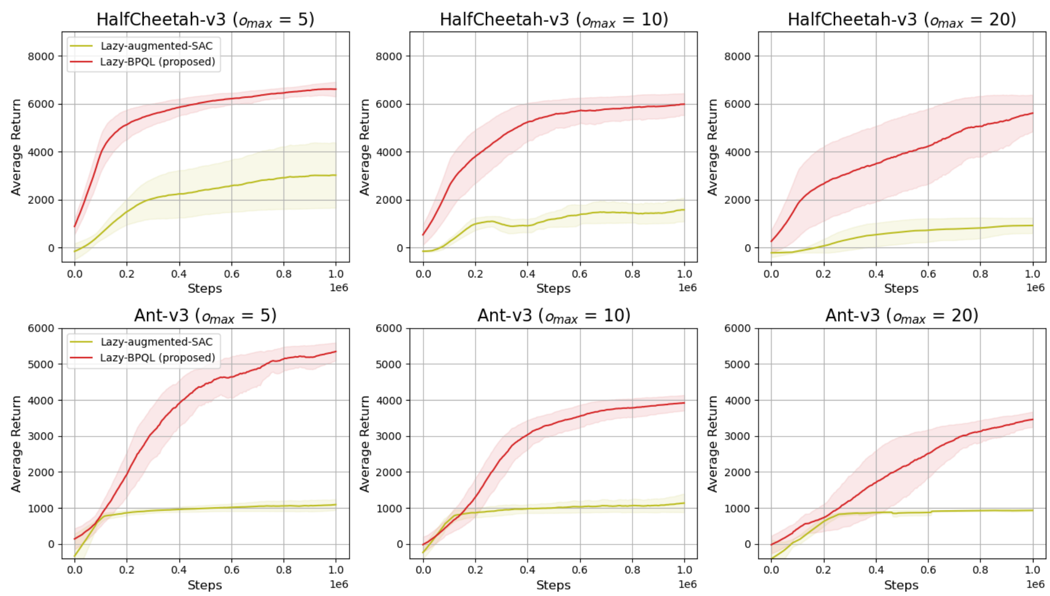


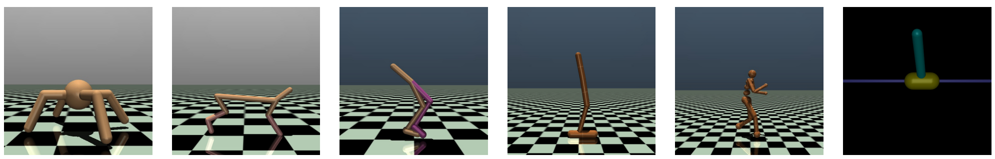
Figure 9: Performance curves of lazy-augmented-SAC and lazy-BPQL for continuous control tasks from the MuJoCo benchmark with random delays of $o_{\max} = \{5, 10, 20\}$. From the results, lazy-BPQL dominates lazy-augmented-SAC, underscoring the importance of avoiding the state-space explosion issue.

B Experimental details

B.1 Environmental details

Table 2: Environmental details of the MuJoCo benchmark.

Task	State dimension	Action dimension	Time-step (s)
Ant-v3	27	8	0.05
HalfCheetah-v3	17	6	0.05
Walker2d-v3	17	6	0.008
Hopper-v3	11	3	0.008
Humanoid-v3	376	17	0.015
InvertedPendulum-v2	4	1	0.04



(a) (b) (c) (d) (e) (f)

Figure 10: Experimental environments in the MuJoCo benchmark: (a) Ant-v3 (b) HalfCheetah-v3, (c) Walker2d-v3, (d) Hopper-v3, (e) Humanoid-v3, and (f) InvertedPendulum-v2

B.2 Implementation details

The implementation details of the proposed lazy-BPQL align with those presented in Kim et al. (2023), with the specific hyperparameters listed in Table 3. Since the baseline algorithms included in our experiments employ the SAC algorithm as their foundational learning algorithm, the hyperparameters are consistent across all approaches, except for the DC/AC algorithm.

Table 3: Hyperparameters for lazy-BPQL and the baselines.

Hyperparameters	Values
Actor network	256, 256
Critic network	256, 256
Learning rate (actor)	3e-4
Learning rate (critic)	3e-4
Temperature (α)	0.2
Discount factor (γ)	0.99
Replay buffer size	1e6
Mini-Batch size	256
Target entropy	-dim$\|\mathcal{A}\|$
Target smoothing coefficient (ξ)	0.995
Optimizer	Adam (Kingma, 2014)
Total time-steps	1e6

B.3 PSEUDO CODE OF LAZY-BPQL

The proposed lazy-agent can be seamlessly integrated into the BPQL framework with minimal modifications by *using* the initial state for decision-making at its maximum delayed times. Subsequently, all states become naturally available for use at their respective maximum delayed times.

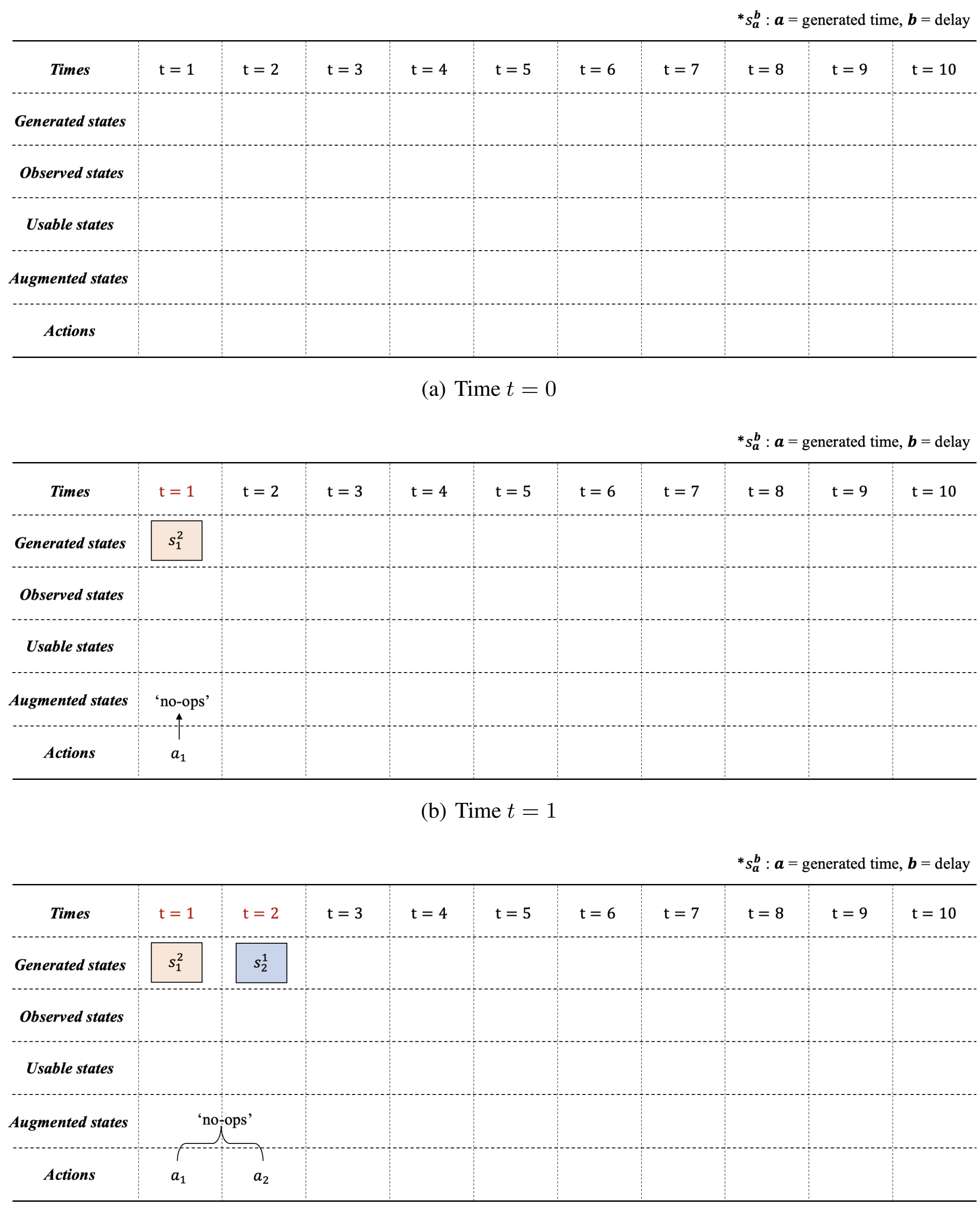
In the implementation, a temporary buffer $\mathcal{B}$ has been employed, as utilized by Kim et al. (2023), to store *observed* states, corresponding rewards, and action histories, which enables the agent to access timely and relevant information for constructing augmented states.

Algorithm 1 Lazy Belief Projection-based Q-Learning (Lazy-BPQL)

Input: actor $\bar{\pi}_\phi(a|\hat{x})$, beta critic $Q_{\theta,\beta}(s,a)$, target beta critic $Q_{\tilde{\theta},\beta}(s,a)$, replay buffer $\mathcal{D}$, temporary buffer $\mathcal{B}$, maximum delay o_{max}, beta critic learning rate λ_Q, actor learning rate $\lambda_{\bar{\pi}}$, soft update rate ξ, episodic length H, and total number of episodes E.
for episode $e = 1$ to E **do**
  for time-step $t = 1$ to H **do**
    if $t < o_{\text{max}}$ **then**
      select random or 'no-ops' action a_t
      execute a_t on environment
      put a_t, observed states, rewards to $\mathcal{B}$
    else if $t = o_{\text{max}}$ **then** ▷ wait for o_{max} time-steps
      select random or 'no-ops' action a_t
      execute a_t on environment
      put a_t, observed states, rewards to $\mathcal{B}$
    else
      get $s_{t-o_{\text{max}}}, a_{t-o_{\text{max}}}, ..., a_{t-1}$ from $\mathcal{B}$
      ▷ get most recent *usable* state and action histories
      $\hat{x}_t \leftarrow (s_{t-o_{\text{max}}}, a_{t-o_{\text{max}}}, ..., a_{t-1})$ ▷ construct augmented state
      $a_t \leftarrow \bar{\pi}_\phi(\hat{x}_t)$
      execute a_t on environment
      put a_t, observed states, rewards to $\mathcal{B}$
      if $t > 2o_{\text{max}}$ **then**
        get $s_{t-2o_{\text{max}}}, s_{t-2o_{\text{max}}+1}, s_{t-o_{\text{max}}}, r_{t-o_{\text{max}}}, a_{t-2o_{\text{max}}}, ..., a_{t-o_{\text{max}}}$ from $\mathcal{B}$
        $\hat{x}_{t-o_{\text{max}}} \leftarrow (s_{t-2o_{\text{max}}}, a_{t-2o_{\text{max}}}, ..., a_{t-o_{\text{max}}})$
        $\hat{x}_{t-o_{\text{max}}+1} \leftarrow (s_{t-2o_{\text{max}}+1}, a_{t-2o_{\text{max}}+1}, ..., a_{t-o_{\text{max}}+1})$
        store $(\hat{x}_{t-o_{\text{max}}}, s_{t-o_{\text{max}}}, a_{t-o_{\text{max}}}, r_{t-o_{\text{max}}}, \hat{x}_{t-o_{\text{max}}+1}, s_{t-o_{\text{max}}+1})$ in $\mathcal{D}$
        pop $s_{t-2o_{\text{max}}}, a_{t-2o_{\text{max}}}$ from $\mathcal{B}$
      end if
    end if
  end for
  for each gradient step **do**
    $\theta \leftarrow \theta - \lambda_Q \nabla \mathcal{J}_{Q_\beta}(\theta)$ ▷ update beta critic
    $\phi \leftarrow \phi - \lambda_{\bar{\pi}} \nabla \mathcal{J}_{\bar{\pi}}(\phi)$ ▷ update actor
    $\tilde{\theta} \leftarrow \xi\theta + (1-\xi)\tilde{\theta}$ ▷ update target beta critic
  end for
end for
Output: actor $\bar{\pi}_\phi$

C VISUAL REPRESENTATION OF LAZY-AGENT

In this section, we provide a visual representation of the proposed lazy-agent employed in RDMDPs, where the maximum delay is set to $o_{\max} = 3$.

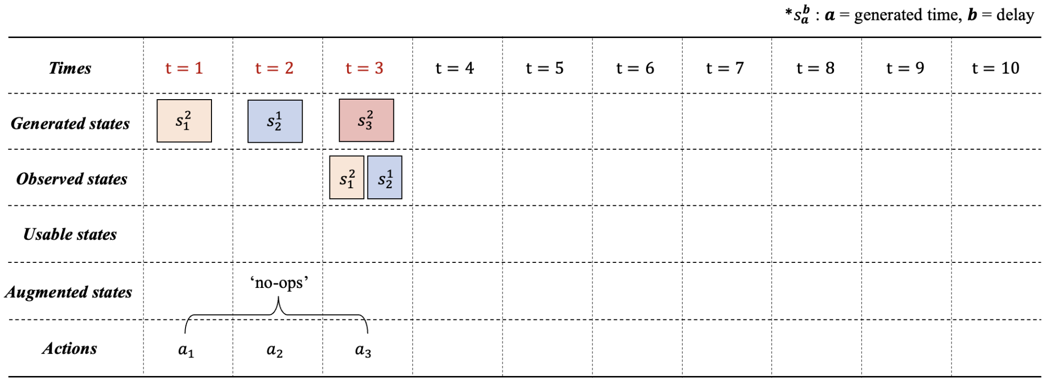


(a) Time $t = 0$

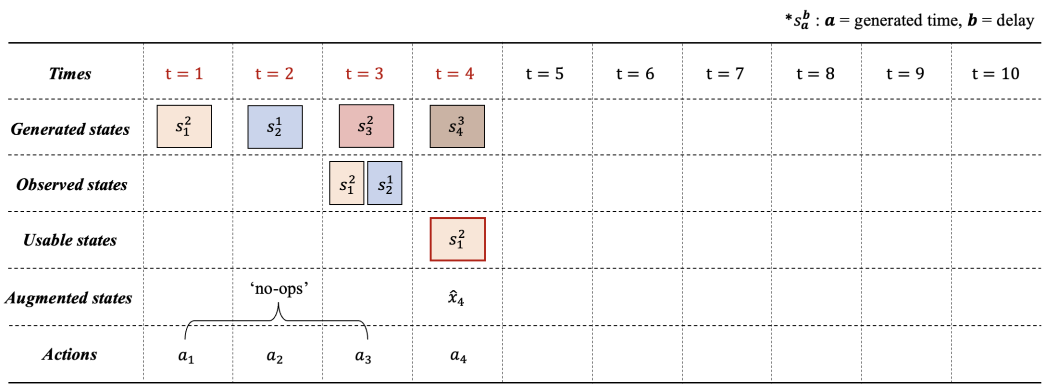
(b) Time $t = 1$

(c) Time $t = 2$

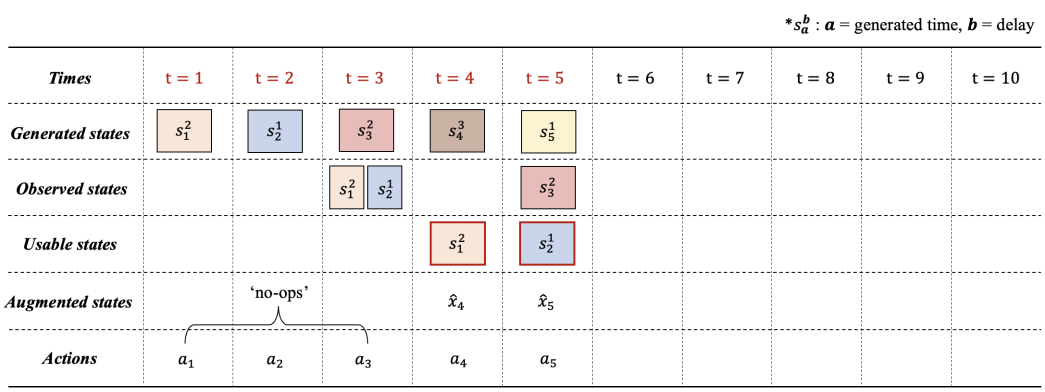
Figure 11: At times 1 and 2, the states s_1^2 and s_2^1 are generated but remain unobserved by the lazy-agent due to delays. In this scenario, the lazy-agent does nothing ('no-ops') until the initial state s_1^2 becomes usable.



(a) Time $t = 3$

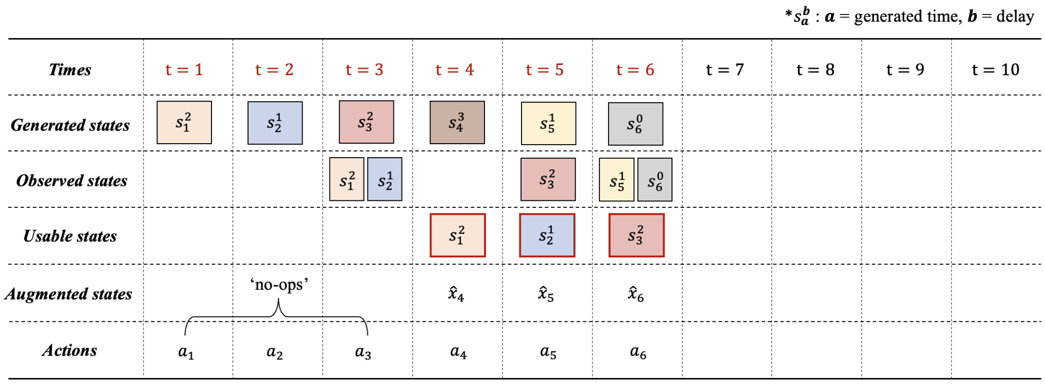


(b) Time $t = 4$

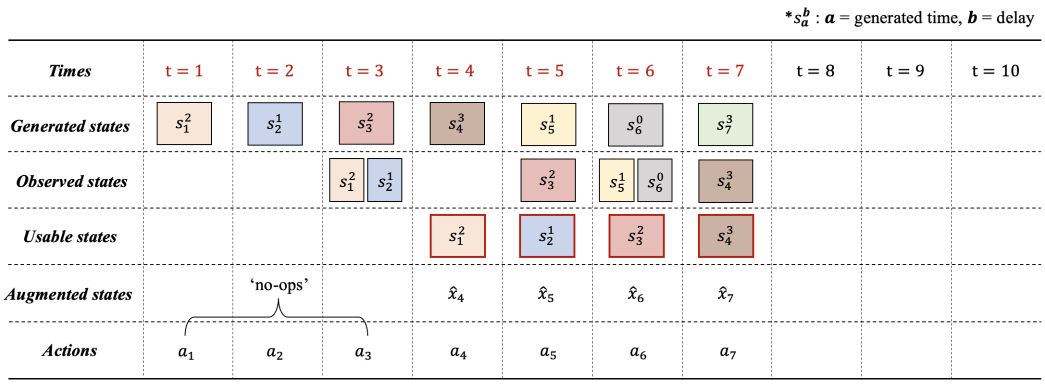


(c) Time $t = 5$

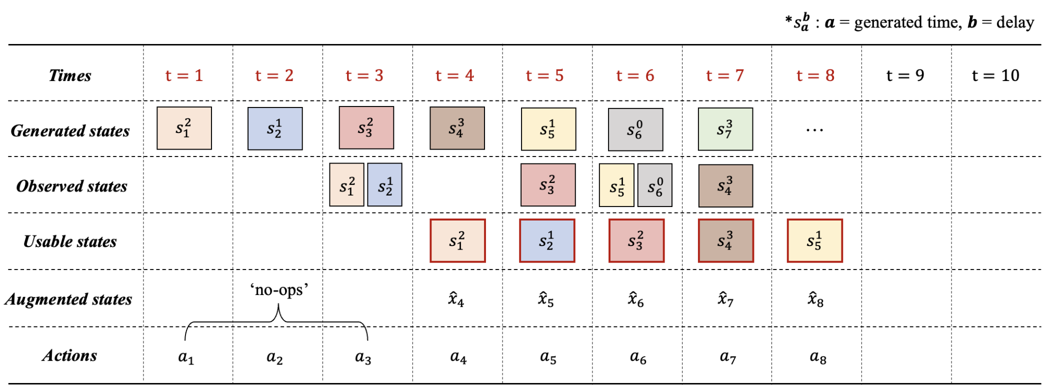
Figure 12: At time 3, states s_1^2 and s_2^1 are observed simultaneously. As the lazy-agent uses these observed states at their maximum delayed times, s_1^2 is used at time 4 and s_2^1 is used at time 5. These states are reformulated as augmented states before being fed into the policy, thereafter determining the appropriate actions. States s_3^2, s_4^3, and s_5^1 are generated at corresponding times, with s_3^2 being observed at time 5.



(a) Time $t = 6$

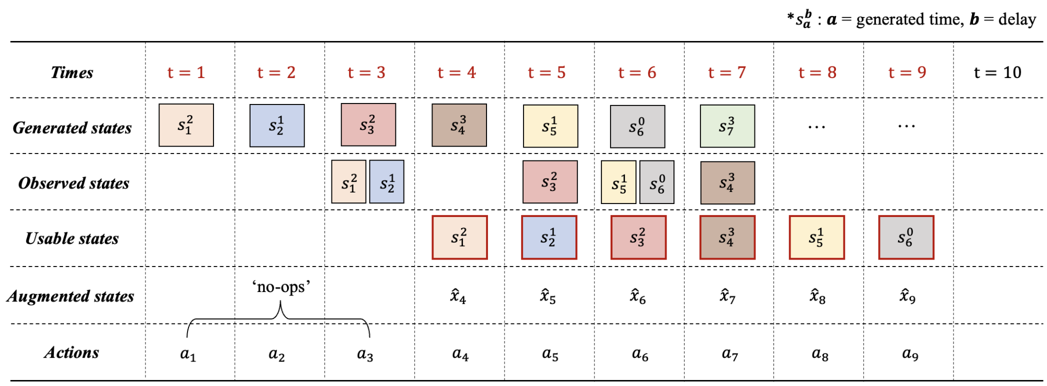


(b) Time $t = 7$

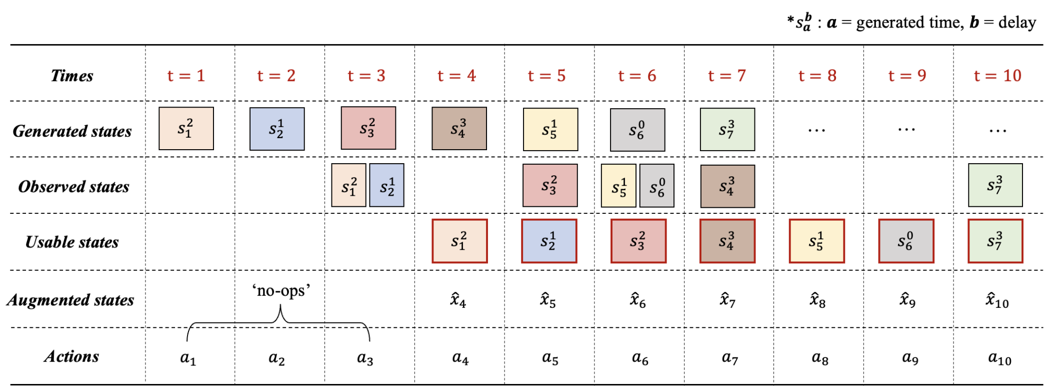


(c) Time $t = 8$

Figure 13: States s_6^0 and s_7^3 are generated at respective times. At time 6, states s_5^1 and s_6^0 are observed simultaneously but are not immediately usable because the previously generated states, s_3^2 and s_4^3, have not yet been used in decision-making processes. Instead, s_3^2 is used at this time. At time 7, state s_4^3 is observed and is available for use immediately. At time 8, state s_5^1 becomes usable, as all previously generated states have now been both observed and used.



(a) Time $t = 9$



(b) Time $t = 10$

Figure 14: At times 9 and 10, states s_6^0 and s_7^3 are used in sequence. Despite the state observations occurring simultaneously or being out of order, all the delayed states are consistently used in sequence at their maximum delayed times, i.e., $\tau(s_n^{o_n}) = n + o_{\max}, \forall n > 0$.