Novel time aggregation based algorithms for Markov Decision Processes

João Marcelo L. G. Leite, Lino G. Marujo and Edilson F. Arruda

***Abstract*—We present two novel approaches to accelerate the convergence of a class of Markov decision problems (MDPs) with stationary state space components, which iterate on reduced state and action spaces and converge to an optimal policy. The time aggregation based algorithm (TABA) partitions the state space into disjoint sets, one for each possible combination of the non-stationary components, and iterates in the sets. The local search policy set iteration (LSPSI) algorithm introduces a novel modified policy evaluation procedure that seamlessly performs a local search over a very large set of candidate policies, by sampling a reduced subset of actions at each state's value function update. Both approaches, as well as a combination of them, are validated by means of a mining supply chain application with a large number of stationary state components and a large set of feasible actions. The experiments suggest that the proposed frameworks are very efficient for such a class of problems.**

***Index Terms*—Markov decision processes, Stochastic systems, Optimization under uncertainty, Supply Chains**

I. INTRODUCTION

There are multiple stochastic optimisation frameworks devised for specific problem classes [1]. Sequential decision problems under uncertainty are particularly relevant, comprising a sequence of decision periods, feasible control actions, and costs incurred in an uncertain environment [2]–[4]. At each period and depending on the system's state, a decision incurs a cost and triggers a probabilistic transition to another state, where another decision re-initiates the loop. The aim is to find an optimal control sequence with respect to a performance criterion over a typically long horizon. Markov decision processes [5] are the modelling framework, leading to dynamic programming (DP) algorithms that find an optimal policy whilst implicitly considering all possible uncertain scenarios in the long run [2, 6]. Classical value (VI) and policy (PI) iteration are arguably the most utilised DP algorithms.

The computational complexity of DP algorithms is a function of the number of states and control actions [7, 8]. However, complex problems require increasingly large numbers of states and actions, effectively rendering moderately complex problems virtually intractable. Termed *curse of dimensionality*, this phenomenon has motivated a large body of literature. To reduce the computational complexity and accelerate convergence, one can utilise a series of increasingly accurate and computationally more tractable approximate models [9], apply sampling techniques to the VI and PI algorithms [10, 11], optimise the computational performance of specific steps of the classical algorithms [12] or build a graph starting from an initial state and, taking advantage of the problem's topology, deploy DP only in the visited states [13]. While these techniques do improve performance, the curse of dimensionality remains an open problem in the literature.

Using approximations is an intuitive and widely utilised alternative to counter dimensionality. Designed to tackle problems with an incomplete knowledge of the transition probabilities, reinforcement learning (RL) algorithms exploit stochastic approximation concepts [14] to learn the system's dynamics whilst searching for the optimal solution [4, 15]. The classical Q-learning and *SARSA* algorithms employ temporal difference learning to update the value of taking an action at some state, considering the observed rewards and future states [16]. Variations of these algorithms have been combined and amply utilised [17]. Approximate dynamic programming (ADP) is another influential approach that uses approximate functions to estimate state values and near-optimal policies. Proposed in the early years of the field [18], it has recently attracted attention in the literature with the approximate value iteration (AVI) and the least squares policy iteration algorithms [19, 20]. While successful in some classes of problems, these approaches may face convergence issues and their effectiveness hinges on the quality of the proposed approximation scheme.

Reducing the state space's size by aggregating states [21]–[24] can mitigate the curse of dimensionality, but simple state aggregation does not mpreserve the Markov property and effectively solves an alternative problem [25]. Proposed to find an equivalent embedded formulation when no control choices exist for the majority of states, time aggregation reduces the state space's size whilst retaining the Markov property [25]. The approach was later extended to general MDPs via a two-phase algorithm [26, 27]. The first phase keeps a constant control policy in the bulk of the state space whilst optimising the performance in a comparatively small subset of states. The second step applies policy improvement to produce a better policy for the next iteration of the first phase; the two phases alternate up to convergence. Despite the performance gains, the approach is limited by the size of the state space in the policy improvement step. To circumvent this issue, [28] proposed a generalised decomposition in multiple subsets of reduced size to calculate the steady state of a Markov chain. The approach, however, does not include optimisation.

This paper proposes a combination of two algorithms for large-scale MDPs. The first is a novel multi-cluster time aggregation algorithm, called *time aggregation based algorithm (TABA)*. Designed for a class of Markov decision problems with stationary state components, here called *semi stationary MDPs* (SSMDP), TABA uses time aggregation to divide the state space into multiple subsets and utilises properties of the stationary components to accelerate convergence. One of TABA's novel contributions is that it uses the properties of SSMDPs to seamlessly derive a multi-subset partition of the state space that can be solved directly, in a single stage, without the need to alternate policy evaluation and improvement steps as in [26]. TABA extends the multi-subset partitioning approach in [28] by introducing optimisation. Unlike state aggregation approaches, TABA maintains the Markov property.

The second algorithm is called *local search policy set iteration (LSPSI)*, and its novel contribution is a powerful local search step within a policy iteration framework. Inspired by *policy set iteration (PSI)* [10], which samples a number (N) of policies per iteration, LSPI optimises over an exponential number of candidate policies in one iteration by sampling a fixed proportion of the action space in each state. If n actions are sampled per state in the state space S, the local search will cover $n^{|S|}$ feasible policies, thereby accelerating convergence.

J. M. L. G. Leite is with Instituto de Ensino e Pesquisa - INSPER, São Paulo, Brazil (e-mail: joaomlgl@insper.edu.br).

L. G. Marujo is with Industrial Engineering Program, Alberto Luiz Coimbra Institute – Graduate School and Research in Engineering, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil (e-mail: lgmarujo@ufrj.br).

E. F. Arruda is with Southampton Business School, University of Southampton, Building 175, Boldrewood Innovation Campus, Burgess Road, Southampton, SO16 7QF (e-mail: e.f.arruda@southampton.ac.uk).

Finally, as TABA and LSPSI act on different aspects of the *curse of dimensionality* - the size of the state space and the size of the action space - we propose a novel combined algorithm termed *(TABA+LSPSI)*. To validate the approach, we explore an interesting and complex mining supply chain problem. The field has evolved from classical open pit mine design and optimisation [29]–[31] to approaches that analyse the full operation across the entire mining supply chain [32]–[34]. Specifically, the case study explores the novel application of an MDP formulation to the whole logistics network, from mine to client, involving complex configurations of the state and action spaces, as well as uncertainties spread over the supply chain. The numerical experiments illustrate the strengths of the proposed algorithms. Both TABA and LSPSI significantly reduce the convergence time with respect to value and policy iteration; they are also considerably faster than policy set iteration (PSI). Finally, the combination of TABA and LSPSI produces the best results over all experiments. Indeed, (TABA+LSPSI) converges in a matter of seconds in all experiments and the performance is relatively insensitive to increasing discount factors.

The remainder of this work is organised as follows. Section II introduces Markov decision processes and the optimality conditions. Section III defines semi-stationary Markov decision processes (SSMDP), introduces the proposed algorithms, and discusses their convergence. Section IV utilises a mining supply chain problem to validate the algorithms and compare their results to value iteration, policy iteration and policy set iteration [10]. Finally, Section V concludes the paper.

II. Mathematical model

Consider a discrete-time dynamic system with time index $t \geq 0$ and discrete, finite state space S. At each period $t \geq 0$, the decision-maker observes the environment's *state* $s_t = s \in S$ and selects an *action* $a_t = a$ from the discrete, finite set $A(s)$ of possible actions in s. The *action space* $A = \bigcup_{s \in S} A(s)$ is the set of all available actions. The system then experiences a probabilistic transition to some state $s_{t+1} = s' \in S$, which happens with probability $p(s'|s,a)$. Since $p : S \times A \times S \to [0,1]$ is an indexed transition probability function, it follows that $\sum_{s' \in S} p(s'|s,a) = 1,\ \forall a \in A(s)$.

At each step, the decision maker observes the state s and selects an action $a \in A(s)$, whereupon the system incurs a reward $r(s,a)$ determined by the reward function $r : S \times A \to \mathbb{R}_+$, bounded from below and above. To select an action, the decision maker uses a *decision rule* d_t at $t \geq 0$. This paper considers the class of *stationary control policies* Π that comprises policies $\pi = \{d, d, \ldots\}$ and $\pi_t(s) = \pi(s) = d(s)$, which prescribe the same decision rule $d_t = d, \forall t \geq 0$. Starting from state $s_0 = s$, the long-term reward of policy $\pi \in \Pi$ is:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left\{\sum_{t=0}^{+\infty} \lambda^t r(s_t, \pi(s_t)) | s_0 = s\right\}, \forall s \in S,$$

where $0 < \lambda < 1$ is a discount factor, used to convert future rewards into net present value [35]. Henceforth, we will replace s_t and s_{t+1} by s and s', respectively. This is to avoid confusion with s_j, where j will indicate the dimension, which will be used in Section III.

The decision maker seeks a policy $\pi^* \in \Pi$ that maximises the long-term reward from each state $s \in S$ and yields

$$V^*(s) = V^{\pi^*}(s) = \max_{\pi \in \Pi} V^{\pi}(s), \forall s \in S. \tag{1}$$

The existence of a policy satisfying (1) is guaranteed as the rewards are bounded and do not vary in time, the state (S) and action (A) spaces are discrete and finite, and the indexed transition probabilities ($p(s'|s,a)$) are stationary, i.e., do not vary over time [2, 3].

A. Optimality equations

To solve (1) we need the value function $V^* : S \to \mathbb{R}$. Classical results [2, 3] yield that:

$$V^*(s) = \max_{\pi \in \Pi}\left[r(s,\pi(s)) + \lambda \sum_{s' \in S} p(s'|s,\pi(s)) V^*(s')\right], \tag{2}$$

where:

$$\pi^*(s) = \arg\max_{\pi \in \Pi}\left[r(s,\pi(s)) + \lambda \sum_{s' \in S} p(s'|s,\pi(s)) V^*(s')\right]. \tag{3}$$

Value iteration (VI) is arguably the most utilised procedure to find the common solution of (1) and (2). It iterates on the space of real-valued functions $\mathcal{V} : S \to \mathbb{R}$, starting from $V_0 \in \mathcal{V}$. It uses following value function update:

$$V_{t+1}(s) = TV_t(s) := \max_{a \in A(s)}\left\{r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) V_t(s')\right\} \tag{4}$$

Under the same conditions that ensure the existence of a solution to (1), see Section II, value iteration converges linearly and monotonically V^* [2, 3], with convergence rate λ. Like policy iteration [2, 3], it is very efficient, but rendered intractable for moderate problem dimensions due to the curse of dimensionality [19].

In the next section, we will introduce two new algorithms to mitigate this effect for a class of MDPs with partly stationary exogenous uncertainty. They use time aggregation [25, 26] and policy set manipulation [10] to exploit the problem structure and accelerate convergence, and can be applied to real-world problems such as iron-ore logistics networks, see Section IV.

III. Time-aggregation and policy set manipulation based approaches to large-scale MDPs

Consider an MDP wherein each state $s \in S$ is multi-dimensional (belongs to $\Re^I$), with $s_j,\ j \in \{1, \ldots, I\}$ denoting the state's value in dimension j, and $1 < I < \infty$ corresponding to the number of dimensions. Assume that $K \leq I$ dimensions follow a traditional MDP, in which the state transition depends on the state-action pair. The other $I - K$ dimensions evolve according to a stationary stochastic process that is not influenced by the system's state or the control policy. Without loss of generality, we can simply relabel dimensions appropriately if necessary, so that these are the last $I - K$ components of the state. This gives rise to Assumption 1.

Assumption 1: Let $S = S_c \times S_n$, where S_c is the space of the first K components of the state and S_n corresponds to the remaining components. Assume that

$$p(s'|s,a) = p(s'_1, \ldots, s'_K|s,a) P(\xi = (s'_{K+1}, \ldots, s'_I)), \tag{5}$$

where ξ is a random variable with sample space S_n.

Assumption 1 is typical in some real-world problems. It holds, for example, in inventory management problems for complex supply chains. The first K components correspond to storage levels, which depend on previous states and actions. On the other hand, contracted demand volumes and market prices change every period according to a prescribed probability distribution that does not depend on previous states and actions. Hence, they can be modelled as the last $I - K$ elements. Typically, their realization needs to be known before making a new decision, therefore they must be included in the system's state. Another area where Assumption 1 is frequently applied is wireless communication. The transition probabilities depend only on the previously known error probabilities, and

are independent of the state (age-transmission) and the action (re-transmission or generation) [36, 37].

Definition 1: A semi-stationary Markov decision process (SSMDP) is an MDP for which Assumption 1 holds.

We will now introduce novel time-aggregation-based algorithms that are especially efficient for SSMDPs.

A. Time-aggregation-based algorithm (TABA)

Consider a state space (S) partition into Z disjoint sets (G_z):

$$S = \bigcup_{z=1}^{Z} G_z, \quad G_i \cap G_j = \emptyset, \forall\ i, j \leq Z \text{ such that } i \neq j.$$

For each G_z let us define:

- $\overline{V}^{G_z}$: a vector with the current value function estimate of each state $s \in G_z$. Its cardinality is $|G_z|$:

$$\overline{V}^{G_z} = [V(s)], s \in G_z.$$

- μ_z: the normalised steady-state distribution in G_z

$$\mu_z = [\mu_z(s)], s \in G_z, \sum_{s \in G_z} \mu_z(s) = 1.$$

- $p_G(G'_z|s,a)$: the probability of reaching subset G'_z from state s under control action a

$$p_G(G'_z|s,a) = \sum_{s' \in G'_z} p(s'|s,a). \tag{6}$$

Assuming that μ_z is known for all subsets G_z in the partition, it follows that

$$V(G_z) = \mu_z \cdot \overline{V}^{G_z} \tag{7}$$

is the expected value function estimate at subset G_z. We can then reformulate the value iteration update in (4) as follows:

$$TV_t(s) = \max_{a \in A(s)} \left\{ r(s,a) + \lambda \sum_{z=1}^{Z} p_G(G'_z|s,a) V_t(G'_z) \right\}. \tag{8}$$

Algorithm 1 is TABA's pseudo-code. Observe that the computational cost of Eq. (8) is proportional to the number of subsets in the state space partition, given by Z. Hence, replacing (4) with (8) in the value iteration algorithm can produce significant computational savings when $Z \ll |S|$. This is the motivation for the algorithm.

One difficulty with the value function update in (8) is that it assumes that the steady state probability μ_z is known across all subsets $G_z \in S$. While this distribution is a function of optimal policy π^* in a general MDP setting, we can exploit the structure of SSMDPs in Definition 1 to derive an appropriate partition whereby μ_z can be easily attained. For SSMDPs, Assumption 1 suggests a straightforward partition whereby each subset corresponds to a unique combination of the first K components of the state:

$$G_z = \{s \in S : \{s_1, \ldots, s_K\} = s_z \in S_c \text{ and } z \in \{1, \ldots, |S_c|\}.$$

Using this suggested partition, the steady-state distribution within subset G_z can be calculated as:

$$\mu_z = [\mu_z(s_n)] = [P(\xi = s_n)],\ \forall\ s_n \in S_n,$$

where $s_n \in S_n$ and ξ is a random variable with known probability distribution.

Therefore, TABA iterates across all possible combinations of the first K components of the state space, as each gives rise to a subset G_z. The computational gains happen because, for each subset G_z, we need not consider each combination of the last $I - K$ components individually. Instead, for each subset G_z, we use the expected value given by Eq. (7), which depends on the steady-state distribution μ_z over the set of states $\{s_{K+1}, \ldots, s_I\} = s_n \in S_n$. Note that Assumption 1 implies that μ_z is independent of the control policy, does not vary over time, and can be calculated a priori. These properties enable us to use (8) in the value function update step.

Algorithm 1: TABA

Input: An arbitrary initial solution $V_0 \in \mathcal{V}$, a state space (S) partition into Z disjoint sets (G_z), the normalised steady-state distribution within subset G_z (μ_z), an arbitrary discount factor $\lambda \in [0, 1)$ and an arbitrary tolerance ϵ

Output: Optimal solution V^*, and optimal stationary policy π^*

$t \leftarrow 0$;

$V_t(G_z) \leftarrow \sum_{s \in G_z} \mu_z(s) V_t(s)\ \forall z = \{1, 2, \cdots, Z\}$;

repeat

  for *each* G_z **do**

    $$V_t(G_z) := \sum_{s \in G_z} \mu_z(s) V_t(s)$$

  end

  for *each* $s \in S$ **do**

    $$TV_t(s) := \max_{a \in A(s)} \left\{ r(s,a) + \lambda \sum_{z=1}^{Z} p_G(G'_z|s,a) V_t(G'_z) \right\}$$

    $$V_{t+1}(s) := TV_t(s)$$

  end

  $t \leftarrow t + 1$;

until $\|V_t - V_{t-1}\|_\infty \leq \epsilon$;

$V^* \leftarrow V_t$;

$$\pi^*(s) \leftarrow \arg\max_{a \in A(s)} \left[r(s,a) + \lambda \sum_{z=1}^{Z} p_G(G'_z|s,a) V_t(G'_z) \right], \forall s \in S;$$

return V^*, π^*.

A preliminary version of this algorithm [34] utilised a more complicated and less efficient value function update:

$$v_*(s) \underset{s \in G_{\mathbb{S}}}{=} \max_{\pi} \Bigg\{ \sum_{s' \in G_{\mathbb{S}}} p_d(s'|s,\pi(s)) \Big\{ r(s,\pi(s),s') + \lambda v_*^{t+1}(s') \Big\} + \sum_{z \neq \mathbb{S}} p_G(G'_z|s,\pi(s)) \Big\{ r(s,\pi(s),G'_z) + \lambda V^{t+1}(G'_z) \Big\} \Bigg\} \tag{9}$$

that differentiated states from the same subset $G_{\mathbb{S}}$ and states from other subsets in $S \backslash G_{\mathbb{S}}$. By realising that (9) is equivalent to (8), we are able to not only simplify the value function update, but also increase computational efficiency. Next, we prove the convergence of Algorithm 1.

Lemma 1: Under the proposed partitioning scheme, Equation (8) in Algorithm 1 is equivalent Eq. (4) in VI Algorithm for an SSMDP.

Proof 1: Substituting (7) into (8) yields:

$$TV_t(s) := \max_{a \in A(s)} \left\{ r(s,a) + \lambda \sum_{z=1}^{Z} p_G(G'_z|s,a) \mu_z \overline{V}_t^{G'_z} \right\}$$

$$= \max_{a \in A(s)} \left\{ r(s,a) + \lambda \sum_{z=1}^{Z} p_G(G'_z|s,a) \sum_{s' \in G_z} \mu_z(s') V_t(s') \right\}$$

$$= \max_{a \in A(s)} \left\{ r(s,a) + \lambda \sum_{z=1}^{Z} \sum_{s' \in G_z} p_G(G'_z|s,a) \mu_z(s') V_t(s') \right\}.$$

Now, by realising that $\mu_z(s') = P(\xi = s')$ in Eq. (5) and by substituting (6) into the last expression, we obtain:

$$TV_t(s) = \max_{a \in A(s)} \left\{ r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) V_t(s') \right\}.$$

We conclude the proof by noting that the expression above is equal to Eq. (4).

Theorem 1: Algorithm 1 converges to the solution of (2)-(3).

Proof 2: From Lemma 2, it follows that each iteration of the

TABA algorithm is equivalent to an iteration of the classical value iteration algorithm. Convergence follows because the classical value iteration algorithm has guaranteed convergence to the solution of (2)-(3) [2].

Remark 1: Although the value function update in Algorithm 1 is equivalent to the value iteration update, the computational effort is reduced as TABA uses the Z aggregate values from Eq. (7) instead of the whole vector V_t. That generates significant computational savings when $Z \ll |S|$.

B. Local search policy set iteration (LSPSI)

Based on the value iteration algorithm, TABA exploits the structure of SSMDPs to reduce the computational effort. Similarly, Policy Set iteration (PSI) [10] is designed to accelerate the policy iteration convergence by sampling multiple policies at each iteration and using the maximum value over these policies in the policy improvement step.

In this section, we introduce a novel algorithm inspired by PSI and called *LSPSI*, that takes advantage of the combinatorial nature of the problem to expand the set of policies compared at each iteration. We start with the policy iteration algorithm structure, combining policy evaluation and policy improvement steps. However, instead of using matrix inversion for policy evaluation, we use value iteration, albeit not constrained to a single policy. We use a reduced subset of actions $H(s) \subset A(s)$ for each state $s \in S$, turning the policy evaluation step into a local search over the exponentially large set of all policies produced by the combination of the actions in $H(s)$, for all $s \in S$. We randomly select $H(s)$ to comprise a fixed proportion (α) of all possible actions in s such that:

$$\frac{|H(s)|}{|A(s)|} = \alpha \leq 1.$$

To ensure that the next selected action is at least as good as the last, the algorithm always includes the greedy actions with respect to the last value function update ($a_{t-1}(s)$):

$$D(s) = H(s) \cup a_{t-1}(s). \tag{10}$$

Policy evaluation updates the value function with:

$$TV_{t\_eval}(s) = \max_{a \in D(s)} \left\{ r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) V_{t\_eval}(s') \right\}, \tag{11}$$

where $D(s)$ comes from Eq. (10) at each iteration. Sampling different actions at each iteration ensures that nearly all actions are tried before convergence, whilst reducing each iteration's computational effort. Algorithm 2 details the LSPSI's steps.

Both PSI and LSPSI introduce a policy update opportunity to the policy evaluation step. While PSI evaluates $N = |\Psi|$ policies in parallel at each policy set evaluation step (where Ψ is a set that contains N policies sampled from Π), LSPSI evaluates $n = |D(s)|$ distinct actions for each state $s \in S$ at each value iteration. Hence, LSPSI evaluates $n^{|S|}$ policies at each modified policy evaluation, covering a much larger number of policies. When the modified policy evaluation step converges, the policy obtained will be the best among all possible combinations of the actions in $D(s)$, $s \in S$. Furthermore, actions not contained in the last set $D(s)$ will have been tried before convergence, thus increasing the probability of attaining a near-optimal policy at the modified policy evaluation step.

After the modified policy evaluation step in Algorithm 2, there is a complete policy improvement step which evaluates all possible actions. This step was missing in the preliminary algorithm introduced in [34]:

$$TV_t(s) := \max_{a \in A(s)} \left\{ r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) V_t(s') \right\}. \tag{12}$$

Eq. (12) is equivalent to a policy improvement step in a classical PI algorithm [2]. Since the resulting improved policy is among the policies evaluated in the next modified policy evaluation step (line 3), it is clear that the LSPI algorithm encompasses the PI algorithm. To verify convergence to an optimal policy, it suffices to see that, when LSPI converges, the PI algorithm embedded in it also converges, as the current policy can no longer be improved. Therefore, convergence to the optimal policy follows from Theorem 6.4.6 in [2]. A similar argument is applied in [10] to prove the convergence of PSI to the solution of (1).

Observe that when $\alpha = 0$, $H(s) = \emptyset \, \forall s \in S$ in Algorithm 2. Therefore, only the incumbent policy will be evaluated in the policy evaluation step, rendering LSPI equivalent to PI. Conversely, $\alpha = 1$ implies $H(s) = A(s) \, \forall s \in S$, rendering LSPI equivalent to VI as all actions are considered.

Algorithm 2: LSPSI algorithm

Input: An arbitrary initial solution $V_0 \in \mathcal{V}$, an arbitrary initial policy $\pi_0 \in \Pi$, an arbitrary discount factor $\lambda \in [0, 1)$ and an arbitrary tolerance ϵ

Output: Optimal solution V^*, and optimal stationary policy π^*

1 $t \leftarrow 0$;
2 **repeat**
3 **Modified policy evaluation step:** $t\_eval \leftarrow 0$; $V_{t\_eval} \leftarrow V_t$; $\pi_{t\_eval} \leftarrow \pi_t$;
4 **repeat**
5 **for** *each* $s \in S$ **do**
6 Choose $H(s) \subset A(s)$ at random; $D(s) \leftarrow H(s) \cup \pi_{t\_eval}(s)$

$$TV_{t\_eval}(s) := \max_{a \in D(s)} \left\{ r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) V_{t\_eval}(s') \right\}$$

$$V_{t\_eval+1}(s) := TV_{t\_eval}(s)$$

$$\pi_{t\_eval+1}(s) \leftarrow \arg \max_{a \in D(s)} \left[TV_{t\_eval}(s) \right], \forall s \in S$$

7 **end**
8 $t\_eval \leftarrow t\_eval + 1$;
9 **until** $\|V_{t\_eval} - V_{t\_eval-1}\|_\infty \leq \epsilon$;
10 **Policy update step:** $V_t \leftarrow V_{t\_eval}$;
11 **for** *each* $s \in S$ **do**
12

$$TV_t(s) := \max_{a \in A(s)} \left\{ r(s,a) + \lambda \sum_{s' \in S} p(s'|s,a) V_t(s') \right\}$$

$$V_{t+1}(s) := TV_t(s)$$

$$\pi_{t+1}(s) \leftarrow \arg \max_{a \in A(s)} \left[TV_t(s) \right], \forall s \in S$$

13 **end**
14 $t \leftarrow t + 1$;
15 **until** $\|V_t - V_{t-1}\|_\infty \leq \epsilon$;
16 $V^* \leftarrow V_t$; $\pi^* \leftarrow \pi_t$;
17 **return** V^*, π^*.

C. Combined algorithm: TABA and LSPSI

Finally, we propose an algorithm that combines the complementary properties of TABA (state space reduction) and LSPSI (action space reduction). The new algorithm follows the same steps as Algorithm 2, but instead of the value function update in Eq. (11), it uses:

$$TV_{t\_eval}(s) = \max_{a \in D(s)} \left\{ r(s,a) + \lambda \sum_{z=1}^{Z} p_G(G'_z|s,a) V_{t\_eval}(G'_z) \right\},$$

and the policy update step in Eq. (12) becomes:

$$TV_t(s) = \max_{a \in A(s)} \left\{ r(s,a) + \lambda \sum_{z=1}^{Z} p_G(G'_z|s,a) V_t(G'_z) \right\}.$$

As detailed in Section IV, combining the properties of TABA and LSPSI tends to be very effective to circumvent the *curse of dimensionality* in SSMPDs. Section IV compares the performance of the algorithms in the light of a mining supply chain example.

D. Computational complexity analysis

Compared with traditional MDP algorithms, the computational complexity analysis of the proposed algorithms indicates a way to circumvent the *curse of dimensionality*. It is well known that each VI iteration has a computational complexity $\mathcal{O}(|S|^2 \times |A|)$ [2]. On the other hand, each PI iteration's effort is of $\mathcal{O}((N+m+1)\times(|X|^2\times|A|+|X|^3))$, however the convergence rate is quadratic with respect to the number of policy evaluations [2]. This can abbreviate the overall convergence time. PSI's iteration effort $\mathcal{O}((N+m+1)\times(|X|^2\times|A|+|X|^3))$ exceeds that of PI; the expectation, however, is that increasing the number of evaluated policies will reduce the number of iterations [10].

TABA algorithm uses the VI approach, although it does not traverse all possible future states. Using the SSMDP properties in Eq. (7), Algorithm 1 needs only to evaluate the values of Z disjoint sets. Therefore, each iteration has computational complexity $\mathcal{O}(|S| \times |A| \times Z)$. As $Z \leq |S|$ and the number of iterations is the same as that of VI, TABA converges faster than VI with rate $\dfrac{Z}{|S|}$.

LSPSI evaluates $|D| = \alpha \times |A| + 1$ actions in each policy evaluation iteration. After convergence of the policy evaluation (k iterations) steps, we have a full policy improvement step. Each LSPSI iteration has a computational complexity of $\mathcal{O}(k\times|S|^2\times|D|+|S|^2\times|A|)$. As with PSI, convergence is quadratic and the number of LSPSI iterations, i.e., the number of policy evaluation steps in Alg. 2, is lower than or equal to that of PI. The rationale is that the decrease in the number of iterations up to convergence will reduce the overall convergence time.

Finally, the computational complexity of the combined algorithm TABA and LSPSI is $\mathcal{O}(k\times|S|\times|D|\times Z+|S|\times|A|\times Z)$. This proposed algorithm will benefit both TABA and LSPSI computational gains.

Since LSPSI's modified policy evaluation step can be seen as the application of TABA with a reduced number of actions, one can expect the number iterations of this step to be similar to TABA's for each application of the *policy update step* within LSPSI. Hence, one can expect the number of iterations, counted as the number of modified policy evaluation steps, to be bounded by the product of the number of TABA and PI iterations. The same applies to TABA+LSPSI. The computational gains stem from the reduced computational complexity at each iteration.

IV. Numerical Experiments

To illustrate the application and provide insights into the performance of the proposed algorithms, we will use an important example from the area of supply chains.

A. Mining industry supply chain problem

We evaluate our algorithms in the light of the iron ore logistics operations introduced in [34]. The explored mining enterprise encompasses the whole supply chain, with mines, port, intermediate storage (IS), customers under contract and a spot market (Figure 1).

At each decision epoch, the decision maker observes the system's state and selects a feasible decision whilst aiming to maximise the net present value of the company's future profit. The decision variables (available actions at time t, $a_t \in A$, arrows and black bold underlined text in Figure 1) are: Mine production (a_t^1); *Volume sent from port to intermediate storage (IS) (a_t^2)*; Volume sent from port to customers under contract (a_t^3); *Volume sent from port to spot market (a_t^4)*; Volume sent from IS to customer under contract (a_t^5); *Volume sent from IS to spot market (a_t^6).*

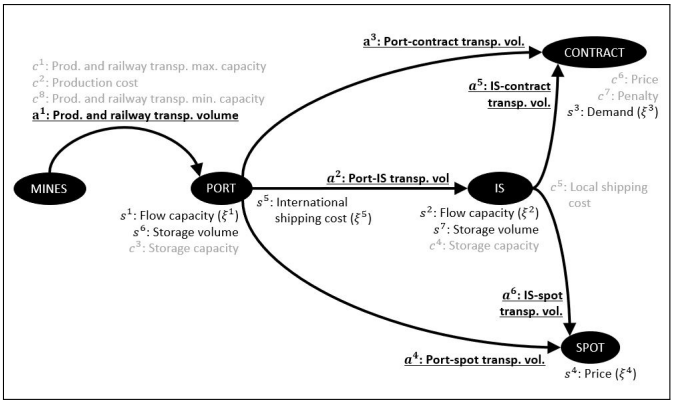

Figure 1. Mine-to-client supply chain model

To choose the best actions, the company should evaluate seven state variable components ($s_t \in S,\ \forall\, t$, black text in Fig. 1): Port flow capacity (s_t^1); *IS flow capacity (s_t^2)*; Demand for customers under contract (s_t^3); *Spot price (s_t^4)*; International shipping price from port (s_t^5); *Port initial storage volume (s_t^6)*; IS initial storage volume (s_t^7).

The random variable vector has 5 components, also described in Figure 1: Port flow capacity (ξ_t^1); *IS flow capacity (ξ_t^2)*; Demand for customers under contract (ξ_t^3); *Spot price (ξ_t^4)*; International shipping price from port (ξ_t^5).

After state, action and random variable vectors are presented, it is possible to describe the transition function ($s_{t+1} = f(s_t, a_t, \xi_t)$) as: $s_{t+1}^1 = \xi_t^1$; $s_{t+1}^2 = \xi_t^2$; $s_{t+1}^3 = \xi_t^3$; $s_{t+1}^4 = \xi_t^4$; $s_{t+1}^5 = \xi_t^5$; $s_{t+1}^6 = s_t^6 + a_t^1 - a_t^2 - a_t^3 - a_t^4$; $s_{t+1}^7 = s_t^7 + a_t^2 - a_t^5 - a_t^6$.

This is clearly an SSMDP (see Assumption 1), with $K = 2$ components depending on previous states and actions, and $(I - K) = 5$ stationary stochastic components.

The model contains eight fixed parameters: maximum production capacity (c^1); *production cost(c^2)*; port storage capacity (c^3); *IS storage capacity (c^4)*; local shipping cost (c^5); *contractual customer price (c^6)*; contractual penalty for unmet demand (c^7); *minimum production capacity (c^8).*

The system has typical supply chain constraints relating to capacity, flow conservation and demand: 1) Max. Prod. Capacity: $a_t^1 \leq c^1$; 2) Min. Prod. Capacity: $a_t^1 \geq c^8$; 3) Port Max. Flow: $a_t^2+a_t^3+a_t^4 \leq s_t^1$; 4) IS Max. Flow: $a_t^5+a_t^6 \leq s_t^2$; 5) Port Stor. Cap.: $s_t^6+a_t^1-a_t^2-a_t^3-a_t^4 \leq c^3$; 6) IS Stor. Cap.: $s_t^7 + a_t^2 - a_t^5 - a_t^6 \leq c^4$; 7) Contract Demand: $a_t^3 + a_t^5 \leq s_t^3$.

The single period profit ($r_t(s,a)$) is the total revenue (*TotRev*), minus total production and logistics cost (*TC*), minus penalty cost (*Penal*) for undelivered contract volumes:

$$
\begin{aligned}
&r_t(s,a) = TotRev(s,a) - TC(s,a) - Penal(s,a),\\
&TotRev(s,a) = c^6 \cdot (a_t^3 + a_t^5) + s_t^4 \cdot (a_t^4 + a_t^6),\\
&TC(s,a) = c^2 \cdot a_t^1 + s_t^5 \cdot (a_t^2 + a_t^3 + a_t^4) + c^5 \cdot (a_t^5 + a_t^6),\\
&Penal(s,a) = c^7 \cdot (s_t^3 - a_t^3 - a_t^5).
\end{aligned}
$$

The goal is to maximise the long-term discounted reward Eq. (1), with discount factor $0 \leq \lambda < 1$.

B. Experimental results

We ran the experiments in a personal computer with Intel® Core™ i7-7500 CPU, @2.70GHz 2.9GHz processor and 16,00GB RAM,

running Windows 10. We optimised the logistics operations described in Section IV-A, with the parameters in Table I, where $p(x)$ means "*probability that x will occur*". Considering all possible system's configurations, there are 1,296 possible states which, combined with all feasible decisions, result in more than 169 million of achievable transitions (initial state, decision and final state) and more than 1.5 million transitions to groups (initial state, decision and final group). In our experiment, each time period (t) represents one calendar month.

Table I
PARAMETER SETTINGS

Category	Parameter settings
Mines	$c^1 = 13\ kt\ (kt = 1,000t)$
	$c^8 = 8\ kt$
	$c^2 = \$12$
Port and IS	$c^3 = 3\ kt$
	$s^1 \in \{10; 11; 12\}\ kt; \begin{cases} p(10) = p(12) = 20\% \\ p(11) = 60\% \end{cases}$
	$c^4 = 2\ kt$
	$s^2 \in \{2; 3\}\ kt; \begin{cases} p(2) = 40\% \\ p(3) = 60\% \end{cases}$
Shipping	$s^5 \in \{16; 18; 20\}\ kt; \begin{cases} p(16) = p(20) = 20\% \\ p(18) = 60\% \end{cases}$
	$c^5 = \$1$
Customers	$c^6 = \$60$
	$s^3 \in \{8; 9\}\ kt; \{\ p(8) = p(9) = 50\%$
	$c^7 = \$100$
	$s^4 \in \{30; 60; 90\}\ kt; \begin{cases} p(30) = p(90) = 25\% \\ p(60) = 50\% \end{cases}$
	$c^5 = \$1$

To implement LSPSI and LSPSI+TABA, it is necessary to define parameter α, recalling that when $\alpha = 0$ LSPSI becomes PI and when $\alpha = 1$ it becomes VI. To measure the impact of α, 5 different values are applied (0.1%, 1.0%, 5.0%, 10.0% e 20.0%) with 3 different values of λ (0.99, 0.95 and 0.9). Since both LSPSI and LSPSI+TABA involve a random selection of actions, the results include the mean, standard deviation and 95% confidence interval of each analysed outcome over multiple runs.

Table II summarises the LSPSI results for different values of α, under 50 runs. For all experiments, we report the number of LSPI iterations as the number of modified policy evaluation steps in Alg. 2. As expected, the number of iterations decreases as α increases. For $\lambda = 0.90$, the fastest convergence is for $\alpha = 1.0\%$, whilst for $\lambda = 0.95$ and $\lambda = 0.99$, $\alpha = 0.1\%$ attains fastest convergence.

Table II
LSPI PERFORMANCE FOR DIFFERENT PARAMETERS α

λ			α's 0,1%	1,0%	5,0%	10%	20%
0.90	Num. Iterations	Mean	110.1	55.1	37.5	32.0	28.3
		SD	5.8	2.5	0.9	0.7	0.8
		CI (95%)	[98.8, 121.5]	[50.2, 60.0]	[35.7, 39.4]	[30.6, 33.4]	[26.8, 29.8]
	Tot. Time (s)	Mean	166.1	**117.6**	166.4	219.7	313.6
		SD	11.5	13.5	3.1	3.9	18.9
		CI (95%)	[143.5, 188.7]	[91.1, 144.2]	[160.4, 172.4]	[212.0, 227.5]	[276.5, 350.6]
0.95	Num. Iterations	Mean	129.3	68.1	46.7	41.0	37.7
		SD	5.0	1.5	0.9	0.4	0.7
		CI (95%)	[119.5, 139.0]	[65.2, 71.0]	[45.0, 48.4]	[40.1, 41.8]	[36.4, 39.0]
	Tot. Time (s)	Mean	**171.2**	184.4	237.0	266.2	404.7
		SD	23.6	15.7	24.4	15.3	23.1
		CI (95%)	[124.9, 217.4]	[153.6, 215.2]	[189.2, 284.7]	[236.2, 296.2]	[359.3, 450.1]
0.99	Num. Iterations	Mean	199.3	112.6	91.0	88.7	87.8
		SD	2.7	1.1	0.0	0.7	0.4
		CI (95%)	[194.0, 204.6]	[110.4, 114.8]	[91.0, 91.0]	[87.2, 90.1]	[87.1, 88.6]
	Tot. Time (s)	Mean	**149.1**	165.4	289.4	504.2	877.7
		SD	13.5	21.5	2.8	35.7	57.2
		CI (95%)	[122.6, 175.6]	[123.2, 207.6]	[283.9, 294.9]	[434.1, 574.2]	[765.6, 989.8]

Table III summarises the results for LSPSI+TABA over 100 different runs. For all experiments, we report the number of LSPI+TA iterations as the number of modified policy evaluation steps in Alg. 2. As expected, it outperforms LSPSI under all parameter configurations. Note that as α increases, the total number of iterations decreases, however the total time increases. For LSPSI+TABA, $\alpha = 0.1\%$ attained the fastest convergence for all discount factors. Based on these results, we will subsequently use the value $\alpha = 0,1\%$ for both LSPSI and LSPSI+TABA in the next experiment.

Table III
LSPSI+TABA PERFORMANCE FOR DIFFERENT PARAMETERS α

λ			α's 0.1%	1.0%	5.0%	10%	20%
0.90	Num. Iterations	Mean	118.7	59.6	40.0	34.3	30.7
		SD	6.6	1.9	0.7	0.6	0.5
		CI (95%)	[105.8, 131.6]	[55.8, 63.3]	[38.6, 41.4]	[33.2, 35.5]	[29.7, 31.6]
	Tot. Time (s)	Mean	**5.8**	6.5	10.3	14.1	21.7
		SD	0.7	0.7	0.6	0.8	1.4
		CI (95%)	[4.3, 7.2]	[5.2, 7.8]	[9.1, 11.5]	[12.5, 15.8]	[19.1, 24.4]
0.95	Num. Iterations	Mean	130.6	69.1	45.6	40.1	37.0
		SD	5.2	1.4	0.6	0.4	0.2
		CI (95%)	[120.5, 140.7]	[66.4, 71.8]	[44.4, 46.8]	[39.3, 40.8]	[36.5, 37.4]
	Tot. Time (s)	Mean	**6.2**	7.1	11.3	16.5	25.2
		SD	0.6	0.4	0.9	1.0	1.8
		CI (95%)	[4.9, 7.4]	[6.2, 7.9]	[9.5, 13.1]	[14.6, 18.4]	[21.6, 28.8]
0.99	Num. Iterations	Mean	204.1	116.0	95.0	93.0	92.0
		SD	2.8	0.8	0.1	0.0	0.0
		CI (95%)	[198.6, 209.5]	[114.4, 117.6]	[94.8, 95.2]	[93.0, 93.0]	[92.0, 92.0]
	Tot. Time (s)	Mean	**5.8**	8.6	19.3	32.6	56.9
		SD	0.4	1.1	1.2	1.9	3.7
		CI (95%)	[5.1, 6.5]	[6.5, 10.7]	[17.0, 21.7]	[28.9, 36.2]	[49.7, 64.1]

To illustrate the effect of α on the convergence time, Fig. 2 depicts the variation of the cost-to-go function, calculated as the logarithm of the infinite norm percentage, over time - for all values of λ and α. The error tolerance ϵ was set to 10^{-4}. Note that with lower values of α the convergence of the modified policy evaluation step is faster, but the algorithm needs more policy improvement steps to converge - as fewer actions are examined. The opposite holds for higher values of α. Fig. 2 shows that, even needing more policy improvement steps, $\alpha = 0.1\%$ attains the fastest convergence for all values of λ.

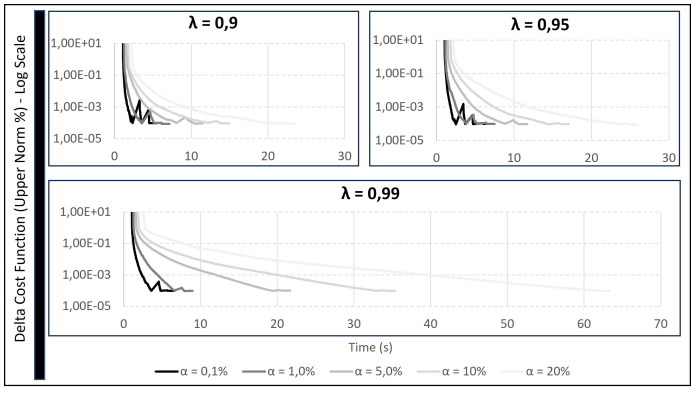

Figure 2. LSPSI+TABA convergence overt the time for different λ's and α's

To evaluate the comparative efficiency of the proposed algorithms, we implemented value iteration (VI), policy iteration (PI) and policy set iteration (PSI) [10] in addition to TABA, LSPSI and TABA+LSPI. We considered three distinct discount factors (λ): 0.9, 0.95 and 0.99. As expected, all algorithms converged to the same optimal policy for each λ. The structure of the optimal policy is detailed in [34], and the results are summarised in Table IV.

As expected, Table IV shows that VI and PI take longer to converge. To serve as another baseline, we also implemented the PSI algorithm [10] with 10 additional random policies at each policy set evaluation step. As expected, PSI needed less iterations than PI in

Table IV
COMPARISON BETWEEN IMPLEMENTED ALGORITHMS

λ		VI	PI	PSI	LSPSI	TABA	LSPSI+TABA
0.90	# Iterations	23	6	4	110.1	27	118.7
	Tot. Time (s)	1,151	298	287	166.1	31.5	**5.8**
	Iter. Time (s)	50.0	49.7	71.8	1.51	1.17	0.05
0.95	# Iterations	33	8	5	129.3	34	130.6
	Tot. Time (s)	2,117	495	488	171.2	44.8	**6.2**
	Iter. Time (s)	64.1	61.8	97.6	1.32	1.32	0.05
0.99	# Iterations	85	11	5	199.3	92	204.1
	Tot. Time (s)	4,743	596	1,034	149.1	114	**5.8**
	Iter. Time (s)	55.8	54.2	206	0.75	1.24	0.03

all examples; however the policy set evaluation time was so large for $\lambda = 0.99$ as to render the algorithm slower than PI in this instance. Observe that LSPSI outperforms PSI, as well as VI and PI. TABA is even more efficient and considerably outperforms LSPSI, converging about five times as fast for $\lambda = 0.9$ and nearly four times as fast for $\lambda = 0.95$.

Finally, LSPSI+TABA attains the best overall results; it is orders of magnitude better than VI, PI and PSI, whilst also topping TABA and LSPSI. The increased efficiency is due to a very fast iteration time, owing to both LSPSI's action sampling and TABA's state-space partition. When compared to standard VI, LSPSI+TABA converged nearly 200 times as fast for $\lambda = 0.9$, 340 times as fast for $\lambda = 0.95$, and almost 820 times as fast for $\lambda = 0.99$. This is a very interesting result, as it suggests that LSPSI+TABA is relatively insensitive to increasing discount factors, whilst standard VI and PI become increasingly slower as the discount factor approaches one.

Notice that, as anticipated in Section III-D, TABA's iteration count is similar to VI's. Furthermore, the iteration count for LSPI and LSPI+TABA is similar for all experiments, never exceeding the product between TABA's and PI's iteration counts.

C. Problem variations

To better understand how LPSI+TABA performs when the complexity of the problem increases, the algorithm was tested in two variations of the original problem. Table V details the comparison among the three problem instances.

Table V
COMPARISON AMONG THE THREE PROBLEM INSTANCES

	States	Decisions (average)	Total Transitions	Sets	Transitions to sets	Elements per set
Original problem	1,296	101	169,435,152	12	1,568,844	108
Instance 2	5,400	101	2,941,582,500	12	6,536,850	450
Instance 3	2,160	221	1,030,970,268	20	9,546,021	108

1) Instance 2 - Increase only in the number of group elements: In this variation, we increased the number of possible states for certain stationary components of the state. Table VI details the parameter variations, which resulted in an increase of the number of group elements in each set G_z, from 108 to 450. The number of states increased from 1,296 to 5,400, which, combined with all feasible decisions, resulted in more than 2.9 billion achievable transitions (initial state, decision, and final state).

As detailed in Section III-D, the computational complexity of the combined algorithm TABA and LSPSI is $\mathcal{O}(k \times |S| \times |D| \times Z + |S| \times |A| \times Z)$. When Z remains the same, the computational complexity is linear in $|S|$. As the number of elements in the group and the number of states ($|S|$) increased 4.16 times and the number of decisions ($|D|$) and the number of groups (Z) remained the same, it was expected that the total convergence time would be almost 4 times longer. Table VII confirms this: as an example for $\lambda = 0.90$ and $\alpha = 20\%$, the total average time increased approximately four times, from 21.7 to 84.3 seconds. However, the number of iterations remained similar: as an example for $\lambda = 0.99$ and $\alpha = 0.1\%$, the average number of iterations increased from 204.1 to 214.4.

2) Instance 3 - Increase in the number of groups and decisions: In this variation, the number of group elements remained the same; however, the number of groups ($|Z|$) increased, from 12 to 20 (1.7 times). The number of decisions ($|D|$) also increased, from 101 to 221 (2.2 times). Table VIII details the parameter variations for this instance. The number of states ($|S|$) rose from 1,296 to 2,160 (1.7 times), which, combined with all feasible decisions, resulted in more than 1.0 billion achievable transitions.

Table VI
PARAMETER CHANGES - INSTANCE 2 VS. ORIGINAL PROBLEM

Category	Parameter settings
Shipping	$s^5 \in \{16; 17; 18; 19; 20\}\ kt$; $\begin{cases} p(16) = p(20) = 15\%; \\ p(17) = p(19) = 20\%; \\ p(18) = 30\%; \end{cases}$
Customers	$s^3 \in \{7; 8; 9\}\ kt$; $\{\ p(7) = p(8) = p(9) = 33.3\%$
	$s^4 \in \{30; 45; 60; 75; 90\}\ kt$; $\begin{cases} p(30) = p(90) = 15\% \\ p(45) = p(75) = 20\% \\ p(60) = 30\% \end{cases}$

Table VII
LSPSI+TABA PERFORMANCE FOR INSTANCE 2

λ			α's 0.1%	1.0%	5.0%	10%	20%
0.90	Num. Iterations	Mean	121.5	59.5	39.5	34.1	30.0
		SD	4.5	0.9	0.5	0.3	0.0
		CI (95%)	[112.6, 130.4]	[57.6, 61.3]	[38.5, 40.5]	[33.5, 34.7]	[30.0, 30.0]
	Tot. Time (s)	Mean	**22.1**	25.8	39.8	56.3	84.3
		SD	3.3	2.7	2.1	2.9	4.3
		CI (95%)	[15.6, 28.6]	[20.5, 31.1]	[35.7, 43.9]	[50.6, 62.0]	[75.8, 92.7]
0.95	Num. Iterations	Mean	131.6	68.9	45.6	40.0	37.0
		SD	3.1	0.8	0.5	0.0	0.0
		CI (95%)	[125.5, 137.7]	[67.4, 70.4]	[44.6, 46.6]	[40.0, 40.0]	[37.0, 37.0]
	Tot. Time (s)	Mean	**21.6**	28.0	43.2	64.5	102.1
		SD	2.3	1.7	2.3	3.1	4.5
		CI (95%)	[17.1, 26.1]	[24.7, 31.4]	[38.6, 47.7]	[58.5, 70.6]	[93.3, 110.8]
0.99	Num. Iterations	Mean	214.4	125.1	102.9	100.0	99.0
		SD	1.3	0.4	0.3	0.0	0.0
		CI (95%)	[212.0, 216.9]	[124.2, 126.0]	[102.4, 103.5]	[100.0, 100.0]	[99.0, 99.0]
	Tot. Time (s)	Mean	**22.8**	32.9	82.2	138.7	251.2
		SD	2.7	1.8	3.9	5.0	7.3
		CI (95%)	[17.5, 28.1]	[29.3, 36.4]	[74.6, 89.8]	[129.0, 148.5]	[236.9, 265.6]

As $|S|$, $|D|$, and Z increased, it was expected that the total convergence time would be almost 6.1 times longer ($1.7 \times 2.2 \times 1.7$). Table IX confirms this: as an example for $\lambda = 0.90$ and $\alpha = 20\%$, the total average time increased about sixfold from 21.7 to 136.6 seconds. However, the number of iterations reduced, as an example for $\lambda = 0.99$ and $\alpha = 0.1\%$, the average number of iterations fell 13% from 204.1 to 177.4.

Table VIII
PARAMETER CHANGES - INSTANCE 3 VS. ORIGINAL PROBLEM

Category	Parameter settings
Mines	$c^1 = 14\ kt\ (kt = 1,000t)$
Port and IS	$c^3 = 4\ kt$
	$s^1 \in \{12; 13; 14\}\ kt$; $\begin{cases} p(12) = p(14) = 20\% \\ p(13) = 60\% \end{cases}$
	$c^4 = 3\ kt$
	$s^2 \in \{3; 4\}\ kt$; $\begin{cases} p(3) = 40\% \\ p(4) = 60\% \end{cases}$

V. CONCLUDING REMARKS

This work presented TABA, an MDP algorithm based on time aggregation for a class of MDPs with stationary state space components. The algorithm establishes and solves an equivalent problem with a reduced state space and is able to converge orders of magnitude faster than traditional value and policy iteration algorithms. By clustering the states according to the components that depend upon the state-action pair, the approach iterates on the clusters and is proved to converge to the optimal solution. The algorithm is especially efficient when the number of clusters is significantly smaller than the size of the state space. That happens, for example, in mining supply chain problems that are subject to a large number of exogenous uncertainties, which should be realised before taking a decision.

λ			α's 0.1%	1.0%	5.0%	10%	20%
0.90	Num. Iterations	Mean	106.1	55.8	37.2	32.5	29.6
		SD	5.6	1.8	0.7	0.5	0.5
		CI (95%)	[95.1, 117.2]	[52.3, 59.3]	[35.8, 38.6]	[31.5, 33.4]	[28.6, 30.5]
	Tot. Time (s)	Mean	**31.3**	38.1	59.9	86.3	136.6
		SD	3.4	3.2	4.0	4.2	6.4
		CI (95%)	[24.6, 38.0]	[31.8, 44.4]	[52.1, 67.8]	[78.1, 94.5]	[124.0, 149.2]
0.95	Num. Iterations	Mean	116.6	64.6	43.9	39.0	36.3
		SD	3.2	1.2	0.6	0.5	0.5
		CI (95%)	[110.3, 123.0]	[62.2, 67.1]	[42.7, 45.0]	[38.1, 39.9]	[35.4, 37.2]
	Tot. Time (s)	Mean	**35.2**	40.7	66.4	100.1	158.7
		SD	3.8	2.3	3.2	4.4	8.1
		CI (95%)	[27.8, 42.5]	[36.2, 45.1]	[60.1, 72.6]	[91.6, 108.6]	[142.8, 174.6]
0.99	Num. Iterations	Mean	177.4	96.6	79.6	78.0	77.0
		SD	2.1	0.6	0.5	0.0	0.0
		CI (95%)	[173.3, 181.5]	[95.5, 97.7]	[78.7, 80.6]	[78.0, 78.0]	[77.0, 77.0]
	Tot. Time (s)	Mean	**32.7**	44.9	105.1	181.2	319.8
		SD	2.9	3.9	4.1	5.9	9.9
		CI (95%)	[27.0, 38.5]	[37.2, 52.7]	[97.2, 113.1]	[169.7, 192.7]	[300.5, 339.2]

Table IX
LSPSI+TABA PERFORMANCE FOR INSTANCE 3

To circumvent the size of the action space, we also proposed LSPI, which uses a generalised policy evaluation that applies value iteration whilst considering only a sample of actions from the action space. This amounts to a local search within a very large set of available policies that accelerates convergence. Albeit simple, the approach introduces a novel way of sampling policies that produces a powerful local search, is guaranteed to reach the optimal solution and can produce significant computational savings in settings with a large number of feasible actions, such as mining supply chain logistics.

Finally, the paper also introduces (TABA+LSPSI), an algorithm that combines the previous approaches. By simultaneously reducing the size of the state space and effectively sampling from the action space, the algorithm's iterations are fast and effective and the convergence time is significantly reduced with respect to both TABA and LSPSI in the numerical experiments. The three algorithms are compared in the light of a complex mining supply chain problem. All of them outperform classical value and policy iteration, as well as the policy set iteration algorithm by a significant margin.

Possible extensions of this work include testing the algorithms with large scale synthetic problems and an attempt to extend TABA for general MDPs that do not meet the semi-stationarity assumption. In such cases, there may not exist an intuitive state space partition with $Z \ll |S|$ and the steady state probability (μ_z) may not be trivially calculated. Another research direction is to investigate distinct sampling strategies for the LSPSI algorithm and to infer the impact of such strategies in the convergence time of the algorithm. Finally, with some adaptations, TABA can be an alternative when it is possible to convert infinite state spaces into finite disjoint sets, and the sampling scheme used by LSPSI can be an alternative to infinite action spaces.

ACKNOWLEDGMENT

This work was partially supported by the National Council for Scientific and Technological Development (CNPq), under grants #311075/2018-5 and #305180/2016-9; by the Carlos Chagas Filho Foundation for Research Support of the State of Rio de Janeiro (FAPERJ), under Grant no. E-26/202.789/2015; and by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Finance Code 001.

REFERENCES

[1] W. Powell, "A unified framework for stochastic optimization," *European Journal of Operational Research*, vol. 275, no. 3, pp. 795–821, 6 2019.
[2] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1994.
[3] D. Bertsekas, *Dynamic Programming and Optimal Control Volume II*. Belmont, Massachusetts: Athena Scientific, 1995.
[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.
[5] R. Bellman, "A Markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, pp. 679–684, 1957.
[6] R. Howard, *Dynamic programming and Markov processes*. New York: The MIT Press, 1960.
[7] E. A. Feinberg and J. Huang, "The value iteration algorithm is not strongly polynomial for discounted dynamic programming," *Operations Research Letters*, vol. 42, no. 2, pp. 130–131, 2014.
[8] B. Scherrer, "Improved and generalized upper bounds on the complexity of policy iteration," *Mathematics of Operations Research*, vol. 41, no. 3, pp. 758–774, 2016.
[9] A. Almudevar and E. Arruda, "Optimal approximation schedules for a class of iterative algorithms, with an application to multigrid value iteration," *IEEE Transactions on Automatic Control*, vol. 57, no. 12, pp. 3132–3146, 2012.
[10] H. Chang, "Policy set iteration for Markov decision processes," *Automatica*, vol. 49, no. 12, pp. 3687–3689, 12 2013.
[11] ——, "Value set iteration for Markov decision processes," *Automatica*, vol. 50, no. 7, pp. 1940–1943, 7 2014.
[12] D. Bertsekas, "Lambda-Policy Iteration: A Review and a New Implementation," in *Reinforcement learning and approximate dynamic programming for feedback control*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2013, pp. 381–409.
[13] E. A. Hansen and S. Zilberstein, "LAO*: A heuristic search algorithm that finds solutions with loops," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 35–62, 6 2001.
[14] H. Robbins and S. Monro, "A Stochastic Approximation Method," pp. 400–407, 1951.
[15] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 5 1992.
[16] W. Sun, Y. Wang, F. Zhang, and Y. Zhao, "Dynamic allocation of surplus by-product gas in a steel plant by dynamic programming with a reduced state space algorithm," *Engineering Optimization*, vol. 50, no. 9, pp. 1578–1592, 9 2018.
[17] A. M. Devraj and S. P. Meyn, "Q-Learning With Uniformly Bounded Variance," *IEEE Transactions on Automatic Control*, vol. 67, no. 11, pp. 5948–5963, 11 2022.
[18] R. Bellman and S. Dreyfus, "Functional Approximations and Dynamic Programming," *Mathematical Tables and Other Aids to Computation*, vol. 13, no. 68, p. 247, 10 1959.
[19] W. Powell, *Approximate Dynamic Programming Solving the Curses of Dimensionality*. New Jersey, USA: John Wiley & Sons, Inc., 2011.
[20] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton: CRC Press, 7 2017.
[21] M. N. Katehakis and L. C. Smit, "A successive lumping procedure for a class of Markov chains," *Probability in the Engineering and Informational Sciences*, vol. 26, pp. 483–508, 2012.
[22] B. C. Geiger, T. Petrov, G. Kubin, and H. Koeppl, "Optimal Kullback-Leibler aggregation via information bottleneck," *IEEE Transactions on Automatic Control*, vol. 60, pp. 1010–1022, 4 2015.
[23] M. N. Katehakis, L. C. Smit, and F. M. Spieksma, "A comparative analysis of the successive lumping and the lattice path counting algorithms," *Journal of Applied Probability*, vol. 53, pp. 106–120, 7 2016.
[24] R. Amjad, C. Blochl, and B. Geiger, "A generalized framework for Kullback-Leibler Markov aggregation," *IEEE Transactions on Automatic Control*, vol. 65, pp. 3068–3075, 7 2020.
[25] X. Cao, Z. Ren, S. Bhatnagar, M. Fu, and S. Marcus, "A time aggregation approach to Markov decision processes," *Automatica*, vol. 38, no. 6, pp. 929–943, 6 2002.
[26] E. Arruda and M. Fragoso, "Solving average cost Markov decision processes by means of a two-phase time aggregation algorithm," *European Journal of Operational Research*, vol. 240, no. 3, pp. 697–705, 2 2015.
[27] ——, "Discounted Markov decision processes via time aggregation," in *2016 European Control Conference, ECC 2016*. Institute of Electrical and Electronics Engineers Inc., 1 2016, pp. 2578–2583.
[28] E. Arruda, M. Fragoso, and F. Ourique, "A multi-cluster time aggregation approach for Markov chains," *Automatica*, vol. 99, pp. 382–389, 1 2019.
[29] R. Dimitrakopoulos, C. Farrelly, and M. Godoy, "Moving forward from traditional optimization: grade uncertainty and risk effects in open-pit design," *Mining Technology*, vol. 111, no. 1, pp. 82–88, 2002.
[30] L. Montiel and R. Dimitrakopoulos, "A heuristic approach for the stochastic optimization of mine production schedules," *Journal of Heuristics*, vol. 23, no. 5, pp. 397–415, 2017.
[31] Y. A. Sari and M. Kumral, "Dig-limits optimization through mixed-integer linear programming in open-pit mines," *Journal of the Operational Research Society*, vol. 69, no. 2, pp. 171–182, 2018.
[32] R. Goodfellow and R. Dimitrakopoulos, "Simultaneous Stochastic Optimization of Mining Complexes and Mineral Value Chains," *Mathematical Geosciences*, vol. 49, no. 3, pp. 341–360, 2017.
[33] J. Zhang and R. Dimitrakopoulos, "Stochastic optimization for a mineral value chain with nonlinear recovery and forward contracts," *Journal of the Operational Research Society*, vol. 69, no. 6, pp. 864–875, 6 2018.
[34] J. Leite, E. Arruda, L. Bahiense, and L. Marujo, "Mine-to-client planning with Markov Decision Process," in *2020 European Control Conference (ECC)*, 2020, pp. 1123–1128.
[35] R. A. Brealey, S. C. Myers, F. Allen, and A. Edmans, *Principles of corporate finance*, 14th ed. Boston, MA: McGraw Hill LLC, 2023.
[36] D. Qiao and M. C. Gursoy, "Age-optimal power control for status update systems with packet-based transmissions," *IEEE Wireless Communications Letters*, vol. 8, pp. 1604–1607, 12 2019.
[37] H. Huang, D. Qiao, and M. C. Gursoy, "Age-energy tradeoff optimization for packet delivery in fading channels," *IEEE Transactions on Wireless Communications*, vol. 21, pp. 179–190, 1 2022.