State Aware Principal Action Space Embedding for Centralized MARL

Tapan Shah
GE Global Research
shah.tapan.r@gmail.com

ABSTRACT

Reinforcement Learning is increasingly being used to solve planning and control problems in dynamic environments. Centralized multi-agent reinforcement learning (C-MARL), while maintaining Markovian property, suffers with scaling of agents due to explosion of action space which leads to algorithmic and hardware challenges. Motivated by applications in autonomous wind farms, we propose an encoding-decoding algorithm for state-aware embedding of actions from action space into a lower dimensional space. This allows learning of joint action policies in a lower dimension, which can be decoded back to the original space. Using a wind farm simulation environment, we show that a combination of deep Q-learning and action embedding can achieve faster convergence, higher rewards and improved scalability. **We want to highlight that all the results and plots in this paper are generated using simulated data.**

KEYWORDS

reinforcement learning, deep Q-learning, multi-agent systems, action dimension reduction

1 INTRODUCTION

Typical operations research problems consist of devising optimal policies for resource allocation/control/planning type of operations in a multi-agent operational setup with global and local constraints. Solving these problems has several applications, e.g. air-transport [1], military planning [34], health-care [4], renewable energy [1] etc. In the subsequent subsection, we describe one such example in the context of an autonomous wind farm. Even though methods described in the later part of the paper are generic enough for wide scale application, we intend to highlight the novelty, both in the application and the solution. Hence, we use two subsections to introduce and motivate the research.

1.1 Motivation: Blade Heating System (BHS) for management for autonomous wind farms

In colder countries, ice accretion on wind turbine blades is a common reason for loss of energy production. According to various studies, the losses in annual energy production (AEP) can be as high as $40 - 50$ % [14, 21]. Additionally, there are other safety and reliability issues. This necessitates deployment of an ice mitigation system in a wind farm in cold locations.

A typical ice mitigation system consists of a blade heating system (BHS) and an icing detector on each turbine. The BHS is an electric coil which can heat up the blade and melt the ice. However, there are certain trade-offs and constraints while operating the BHS.

(1) Starting the BHS on an iced wind turbine re-starts the power generation from the turbine.
(2) The BHS requires power for its operation.
(3) If the expected wind speeds in the near future are low and blade is iced, starting the BHS will generate no power. It might be a better policy to not start the BHS and wait till the wind speeds pick up.
(4) The best policy to operate the BHS is the one which maximizes the power output of the wind farm and at the same time minimizing the power consumption of BHS.
The traditional method to solve such problems is to formulate it as a constrained integer optimization problem,

$$\max_{i_1,\ldots,i_n} \text{Expected Farm Power}, \tag{1}$$

$$\text{subject to Total BHS} \leq \text{limit}, \tag{2}$$

where $[i_i, \ldots, i_n]$ is a binary vector of length equal to number of turbines (n) and $i = 1$ indicates BHS should be started and i = 1 indicates BHS should not be started for that turbine. Techniques like branch-and-bound [16], simulated annealing [12] and heuristics [10] are used to solve problems of this type.

Challenges. Any agent (human or artificial) faces the following challenges when deciding the optimal policy to start the BHS or not.

(1) As discussed earlier, the optimal policy is a function not only of the current wind speeds but also the expected wind speeds in the near future. This stochasticity adds a dynamic layer of complexity into the optimization problem. Robust optimization techniques [3] must be incorporated to solve the resulting nature of problems. Adding this robustness can be prohibitively expensive.
(2) The solution times for solving such combinatorial optimization problems can be large which is a bottleneck for edge applications.
(3) At any given time, the action taken as per the policy affects the environment in the next time instant. For example, if BHS is operated at an instant, there can be no icing till the BHS is stopped. This environmental change in the future because of current action is difficult to incorporate in a traditional combinatorial optimization setup.
(4) With a continuously changing climatic environment, the mathematical formulation might require a continuous refresh.

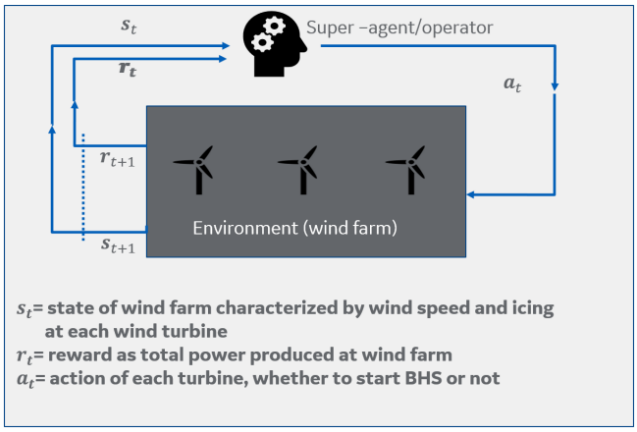

Figure 1: A schematic to illustrate the BHS management problem as a reinforcement learning problem

1.2 Markov Decision Processes and Reinforcement Learning

Markov decision process [22] is a mathematical framework which models a discrete time Markovian stochastic process. This framework is used for modeling decision policies in uncertain domains. A typical Markov decision process is characterized by

(1) A set of possible environment states $\mathcal{S}$.
(2) A set of possible actions $\mathcal{A}$.
(3) A real values reward function $\mathcal{R}(s,a)$.
(4) A function $\mathcal{T}$ which describes the effect of each action on each state.

Reinforcement learning [30] has become a popular method to devise optimal policies for a Markov decision processes. In this method, the optimal policy to choose the action for a given environment state is continually learned using the reward gained for the previous actions.

Cooperative Multi-agent reinforcement learning, whereby multiple agents simultaneously learn optimal policies for a common reward, is a natural mathematical framework which can be used to solve the problem described in Section 1.1. In Figure 1, we superimpose the cooperative MARL framework on an autonomous windfarm to schematically describe BHS management. The motivation of this work is to showcase a scalable and practical application of reinforcement learning to solve a resource allocation/scheduling problem under constrained environments.

2 PRIOR WORK

Recently, there has been efforts to use reinforcement learning to solve combinatorial optimization problems like traveling salesman and bin-packing. In a recent work by Laterre et al. [15], the authors develop a ranked rewards mechanism for single agent games and combine that with deep neural networks and tree search to solve the bin-packing problem. In a different work [11], the authors uses heuristic to convert the output sequence into a feasible solution.

In another set of work, supervised deep learning methods like Pointer Networks [33], which represent combinatorial optimization problems as sequence-to-sequence learning problems, have been developed. The main challenge there is the need for training set consisting of optimal solutions, which can be expensive. To mitigate this, the same architecture is combined with actor-critic methods [2] to avoid the need for expensive training.

The authors in [36] combines Temporal Differencing methods with negative reward mechanism (when constraints are violated) to solve resource scheduling problems. There is other branch of work like [18] where individual agents in the setup perform reinforcement learning and a common combinatorial optimization is carried out to find the optimal learning parameters.

In another line work, the authors assume centralized training and decentralized inference to propose methods like QTRAN [27], QMIX [23], VDN [29]. This methods use a monotonic factorization of the overall Q-value which is than passed onto each agent.

Narrowing our focus on centralized reinforcement learning, a couple of very important work stands out. In [9], the authors use coordination graphs to exploit conditional dependencies between agents to decompose global reward function into a sum of agent-local terms. In another very related work, Sparse cooperative Q-learning [13] encodes dependencies in a coordination graph to coordinate actions of only those agents, which require it. Some other relevant work can be found in [7, 8]. Recently, in a work very similar to ours, the authors in [5] decompose a policy into two component: one that acts in a low-dimensional space of action representation and another which transforms these into actual action. They use supervised learning to learn the action representations. Though the concepts in our paper were developed independently, it is important to highlight that there is a significant overlap in the concepts developed.

3 KEY CONTRIBUTIONS

The key contributions highlighted in this paper are summarized as follows:

(1) We develop a simulator to simulate the operation of a wind farm with multiple wind turbines, each with a ice detector and a global operator (or a manager) to manage the BHS operation.
(2) Assuming complete communication between the multiple agents (wind turbines) and the operator, we formulate a centralized MDP for the operator to jointly determine the actions (whether to start of stop of the BHS) for each agent.
(3) The most important contribution is to develop a method, State Aware Principal Action Embedding (SAPAE), to embed the action into a lower dimensional space in a state-aware manner with an easy decoding algorithm. This allows learning of joint actions in a lower dimensional space which is then decoded to the original action space.
(4) Combining SA-PAE with DQN, we show a 100-fold improvement in scalability and training times of the model.

4 BACKGROUND

A centralized MARL (C-MARL) can be characterized by a tuple $G = (S, A, P, R, N, \gamma)$ where $s \in S$ describes the global state of the environment. A central manager[1] chooses the action tuple

[1]We use manager/operator/super-agent interchangeably in this paper.

$(a_1, \ldots, a_N)$ for all the N agents, s.t. $a_i \in A, \forall i \in [1, \ldots, N]$. This is followed by a transition in the state according to the state transition function $P(s'|s, a)$. The common global reward for each action tuple and environment state is $r(s, a) \in R$ and discount factor is γ. For the rest of the paper, we assume $A \in \{0, 1\}$.

4.0.1 Binary Action Assumption. The assumption that each agent has a binary action space looks significantly restrictive. However, for the class of combinatorial optimization problems which we intend to address in this paper, it is a reasonable to make this assumption.

4.1 Q-learning and DQN

Q-learning [25, 35] is one of the first methods developed for reinforcement learning. It defines a function $Q : S \times A \rightarrow \mathcal{R}$ that measures the performance of each state-action pair combination. The Q-value is defined using the following principle

$$Q(s', a) = r(s, a) + \gamma \max_a Q(s', a). \tag{3}$$

In an iterative setting, the $Q-$value is computed as

$$Q^{new}(s_{t+1}, a_t) = Q(s_t, a_t) + \alpha \left(r(s_t, q_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right) \tag{4}$$

where α determines the proportion of the new and old values. The Q-values for each combination of state and action are stored in tabular form. At any given timestamp t with a state s_t, the action which leads to the maximum Q-value is the optimal action. In a pioneering recent work, non-linear neural networks are used to approximate the Q-value [19]. The author combines this with a technique called experience replay to develop deep Q-networks (DQN).

4.2 Centralized DQN

A naive method for using Q-learning/DQN in a multi-agent scenario is using an independent learning scheme i.e. conducting learning for each agent independently, ignoring the action and rewards of each agent [20, 26, 31]. This clearly violates the Markov property. Surprisingly, the method has shown impressive performance in various settings. However, in a constraint environment, it leads to inevitable challenges.

Instead, in another naive method, we convert the action vector of length N into the corresponding integer. The range of the integers thereafter form the resulting action space which can then be fed into the DQN. For example, for $N = 4$, we get an action space of $[0, \ldots, 15]$. The method has obvious advantage and disadvantage.

(1) The Markov property is maintained.
(2) As N increases, the action space increases exponentially and hence training becomes hard due to both challenges in hardware and algorithmic challenges.
The dichotomy of the two methods is similar to that of Binary Relevance and Label Power-set methods used for multi-label classification [32].

4.3 Constrained Centralized DQN

We propose two methods to ensure that the action vector recommended by the central agent respects the global constraint.

(1) **Reduced Action Space (RAS)**: DQN agent chooses the action over a reduced action space i.e. instead of choosing the action with highest Q-value, it chooses an action with the highest Q-value satisfying the constraint,
(2) **Soft Action Penalty (SAP)**: Design a reward function which has a large negative value when a constraint violating action space is recommended.

5 STATE AWARE PRINCIPAL ACTION EMBEDDING (SA-PAE)

As discussed in Section 4.2, learning joint action vectors in a centralized setting is not a scalable approach. Instead, we propose a novel approach, State Aware Principal Action Embedding (SA-PAE) where we project the joint action in a latent space, learn the optimal policy over the reduced space and decode it back to the original action space. We borrow concepts from two main topics in machine learning:

(1) Feature aware label space dimension reduction for multi-label classification [6, 17].
(2) Online/incremental versions of matrix factorization methods like singular value decomposition, principal component analysis [24].

For each episode e of the MARL, we create a matrix $\mathbf{S}$ and $\mathbf{A}$ by concatenating all the environment state vectors and the corresponding action vectors in that episode. The matrix $\mathbf{A}_e$ is obtained by concatenating all the matrices $\mathbf{A}$ starting from episode 0 to e. Similarly, matrix $\mathbf{S}_e$ is created. In episode $e + 1$ and timestamp t, for any state-action tuple $(\mathbf{s}_e(t), \mathbf{a}_e(t))$, we project $\mathbf{a}_e(t)$ onto a latent space by a encoding/projection matrix $\mathbf{V}$. Intuitively, this matrix projects onto a space spanned by the first k principal components of $\mathbf{A}_e$ conditioned on $\mathbf{S}_e$. This embedded action, $z_e(t)$ is rounded to a binary vector and converted to the corresponding integer. The embedded action is then fed into the replay memory of the downstream DQN. The action recommended by the main agent will reside in the k dimensional latent space. This embedded action is decoded back to the original action space, using a suitable decoding strategy. A simple schematic describing SA-PAE is shown in Figure 2. Adopting the CPLST algorithm from [6], we propose an algorithm which combines DQN and SA-PAE in Figure 3. As storing the concatenated matrices $\mathbf{A}_e$ and $\mathbf{S}_e$ is impractical, we propose a sequential version of SA-PAE, which is amenable to the DQN framework.

Some math notations We use boldface small case letters for vectors, boldface capital letters for matrices. If we use $\mathbf{A}$ and $\mathbf{a}$ without any specification, then $\mathbf{A}$ is a obtained by concatenating multiple instances of $\mathbf{a}$. The matrix $\mathbf{V}$ is the $n \times k$ dimensional principal projection matrix. IncrementalPC$(:, k)$ updates the first k principal components using the algorithm described in [24]. For any matrix $\mathbf{X}$, $\mathbf{X}^\dagger = \left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X}$ is defined as the pseudo-inverse [28]. For a $p \times n$ matrix $\mathbf{X}$, the function rowMean$(\mathbf{X})$ gives a n length vector with means of each of the n rows. For a n length vector $\mathbf{a}$, updateRowMean$(\mathbf{a}, \cdot)$ updates the older row mean vector with the new data.

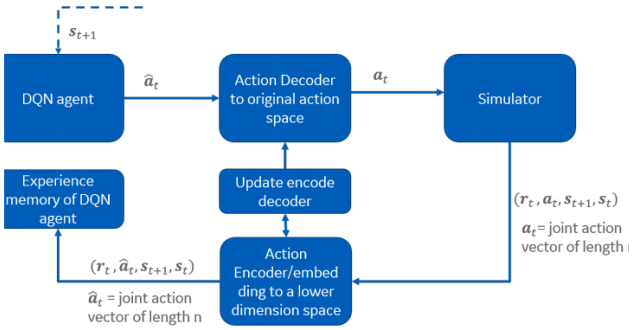

Figure 2: A schematic to illustrate the principles of SA-PAE

5.1 Intuition for the proposed SA-PAE

Due to space constraints, we explain the intuition for the proposed algorithm without going into the mathematical details (interested readers can refer to [6]). Assuming an orthogonal encoding matrix $\mathbf{V}$ and round based decoding, it can be shown that the hamming loss between the actual action and the decoded (from the lower dimension space) action is bounded by the sum of prediction error (error between the action suggested by the agent and desired action in the reduced space) and encoding error (error for projecting action into a lower dimension space). Inspired by Orthogonal Canonical Correlation Analysis, it can be shown that minimizing the encoding error is equivalent to

$$\min \text{Trace}\left(\mathbf{V}\mathbf{Z}^T\left(\mathbf{I}-\mathbf{H}\right)\mathbf{Z}\mathbf{V}^T\right),$$

where $\mathbf{Z} = \mathbf{A} - \bar{\mathbf{a}}$ and $\mathbf{H}$ is the pseudo-inverse of the state matrix $\mathbf{S}$. On the other hand, it can be shown that $\mathbf{V}$ obtained by first k right singular vectors of $\mathbf{Z}$ minimizes the prediction error. Using above concepts, it can be shown that, minimizing both prediction and encoding errors is equivalent to

$$\max \text{Trace}\left(\mathbf{V}\mathbf{Z}^T\mathbf{H}\mathbf{Z}\mathbf{V}^T\right).$$

This leads us to the encoding and decoding algorithms in Figure 4 and 5.

6 NUMERICAL SIMULATIONS FOR BHS MANAGEMENT

In this section, we discuss results of several numerical simulations. For conducting the numerical simulations, we setup a simulation environment, which we explain in the next section.

6.1 BHS-Wind Farm Simulator

We build a simulation environment for N wind turbines, which, at any timestamp t, inputs the state s and action vector a and outputs the reward and the next state. The various components are described below.

(1) **State s**: The state of a wind turbine n is defined as $s_n = (w_n, i_n)$ where w_n is the wind speed and i_n is an indicator variables which is 1 if icing is detected and 0 otherwise. The total state of the wind farm is $s = [s_1, \ldots, s_N]$.

Figure 3: A schematic of reinforcement learning training with DQN+Online SA-PAE.

Algorithm 1: A schematic of reinforcement learning training with DQN+Online SA-PAE.

n = Number of agents
k = Latent space dimension
C = criteria for episode stop
V = Principal projection matrix of dimension k
agent = Central DQN agent for a given state
env = Environment defined for the reinforcement learning
while $e <= maxEpisode$ **do**
  $episodeStop$ = FALSE **while** *not* $episodeStop$ **do**
    $\mathbf{s}$ = State
    $\hat{\mathbf{a}}$ = agent.act($\mathbf{s}$) ... (DQN agent acts upon the state to recommend an embedded action as an integer between 0 and 2^k)
    $\mathbf{a}$ = `decode(`$\mathbf{V}$, $\hat{\mathbf{a}}$`)` ... (binary vector of length n)
    $\mathbf{A}$ = concat ($\mathbf{A}, \mathbf{a}$)
    $\mathbf{S}$ = concat ($\mathbf{S}, \mathbf{s}$)
    $\mathbf{s}', r, C$ = env.step($\mathbf{s}, \mathbf{a}$) ... (updated state, reward and episode stop criteria after the applying action on current state)
    agent.remember ($\mathbf{s}, \mathbf{s}', r, \mathbf{a}$)
    if C **then**
      $episodeStop$ = TRUE
  end
  agent.replay ... (Expoential replay to update Q values using DQN)
  V =`encodeMatrix(`$\bar{\mathbf{a}}$, $\mathbf{A}$, $\mathbf{S}$`)`
end

Figure 4: Function to generate encoding matrix using incremental state aware PCA.

Algorithm 2: Function to generate encoding matrix using incremental state aware PCA.

Function `encodeMatrix(`$\bar{\mathbf{a}}$, $\mathbf{A}$, $\mathbf{S}$`):`
  $\bar{\mathbf{a}}$ = updateRowMean ($\bar{\mathbf{a}}$, rowMean ($\mathbf{A}$))
  $\mathbf{Z} = \mathbf{A} - \bar{\mathbf{a}}$
  $\mathbf{H} = \mathbf{S}\mathbf{S}^{\dagger}$
  $\mathbf{K} = \mathbf{Z}^T\mathbf{H}\mathbf{Z}$
  $\mathbf{V}$ = IncrementalPC ($\mathbf{K}, k$)
  return $\mathbf{V}$

(2) **Reward** r: The reward r_n for a given state $s_n = (w_n, i_n)$ and action a_n for turbine n is defined in such a way that it is very high when both $a_n = 0$ and $i_n = 0$ or both $a_n = 1$ and $i_n = 1$. On the other hand, we define the reward to be low when

Figure 5: Decoding strategy to convert the embedded action to an integer between 0 and 2n.

Algorithm 3: Decoding strategy to convert the embedded action to an integer between 0 and 2^n.

Function `decode(V, â):`

$\hat{a}$ = embedded action between 0 and 2^k
V = Principal projection matrix of dimension $k \times n$
n = Number of agents
$\bar{\mathbf{a}}$ = latest row mean vector
$\hat{\mathbf{a}}_b$ = toBinary $(\hat{a})$ (k-length vector)
$\mathbf{a}$ = round $(\hat{\mathbf{a}}_b \mathbf{V} + \bar{\mathbf{a}}) \ldots$
$(\text{round}(x) = 0 \quad if \quad x < 0.5 \quad else \quad 1)$
return a

$a_n = 0$ and $i_n = 1$ or $a_n = 1$ and $i_n = 0$. The total reward is

$$r(s,a) = \sum_{n=0}^{N} r_n\left(s_n, q_n\right).$$

As discussed in Section 1.1, there is a limit to the total BHS power consumed. In 4.3, we described two strategies to incorporate these constraints in a reinforcement learning formulation, **Reduced Action Space (RAS)** and **Soft Action Penalty (SAP)**. For the RAS strategy, we further modify the reward r as

$$r = -P, \quad \text{if} \quad r > C,$$

where C is the constraint and P is a large penalty value. The intuition is to provide a large negative reward if the action recommended by the agent is such that the constraint is violated.

(3) **State Transition**: For wind speed w_n, we describe an auto-regressive model with lag 1 and i.i.d Gaussian noise to describe wind speed transition. The parameters of the AR model are computed using the actual wind speeds collected from an experimental site.

$$w_n(t+1) = \max\left(0, \alpha w_n(t) + \beta + \mathcal{N}(0,\sigma)\right)$$

where α and β
are AR parameters and $\mathcal{N}(0,\sigma)$ is zero-mean Gaussian noise with variance σ^2. For icing indicator i_n, we again assume an AR-type model given by

$$i_n(t+1) = i_n(t) \quad \text{if} \quad \mathcal{B}(p) = 1,$$

where $\mathcal{B}(p)$ is a Bernoulli random variable with probability of 1 equal to p. Additionally, we enforce that

$$i_n(t+1) = 0 \quad \text{if} \quad a_n = 1.$$

This condition, which means that a turbine cannot be iced if BHS is ON is very important, both theoretically and practically. In real deployment, it is obvious that if BHS is on at any given time, icing will not happen on that turbine. Theoretically, this ensures that the environment is impacted by the action of the agent [2].

6.1.1 Assumptions and comments on the simulator. For the purpose of simplicity, we have made several overly simplistic assumptions in defining our simulator. They are 1) We have not considered any spatial correlations in the state transitions. Typically, we expect certain correlation in the state of two nearby wind turbines. 2) We assume melting of the ice and full power generation within 1 timestamp. This is typically not the case. A BHS takes finite time to melt the ice and significant delay is present between the wind turbine generating full power.

6.1.2 Episode. In deep Q-learning literature, episode is defined as a finite sequence of states, action and rewards. The exponential replay is conducted at the end of each episode. Typically, end of episode is determined by a stopping criteria like win/loss in competitive games. For our framework, we randomly select an integer between 100 and 1000 to determine the number of time stamps in an episode.

6.1.3 Cumulative Power Availability. We define average power availability PA_{avg} as the ratio of average total generated power and average total available power, where the average is computed over an episode. Cumulative power availability PA_{cum} is the cumulative mean of PA_{avg}.

6.2 Results and discussion

In this section, we have multiple sub-sections where we show simulation results for different scenarios. All the simulation were performed on a shared node with following specifications: 8 NVIDIA GPU/node, 2 Intel 2.7 GHz processor and 512 GB DRAM/node.

6.2.1 Comparison with and without SA-PAE. In this section, we compare the cumulative power availability with and without low dimension action embedding using SA-PAE for both SAP and RAS strategy to incorporate constraints (Figure 6). The simulation parameters for this set of results is in Table 1. As observed in Figure 6, SA-PAE+SAP converges to the same CPA as the other two strategies without action embedding. At the same time, the constraint is always respected. Another observation is that SAP strategy performs slightly better than the RAS strategy. Though we do not have a concrete explanation, we believe it is because the SAP strategy allows for greater exploration of the action space.

6.2.2 Performance of SA-PAE as a function of p. In this section, we study the performance of action embeddings as a function of p (Figure 4), which determines the sparsity of the action vectors. The simulation parameters for computing these set of results are the same as in Table 1, except for N, k, p and C. They are specified in Table 2.

As expected, Figure 7 confirms that the CPA reduces with a decrease in sparsity. Two observations are to be noted here. Firstly, as the sparsity decreases (p increases), the policy is very conservative to start with and gradually moves towards the constraint as compared to a higher sparsity scenario. As for $p = 0.45$, the gap

[2] If this was not true, the resulting problem will be a much simpler MDP where the environment changes without any influence by the action.

Parameter	Description	Value
N	number of agents (wind turbines)	20
p	Bernoulli parameter for icing AR model	0.4
(α, β)	Parameters of AR model for wind speeds	$(1, 2000)$
σ^2	Variance of Gaussian noise in AR model for wind speeds	200
C	Constraint on total BHS power	200
(u, v)	Number of nodes in 2 layer DQN	$(8, 4)$
γ	Discount factor	4
bs	DQN batch size	64
mem	DQN replay memory	20000

Table 1: The simulation parameters used for Section 6.2.1. Unless specified, the same parameters will be used for other results.

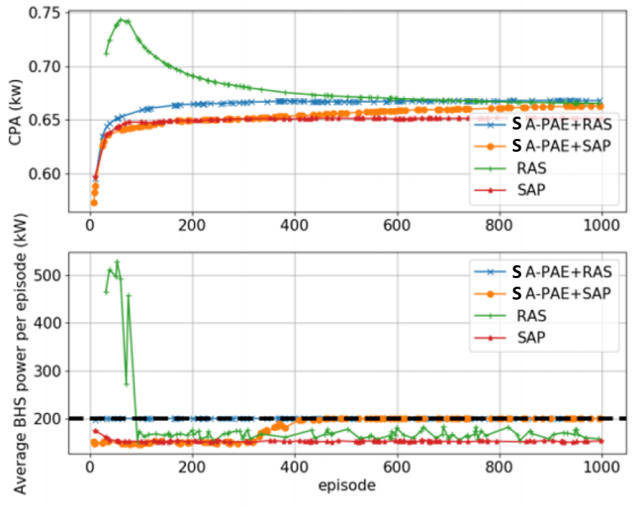

Figure 6: Comparing the CPA and BHS power consumed with and without action embedding, $N = 20$. The horizontal black line is the constraint.

is very large. Secondly, at a lower sparsity, we do see instances where the BHS power breaches the constraint. This is a drawback of the method and going ahead, we intend to make this more robust without making it overly conservative.

6.2.3 Performance of SA-PAE as function of k. In this section, we study the variation of SA-PAE as a function of dimension of the reduced action space (Figure 8). To zoom into the initial trajectories, we limit to only 500 episodes. The simulation parameters are the same as in Table 2. Based on Figure 8, certain interesting observations can be made. The difference between the maximum and minimum CPA, after convergence is around 2 percentage points.

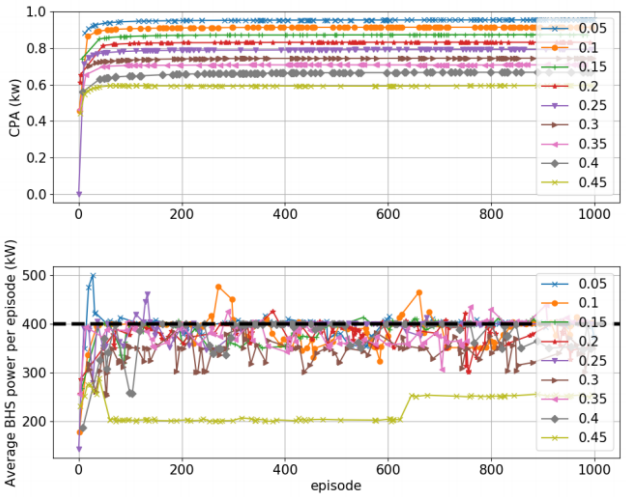

Figure 7: Comparing the CPA and BHS power consumed as a function of p, $N = 40$. The horizontal black line is the constraint. SAP strategy is used.

Parameter	Description	Value
N	Number of agents (wind turbines)	40
p	Bernoulli parameter for icing AR model	$0.05 \ldots, 0.45$
C	Constraint on total BHS power	400
k	Dimension of laten space action	8

Table 2: The simulation parameters used for Section 6.2.2. Most parameters are the same as specified in Table 1.

As the dimension of the reduced space increases, the time taken by the agent to start respecting the constraint increases. Alternately, as the dimension of the reduced space decreases, the agent becomes increasingly conservative in order to avoid breaching the constraint. This is due to significant information loss in the lower dimension.

6.2.4 Scalability of PA-SAE. In this section, we show how using SA-PAE dramatically improves the scalability of the system (Figure 9). The simulation parameters for computing these set of results are the same as in Table 1, except for N, k and C. We vary N, k and C by the following equation,

$$k = \min(20, \max(4, N/5)) \tag{5}$$

$$C = 10N, \quad N \in [10, 15, \ldots, 100]\,. \tag{6}$$

Using the resources we have, we could not train the DQN without action embedding for $N > 25$.

7 CONCLUSIONS

In this paper, we discuss a method to embed the joint action of a central agent in a multi-agent setting to improve scalability and performance while maintaining the Markov property. The key idea is to use state aware online principal component analysis to embed

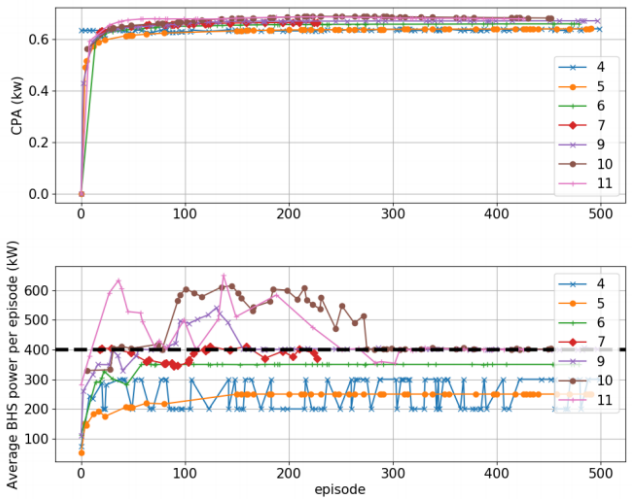

Figure 8: Comparing the CPA and BHS power consumed as a function of k with SA-PAE+SAP, $N = 40$. The horizontal black line is the constraint.

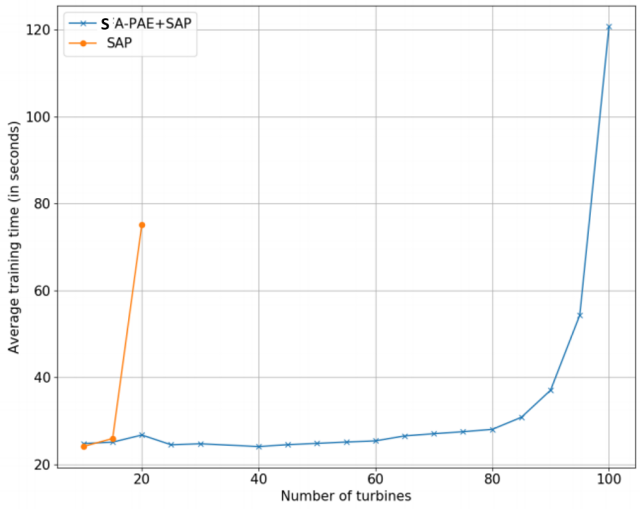

Figure 9: Comparing the CPA and BHS power consumed as a function of k with SA-PAE+SAP, $N = 40$. The horizontal black line is the constraint.

the action into a space of reduced dimension. This novel technique allows the agent to perform better both in terms of performance and scale compared to an agent without any embedding.

7.1 Comparison with coordinated graphs methods

In [9], the authors use coordination graphs to simplify the action space in a centralized MARL. Similarly, in [13], the sparsity in coordination graphs is encoded to reduce the dimension of action space. Both the methods require a knowledge of the coordination graph. In contrast, SA-PAE can be considered a method where the intrinsic structure within the action space is learned in a state-aware manner and encoded simultaneously. We believe, our method is much more generic and easy to implement.

7.2 Comparison with Independent Q-learning and related methods

As showed in [20, 26, 31], ignoring the Markov property and having each agent learn independently is a technique which has been successful in several applications. However, in a constrained scenario like ours, the policy so obtained is very conservative i.e. each agent operates in a fearful fashion to avoid breaching the overall constraint. In another line of work like QMIX [23], VDN [29] etc. which assumes centralized training and decentralized policy making, the constrained environment again leads to overtly conservative policies.

7.3 Comparison with action representations proposed in [5]

This work became visible to us very recently and by that time most of the current manuscript was completed. As a result, we were unable to do a proper comparison with the methods proposed in the paper. As this is the only other work which does not assume apriori information on the action representations, we believe a thorough comparison is necessary and we intend to complete that exercise in a future extensions if this work.

7.4 Future Work

There are several research directions which need to be investigated for efficiently using RL for similar problems.

(1) A more rigorous investigation of relation between number of agents N, sparsity p and reduced dimension k is needed to understand the performance limits and optimal setting.
(2) A state-aware non-linear or manifold embedding might capture the structure of the action space better and needs further exploration.
(3) Another area of research is learning policies for a constrained cooperative MARL in a decentralized scenario i.e. each agent independently learns an optimal policy and satisfies the global constraint simultaneously. This directs towards an Empathetic Reinforcement Learning framework.

REFERENCES

[1] Cynthia Barnhart, Peter Belobaba, and Amedeo R Odoni. 2003. Applications of operations research in the air transport industry. *Transportation science* 37, 4 (2003), 368–391.
[2] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940* (2016).
[3] Dimitris Bertsimas and Melvyn Sim. 2003. Robust discrete optimization and network flows. *Mathematical programming* 98, 1-3 (2003), 49–71.
[4] Margaret L Brandeau, François Sainfort, and William P Pierskalla. 2004. *Operations research and health care: a handbook of methods and applications.* Vol. 70. Springer Science & Business Media.
[5] Yash Chandak, Georgios Theocharous, James Kostas, Scott M. Jordan, and Philip S. Thomas. 2019. Learning Action Representations for Reinforcement Learning. *CoRR* abs/1902.00183 (2019). arXiv:1902.00183 http://arxiv.org/abs/1902.00183
[6] Yao-Nan Chen and Hsuan-Tien Lin. 2012. Feature-aware label space dimension reduction for multi-label classification. In *Advances in Neural Information Processing Systems*. 1529–1537.

[7] Gabriel Dulac-Arnold, Richard Evans, Peter Sunehag, and Ben Coppin. 2015. Reinforcement Learning in Large Discrete Action Spaces. *CoRR* abs/1512.07679 (2015). arXiv:1512.07679 http://arxiv.org/abs/1512.07679

[8] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2017. Counterfactual Multi-Agent Policy Gradients. *CoRR* abs/1705.08926 (2017). arXiv:1705.08926 http://arxiv.org/abs/1705.08926

[9] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. 2002. Coordinated reinforcement learning. In *ICML*, Vol. 2. Citeseer, 227–234.

[10] Juraj Hromkovič. 2013. *Algorithmics for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics*. Springer Science & Business Media.

[11] Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. 2017. Solving a new 3d bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930* (2017).

[12] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. 1983. Optimization by simulated annealing. *science* 220, 4598 (1983), 671–680.

[13] Jelle R Kok and Nikos Vlassis. 2004. Sparse cooperative Q-learning. In *Proceedings of the twenty-first international conference on Machine learning*. 61.

[14] Fayçal Lamraoui, Guy Fortin, Robert Benoit, Jean Perron, and Christian Masson. 2014. Atmospheric icing impact on wind turbine production. *Cold Regions Science and Technology* 100 (2014), 36–49.

[15] Alexandre Laterre, Yunguan Fu, Mohamed Khalil Jabri, Alain-Sam Cohen, David Kas, Karl Hajjar, Hui Chen, Torbjørn S Dahl, Amine Kerkeni, and Karim Beguir. 2019. Ranked Reward: Enabling Self-Play Reinforcement Learning for Bin packing. (2019).

[16] Eugene L Lawler and David E Wood. 1966. Branch-and-bound methods: A survey. *Operations research* 14, 4 (1966), 699–719.

[17] Zijia Lin, Guiguang Ding, Mingqing Hu, and Jianmin Wang. 2014. Multi-label classification via feature-aware implicit label space encoding. In *International conference on machine learning*. 325–333.

[18] Sadayoshi Mikami and Yukinori Kakazu. 1994. Genetic reinforcement learning for cooperative traffic signal control. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. IEEE, 223–228.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[21] Olivier Parent and Adrian Ilinca. 2011. Anti-icing and de-icing techniques for wind turbines: Critical review. *Cold regions science and technology* 65, 1 (2011), 88–96.

[22] Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[23] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2020. Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2003.08839* (2020).

[24] David A Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. 2008. Incremental learning for robust visual tracking. *International journal of computer vision* 77, 1-3 (2008), 125–141.

[25] Gavin A Rummery and Mahesan Niranjan. 1994. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK.

[26] Sandip Sen, Mahendra Sekaran, John Hale, et al. 1994. Learning to coordinate without sharing information. In *AAAI*, Vol. 94. 426–431.

[27] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Hostallero, and Yung Yi. 2019. QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning. *CoRR* abs/1905.05408 (2019). arXiv:1905.05408 http://arxiv.org/abs/1905.05408

[28] Gilbert Strang, G Strang, G Strang, and G Strang. 1993. Introduction to linear algebra, vol. 3. *Wellesley-Cambridge Press Wellesley, MA* 42 (1993).

[29] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. 2018. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th international conference on autonomous agents and multiagent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2085–2087.

[30] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[31] Ming Tan. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*. 330–337.

[32] Grigorios Tsoumakas and Ioannis Katakis. 2007. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)* 3, 3 (2007), 1–13.

[33] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in neural information processing systems*. 2692–2700.

[34] Harvey M Wagner. 1975. *Principles of operations research: with applications to managerial decisions*. Technical Report. Prentice-Hall Englewood Cliffs, NJ.

[35] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[36] Wei Zhang and Thomas G Dietterich. 2000. Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. *Journal of Artificial Intelligence Reseach* 1 (2000), 1–38.