Control Engineering Practice 97 (2020) 104331

Contents lists available at ScienceDirect

Control Engineering Practice

journal homepage: www.elsevier.com/locate/conengprac

Scalable reinforcement learning for plant-wide control of vinyl acetate monomer process

Lingwei Zhu [a,*,1], Yunduan Cui [a,1], Go Takami [b], Hiroaki Kanokogi [b], Takamitsu Matsubara [a]

[a] Graduate School of Science and Technology, Nara Institute of Science and Technology, Takayama-cho 8916-5, Ikoma, Nara, Japan
[b] New Field Development Center, Yokogawa Electric Corporation, Nakacho 2-9-32, Musashino-shi, Tokyo, Japan

ARTICLE INFO

Keywords:
Chemical process control
Reinforcement learning
Vinyl acetate monomer

ABSTRACT

This paper explores a reinforcement learning (RL) approach that designs automatic control strategies in a large-scale chemical process control scenario as the first step for leveraging an RL method to intelligently control real-world chemical plants. The huge number of units for chemical reactions as well as feeding and recycling the materials of a typical chemical process induces a vast amount of samples and subsequent prohibitive computation complexity in RL for deriving a suitable control policy due to high-dimensional state and action spaces. To tackle this problem, a novel RL algorithm: Factorial Fast-food Dynamic Policy Programming (FFDPP) is proposed. By introducing a factorial framework that efficiently factorizes the action space, Fast-food kernel approximation that alleviates the curse of dimensionality caused by the high dimensionality of state space, into Dynamic Policy Programming (DPP) that achieves stable learning even with insufficient samples. FFDPP is evaluated in a commercial chemical plant simulator for a Vinyl Acetate Monomer (VAM) process. Experimental results demonstrate that without any knowledge of the model, the proposed method successfully learned a stable policy with reasonable computation resources to produce a larger amount of VAM product with comparative performance to a state-of-the-art model-based control.

1. Introduction

Chemical processes that are comprised of a large number of sensors and control units require the derivation of complex models for the environment and control laws based on it Dotoli, Fay, Miśkowicz, and Seatzu (2015) and Metzger and Polakow (2011). Obtaining an accurate model is difficult in practice for large-scale processes. As a response to the growing model complexity of such model-based control, model-free approaches as adaptive control, statistical process control methods, and agent-based approaches are naturally emerging (Kano & Nakagawa, 2008; Ng & Srinivasan, 2010; Qian, Huang, Lin, & Li, 2000; Wang & Zhang, 2008; Yang et al., 2008; Zhang, Kano, & Li, 2017).

Reinforcement Learning (RL) (Badgwell, Lee, & Liu, 2018; Sutton & Barto, 1998) is a potentially powerful tool in chemical process control problems since it does not need a model of the plant for control design; it can learn a control policy by trial-and-error interactions with the environment without supervision from experienced operators. Applications in process control fields range from electricity grid control to dynamic power management (Ernst, Glavic, Geurts, & Wehenkel, 2005; Hoskins & Himmelblau, 1992; Lee & Wong, 2010; Liu, Tan, & Qiu, 2010). In the chemical process control context, Hoskins and Himmelblau (1992) applied RL to control a continuous-stirred-tank reactor (CSTR). Syafiie, Tadeo, and Martinez (2007) leveraged RL in a classical pH neutralization control task in a laboratory plant. In other process control problems outside of chemical plants, Liu et al. (2010) used an enhanced Q-learning algorithm to derive a policy that achieves better power-performance tradeoff on both synthetic and small-sized, real workloads. Harp, Brignone, Wollenberg, and Samad (2000) embedded an RL algorithm in a simulator for electricity management to learn a profitable electricity pricing policy. Although these methods achieved success in low-dimensional simulations or small-scaled, real-world experiments, successfully applying RL in chemical plants remains difficult. The above studies mostly employed Q-learning (Watkins & Dayan, 1992), which does not scale well in high-dimensional problems, and is typically expensive in terms of both learning time and computational resources for high-dimensional systems. Deep RL (DRL) techniques on process control (Kubosawa, Onishi, & Tsuruoka, 2018; Spielberg, Gopaluni, & Loewen, 2017), such as the Deep Q-Network (DQN), usually have sample complexity that is several magnitudes larger than traditional RL. Such prohibitive sample complexity excludes them from consideration for plant-wide process control. This problem might explain why (Kubosawa et al., 2018) only experimented with isolated components of the Vinyl Acetate Monomer (VAM) manufacturing

* Corresponding author.
E-mail address: zhu.lingwei.zj5@is.naist.jp (L. Zhu).
[1] These authors contributed equally to this paper.

https://doi.org/10.1016/j.conengprac.2020.104331
Received 23 April 2019; Received in revised form 7 January 2020; Accepted 28 January 2020
Available online 10 February 2020

process from malfunctioning without considering the full context of the problems with the VAM process control, e.g., improving the production rate or product quality (see Discussion for further details).

To the best of our knowledge, few studies have applied RL algorithms to large-scale chemical plant control. The main reasons are intractable computation and the memory cost incurred by such an extremely large number of samples, subsequent derivation for feasible policies, and the poor scalability of traditional RL algorithms, e.g., Q-learning. Traditional RL algorithms suffer from the curse of dimensionality (Bellman, 2003), which refers to the explosion of the number of samples needed as dimensionality increases.

In this paper, Factorial Fast-food Dynamic Policy Programming (FFDPP), a novel scalable reinforcement learning framework for large-scale chemical plants is proposed. FFDPP inherits the smooth policy update of Dynamic Policy Programming (Azar, Gómez, & Kappen, 2012), which achieves stable learning even with insufficient samples in high-dimensional state–action spaces, by considering the Kullback–Leibler divergence between current and new policies as a regularization term, as demonstrated in Cui, Matsubara, and Sugimoto (2017a, 2017b) and Tsurumine, Cui, Uchibe, and Matsubara (2019). To further enhance its scalability for high-dimensional systems such as chemical plants, two effective approximation schemes are exploited: factorial policy representation with factor-wise policy updates (Cui, Zhu, Fujisaki, Kanokogi, & Matsubara, 2018; Matsubara, Gómez, & Kappen, 2014) and Fast-food kernel approximation (Le, Sarlos, & Smola, 2013). Factorial policy representation allows for vast discrete action spaces to be factorized into smaller action sets (Cui et al., 2018; Matsubara et al., 2014), inspired by the fact that conventional heuristic strategies often independently handle each control parameter. This assumption tends to hold true in process control problems since the units that serve different purposes are designed to be independent. Fast-food kernel approximation samples in the frequency domain approximate feature maps (Le et al., 2013) instead of direct sampling in the original space that alleviates the curse of dimensionality.

To investigate the effectiveness of FFDPP for large-scale chemical process control, a simulator of a VAM manufacturing process. Proposed by Luyben and Tyréus (1998) is used, the VAM process is a benchmark problem for testing chemical process design with the following features:

1. a realistically large process flowsheet that contains standard chemical unit operations;
2. a process with the typical industrial characteristics of recycle streams and energy integration;
3. actual, non-ideal chemical components.

The VAM model was further improved by Chen, Dave, McAvoy, and Luyben (2003), Luyben (2011), Olsen, Svrcek, and Young (2005) and Seki et al. (2010). Based on those improvements, Machida et al. (2016) developed a simulation model with a state-of-the-art model-based controller under which the process operates in stable conditions. Experimental results confirmed that the policies learned by FFDPP can achieve comparative performance to the model-based controller without knowledge of the plant model.

The rest of the paper is organized as follows. the preliminary is illustrated in Section 2, followed by a detailed description of the proposed method in Section 3. The VAM process and our objective are shown in Section 4. Simulation results are examined in Section 5. Discussion is described in Section 6, and the conclusion follows in Section 7.

2. Preliminary

2.1. Reinforcement learning

RL models problems as Markov Decision Processes (MDPs) expressed by a quintuple $(S, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. S denotes the state space, $\mathcal{A}$ denotes the action space, and $\mathcal{T}^a_{ss'}$ is the transition from state s to s' by action a. $\mathcal{R} = r^a_{ss'}$ is the immediate reward under that transition, and $\gamma \in (0, 1)$ is the discount factor.

Given state s, value function V_π following specified policy π from that state is defined as the cumulative discounted reward:

$$V_\pi(s) = \mathrm{E}_{\pi,\mathcal{T}}\left[\sum_{t=0}^{\infty} \gamma^t r_{s_t} \,\middle|\, s_0 = s\right], \tag{1}$$

where s_t is the state at time step t and r_{s_t} is the reward at state s_t. $E_{\pi,\mathcal{T}}$ is averaged over π and transition probability $\mathcal{T}$.

RL methods search for optimal policy π^* to maximize (1), i.e., to satisfy the recursive Bellman equations:

$$V_{\pi^*}(s) = \max_\pi \sum_{\substack{a\in\mathcal{A}\\ s'\in S}} \pi(a|s)\left[\mathcal{T}^a_{ss'}\left(r^a_{ss'} + \gamma V_{\pi^*}(s')\right)\right]. \tag{2}$$

It is also convenient to use Q_π to denote the value function of state–action pairs $(s, a) \in S \times \mathcal{A}$ under policy π:

$$Q_{\pi^*}(s,a) = \max_\pi \sum_{s'\in S} \mathcal{T}^a_{ss'}\left(r^a_{ss'} + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') Q_{\pi^*}(s', a')\right). \tag{3}$$

2.2. Dynamic policy programming

Dynamic Policy Programming (DPP) (Azar et al., 2012) is a value function-based RL algorithm. Due to its theoretically proven convergence, sample efficiency, and stability in learning, DPP has many extensions from complex robot-arm manipulation (Cui et al., 2017a) to DRL-based robot control (Tsurumine et al., 2019) and inverse RL (Uchibe, 2018). DPP enforces the stability of policy updates by adding a Kullback–Leibler (KL) divergence term between the baseline policy and the current policy to restrain the agent from taking aggressive steps:

$$KL\big(\pi(\cdot|s) \,\|\, \bar\pi(\cdot|s)\big) = \sum_{a\in\mathcal{A}} \pi(a|s) \log\left(\frac{\pi(a|s)}{\bar\pi(a|s)}\right). \tag{4}$$

A new optimal value function is obtained by incorporating Eq. (4) into (2):

$$V_{\pi^*}(s) = \max_\pi \sum_{\substack{a\in\mathcal{A}\\ s'\in S}} \pi(a|s)\left[\mathcal{T}^a_{ss'}\left(r^a_{ss'} + \gamma V_{\pi^*}(s')\right) - \frac{1}{\eta}\log\left(\frac{\pi(a|s)}{\bar\pi(a|s)}\right)\right], \tag{5}$$

where η is the inverse temperature parameter that weights the effect of the KL term.

Following (Azar, Gómez, & Kappen, 2011; Todorov, 2006), the new optimal value function and policy satisfy the recursive equations:

$$V^{\pi^*}_{\bar\pi}(s) = \frac{1}{\eta}\log \sum_{a\in\mathcal{A}} \bar\pi(a|s) \exp\left[\eta \sum_{s'\in S} \mathcal{T}^a_{ss'}\left(r^a_{ss'} + \gamma V^{\pi^*}_{\bar\pi}(s')\right)\right], \tag{6}$$

$$\bar\pi^*(a|s) = \frac{\bar\pi(a|s)\exp\left[\eta \sum_{s'\in S} \mathcal{T}^a_{ss'}\left(r^a_{ss'} + \gamma V^{\pi^*}_{\pi}(s')\right)\right]}{\exp\left(\eta V^{\pi^*}_{\pi}(s)\right)}. \tag{7}$$

The optimal value function and policy need to be obtained iteratively. By repeatedly replacing the baseline policy in Eq. (6), the optimal value function can be asymptotically obtained. We define an action preference function (Sutton, 1996) and denote iteration t:

$$P_{t+1}(s,a) = \frac{1}{\eta}\log \bar\pi^t(a|s) + \sum_{s'\in S} \mathcal{T}^a_{ss'}\left(r^a_{ss'} + \gamma V^t_{\bar\pi}(s')\right) \tag{8}$$

and compare (8) with (6) and (7), and so the value function and policy at iteration t become:

$$V^t_{\bar\pi}(s) = \frac{1}{\eta}\log \sum_{a\in\mathcal{A}} \exp\left(\eta P_t(s,a)\right), \tag{9}$$

$$\bar\pi^t(a|s) = \frac{\exp\left(\eta P_t(s,a)\right)}{\sum_{a'\in\mathcal{A}} \exp\left(\eta P_t(s,a')\right)}. \tag{10}$$

Substituting (9) and (10) back into (5), the update rule for the action preference function is derived:

$$\begin{aligned} P_{t+1}(s,a) &= \mathcal{O}P_t(s,a) \\ &= P_t(s,a) - \mathcal{M}_\eta P_t(s) + \sum_{s'\in S} \mathcal{T}^a_{ss'}(r^a_{ss'} + \gamma \mathcal{M}_\eta P_t(s')), \end{aligned} \tag{11}$$

where $\mathcal{M}_\eta P_t = \sum_{a\in\mathcal{A}} \frac{\exp(\eta P_t(s,a))P_t(s,a)}{\sum_{a'\in\mathcal{A}}\exp(\eta P_t(s,a'))}$ is the Boltzmann softmax operator defined to replace the log-partition-sum in (9). $\mathcal{O}$ is the DPP operator, and $\mathcal{O}P \in \mathcal{R}^{|S||\mathcal{A}|\times 1}$ is the action preference vector for all the state–action pairs. The optimality of the preference function is obtained when $t \to \infty$.

The original DPP is only applicable to small-sized problems with discrete states and model knowledge. To derive model-free learning with continuous states, function approximation is introduced (Sutton, McAllester, Singh, & Mansour, 1999). For example, linear function approximation defines a feature map as $\Phi=[\phi(x_1),\dots,\phi(x_N)]^T$, where N denotes the number of samples and $\phi(x_n)=[\varphi_1(x_n),\dots,\varphi_M(x_n)] \in \mathcal{R}^M$. x denotes sample-combining state s and action a. φ is the basis function. The action preference at iteration t is approximated by Φ and weight vector: $P_t = \Phi\theta_t$. Weight vector θ_t is obtained by iteratively solving the least-square problem that minimizes the loss between the true action preference and approximation $J \triangleq \|\mathcal{O}P - \Phi\theta\|^2$:

$$\theta_t^* = \arg\min_{\theta_t} J(\theta_t) = [\Phi_t^T\Phi_t + \sigma^2 I]^{-1}\Phi_t^T\mathcal{O}\hat{P}_t, \tag{12}$$

where θ_t^* is the minimizer of $J(\theta)$ at iteration t and σ is a regularization term that avoids overfitting due to a limited number of samples. For simplicity subscript t of the weight vector for approximating the action preference is dropped in later sections, i.e., $P_t = \Phi\theta$.

DPP with simple function approximation remains inefficient in computing feature map Φ in high-dimensional spaces. The traditional approach for computing a feature map is to set a grid uniformly over the space and evaluate at each grid point (Azar et al., 2011). This scales poorly in high-dimensional space since the number of grid points needed to cover the space increases exponentially with dimensionality. This dilemma is known as the curse of dimensionality. Potential solutions include replacing the analytic expression of Φ by approximation and decomposing the action set into subsets.

3. Proposed method

In this section the proposed method, Factorial Fast-food DPP (FFDPP) is introduced. FFDPP is a framework that unifies the efficient approximation of feature map Φ and the factorization of large-scale discrete action sets by DPP.

3.1. Factorial policy with factor-wise smooth update

In the proposed method, without loss of generality every action $a \in \mathcal{A}$ is encoded as $a = [a^{(1)},\dots,a^{(M)}]^T$, where entry $a^{(m)}$ denotes control parameter m and has a discrete set of action elements $[a^{(m)}_{(1)},\dots,a^{(m)}_{(N)}]$. For simplicity, every entry $a^{(m)}$ is assumed to be a set of identical size. Generally, to take full consideration of M control parameters, each with N discrete action requires searching among $|\mathcal{A}| = M^N$ candidates. The corresponding computation easily becomes intractable with an increasingly large M and N. One typical solution is using a smaller hand-engineered action set based on human knowledge (Cui et al., 2017a). However, this may significantly undermine the control flexibility.

Inspired by Cui et al. (2018) and Matsubara et al. (2014), it is reasonable to assume that the policy can be factored into components that are only comprised of a small subset of actions, s.t. the conditional independence conditioned on the states between sub-policies holds:

$$\begin{aligned} &\hat{\pi}(a|s) = \prod_{m=1}^{M}\pi(a^{(m)}|s), \\ \textbf{s.t.}\quad &\pi(a^{(i)}|s) \perp \pi(a^{(j)}|s), \quad \forall i,j \\ &\pi(a|s) \approx \hat{\pi}(a|s), \end{aligned} \tag{13}$$

where $\perp$ denotes the independence between two random variables. By factorizing the action set into several independent sub-action sets, a multi-agent learning framework is defined under which each agent learns different value function $V_\pi^{(m)}$ and policy $\pi^{(m)}$, where superscript m denotes the agent that is handling the m-th subset. The multi-agent framework factorizes the total M^N possible actions to be handled by M agents, each of which has N discrete actions. Thus, the total number of actions that must be considered is reduced to MN. All M agents use the same state space and run in lock-step. The output from each factorized controller is then grouped in the sense that $a = [a^{(1)},\dots,a^{(M)}]$ to produce a unique action and subsequently a unique next state. All controllers are updated simultaneously and have access to the whole state. The system's Markov property is hence preserved.

The factorized value functions and policies are the factored counterparts to Eqs. (6) and (7). In what follows, superscript m of the factorized action in action preference $P^{(m)}$ is omitted for uncluttered notations. The factorized action preferences can be expressed:

$$P^{(m)}_{t+1}(s,a) = \frac{1}{\eta}\log\bar{\pi}^t(a^{(m)}|s) + \sum_{s'\in S}\hat{\mathcal{T}}^a_{ss'}\left(r^a_{ss'} + \gamma V^{t^{(m)}}_{\bar{\pi}}(s')\right), \tag{14}$$

where $\hat{\mathcal{T}}^a_{ss'}$ is a transition model that is different from the standard definition (8). The difference results from the factorial policy. Instead of taking action a and transitioning to the next state s', one generates m sub-actions independently and groups them for the transition.

Following the standard definition of DPP, the n-th sub-action is sampled from:

$$\bar{\pi}^{(m)}_t(a|s) = \frac{\exp\left(\eta P^{(m)}_t(s,a)\right)}{\sum_{a'\in\mathcal{A}^{(m)}}\exp\left(\eta P^{(m)}_t(s,a')\right)}. \tag{15}$$

In a function approximation scenario where the action preference is approximated with a weight and a feature map, namely, $P_t = \Phi\theta$, feature map Φ of size $|\mathcal{M}|\times|S||\mathcal{A}|$ is factorized into M smaller ones of size $|\mathcal{M}|\times|S||\mathcal{A}^{(m)}|$:

$$\Phi \in \mathcal{R}^{M\times S\mathcal{A}} \overset{\text{Factorize}}{\Longrightarrow} \Phi^{(n)} \in \mathcal{R}^{M\times S\mathcal{A}^{(m)}},\ n\in[1,2,\dots,N], \tag{16}$$

where $\Phi^{(m)}$ represents the feature map of the mth agent. Thus, with the function approximation the action preference for the m-th action set is updated:

$$P^{(m)}_{t+1}(s,a) = P^{(m)}_t(s,a) + r^a_{ss'} - \mathcal{M}_\eta P^{(m)}_t(s) + \gamma\mathcal{M}_\eta P^{(m)}_t(s'). \tag{17}$$

Consequently, we solve M regression problems similar to Eq. (12) to obtain the M weight vectors at iteration t:

$$\theta^{(m)^*}_t = \arg\min_{\theta^{(m)}_t} J(\theta^{(m)}_t) = [\Phi^{(m)^T}_t\Phi^{(m)}_t + \sigma^2 I]^{-1}\Phi^{(m)^T}_t\mathcal{O}\hat{P}^{(m)}_t. \tag{18}$$

3.2. Fast-food approximation

Le et al. (2013) proposed Fast-food to approximate kernel expansion with only $\mathcal{O}(n\log d)$ time and $\mathcal{O}(n)$ storage, where n denotes the number of samples and d denotes the dimensionality. In our previous work (Cui et al., 2018) the features of Factorial Kernel Dynamic Policy Programming (FKDPP) were designed by selecting a subset of the most informative samples and computing the approximated kernel matrices from the subset and whole samples. However, when the dimensionality increases, the size of the subset also increases drastically, quickly rendering computation intractable. On the other hand, Fast-food solves this problem by transferring the data to the frequency domain to conduct the sampling. It is mathematically demonstrated that generated feature map $\hat{\Phi}$ recovers the exact Φ with exponentially decreasing error with increasing sampling frequency (Rahimi & Recht, 2008). This is especially suitable for computing high-dimensional problems. By leveraging Fast-food, the DPP-based value function and policies (9) and (10) are now replaced by their approximated counterparts:

$$\hat{V}^t_{\bar{\pi}}(s) = \frac{1}{\eta}\log\sum_{a\in\mathcal{A}}\exp\left(\eta\hat{P}_t(s,a)\right), \tag{19}$$

$$\hat{\pi}^t(a|s) = \frac{\exp\left(\eta\hat{P}_t(s,a)\right)}{\sum_{a'\in\mathcal{A}}\exp\left(\eta\hat{P}_t(s,a')\right)}, \tag{20}$$

Algorithm 1: Factorial Fast-food Dynamic Policy Programming

Input: $M, \tau, \gamma, \eta, \sigma, T, I$
Output: weight vectors $\theta^{(m)}, m = 1, \dots, M$

1 Divide $\mathcal{A}$ into $\{\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(\mathcal{M})}\}$
2 Random exploration with $\pi_{random} = [\pi_{random}^{(1)}, \dots, \pi_{random}^{(M)}]^T$
3 Compute Fast-food matrix V using τ
4 **for** $i = 1, \dots, I$ **do**
5 **for** $t = 1, \dots, T$ **do**
6 *measure state* s_t
7 **for** $m = 1, \dots, M$ **do**
8 $x^{(m)} = [s_t, a_t^{(m)}]^T$
9 *compute* $\phi^{(m)}(x)$ *using Fast-food*
10 *compute* $\hat{P}_t^{(m)}$ *following* (8)
11 *sample action* $a_t^{(m)}$ *from* (20)
12 *collect* $x^{(m)}$ *in* $\mathcal{D}_i^{(m)}$
13 **end**
14 *apply action* $a_t = [a_t^{(1)}, \dots, a_t^{(M)}]$
15 **end**
16 **for** $m = 1, \dots, M$ **do**
17 *compute* $\hat{\Phi}_t^{(m)}$ *using* $\mathcal{D}_i^{(m)}$
18 $\hat{\theta}_{t+1}^{(m)} = [\hat{\Phi}_t^{(m)^T} \hat{\Phi}_t^{(m)} + \sigma^2 I]^{-1} \hat{\Phi}_t^{(m)^T} \mathcal{O} \hat{P}_t^{(m)}$
19 **end**
20 **end**

where $\hat{P}_t = \hat{\Phi}\theta$. The exact quantities and their approximated counterparts are distinguished by the hat symbol $\hat{(\cdot)}$ that represents the Fast-food approximation. The Fast-food DPP update rule is now given by:

$$\hat{P}_{t+1}(s,a) = \hat{P}_t(s,a) + r_{ss'}^a - \mathcal{M}_\eta \hat{P}_t(s) + \gamma \mathcal{M}_\eta \hat{P}_t(s'). \tag{21}$$

Solving the least-square problem for the optimal weight vector, as in Sections 3.1, 2.2 yields:

$$\hat{\theta}^* = \arg\min_{\hat{\theta}} J(\hat{\theta}) = [\hat{\Phi}^T \hat{\Phi} + \sigma^2 I]^{-1} \hat{\Phi}^T \mathcal{O} \hat{P}_t. \tag{22}$$

3.3. Factorial fast-food DPP

The above components are unified in Alg. 1. The system is assumed to start from fixed initial state s_0. Lines 1 and 2 establish the factorial multi-agent framework and initialize each agent with an exploration policy. At the first iteration, the exploration policy is set to randomly explore the unseen environment, which is accomplished by all agents independently sampling from $\pi_{random}^{(m)}$. The Fast-food approximation feature map is computed using user-design parameter τ in line 3. The user must specify number of episodes I for the learning loop beginning at line 4. For every time step t and every agent m, a sample is encountered, and for the corresponding feature vector, action preferences are computed. Then the sample is stacked in sample pool $\mathcal{D}_i^{(m)}$ for a policy update after the episode. Lines 14 to 17 correspond to the FFDPP training that combines (18) and (22) to produce weight vectors that are the approximated counterparts of (12). After the learning, all the factorial weight vectors are combined to produce a learned near-optimal policy:

$$\pi^*(a|s) = [\pi^*(a^{(1)}|s), \dots, \pi^*(a^{(M)}|s)]^T. \tag{23}$$

4. VAM manufacturing process control problem

4.1. VAM process description

The VAM manufacturing process is a popular benchmark problem for many chemical tests due to its realistic complexity and appropriate design (Chen et al., 2003; Luyben, 2011; Machida et al., 2016; Olsen et al., 2005; Seki et al., 2010). As a powerful tool for training operators, the Visual Modeler simulator (Omega Simulation Corp.) can duplicate the VAM manufacturing process. Following the process diagram shown in Fig. 1, the functions of each part are briefly introduced. Further details can be found in Machida et al. (2016) and Seki et al. (2010).

Part 1 is the entry of three types of raw materials: ethylene(C_2H_4), oxygen(O_2), and acetic acid(CH_3COOH).

Part 2 is where the main reaction takes place. Raw materials are converted to vinyl acetate and water as main products, with carbon dioxide as a byproduct.

$$C_2H_4 + CH_3COOH + \tfrac{1}{2}O_2 \longrightarrow CH_2 = CHOCOCH_3 + H_2O,$$
$$C_2H_4 + 3O_2 \longrightarrow 2CO_2 + 2H_2O$$

Here the = indicates a double chemical bond.

Part 3 is composed of a cooler for removing heat generated by exothermic reactions, a separator for separating liquid VAM crude from unreacted gaseous raw materials, and a compressor for circulation.

Part 4 is composed of an absorber to extract gaseous VAM and discharge it to the buffer tank (Part 6).

Part 5 is a gas-purge system that maintains the process's pressure.

Part 6 is a buffer tank that stores the VAM semi-product, which is fed to the distillation column.

Part 7 is an azeotropic distillation column that extracts the VAM-water mixture and cycles the acetic acid back to parts 1 and 4.

Part 8 is a decanter that pours off the final VAM product.

The VAM manufacturing process produces high quality VAM product while maintaining plant-wide stability. This is achieved by operators who are manually coordinating the *control* units, mainly PID controllers from every part of the process. Process stability is defined in terms of the readings of several indicators that monitor for anomalies.

4.2. Formulation of control problems

One objective of the investigated VAM process control problem is to improve the VAM process yield and product quality using control and observation units from the distillation column (part 7) and the decanter (part 8). The optimization objective is defined *profit*:

$$\text{Profit} = a * \text{yield} + b * \text{quality} - c * \text{cost}, \tag{24}$$

where a, b, c are the constants. A model-based controller was derived in Machida et al. (2016) and Seki et al. (2010) under which the VAM process runs in stable conditions and outputs a constant amount of VAM with consistent quality. In what follows, without possible confusion, we shall abbreviate the state-of-the-art model-based controller in Machida et al. (2016) and Seki et al. (2010) as the *model-based controller*. The target is to apply RL to learn a model-free policy that yields profit that is comparable to the model-based controller.

Compared with previous work (Cui et al., 2018) in which only one part of the process was activated and optimized, this paper studies a situation where all eight parts were activated. As a result of complete activation, the plant becomes more sensitive because any perturbation from a single control unit might quickly spread to all the other parts and induce unpredictable changes.

Detailed definitions of the observation and control units investigated in this paper are given in Tables 1(a) and 1(b), and complementary descriptions of the units are depicted here. The product quality measures, *quality sensor 1* and *quality sensor 2* in Table 1(a), whose readings are inversely proportional to the product quality, are constrained to stay in a tighter range compared to previous research. Eight temperature sensors monitor the temperature inside the distillation column. The last dimension of the state is the *gross profit* of the VAM product in Japanese yen (JPY) per hour. Note that this is only the gross profit; since the product quality and the process stability have not been taken into account, they cannot be directly used as a reward for learning. The steam amount, which is part of the cost, is

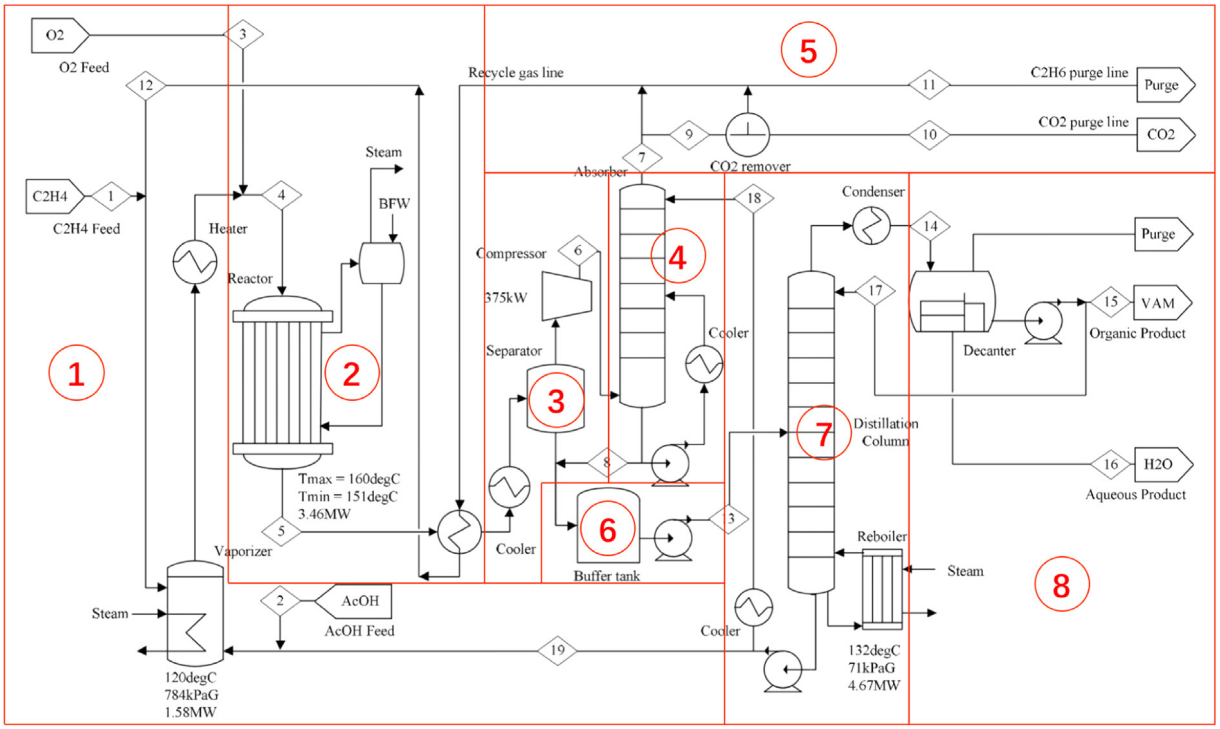


Fig. 1. Flow diagram of VAM process.

Table 1
Observation and control units of investigated task.

(a) Observation units of investigated task

Observation	Description	Control criteria
Production rate sensor	Flow rate of VAM	Optimized as VAM yield, part of *profit*
Level sensor	Level(%) of decanter	<100%
Quality sensor 1	Acetic acid density	<100 ppm
Quality sensor 2	VAM density (inversely proportional to the quality)	<150 ppm
Temperature sensor 1–8	Temperature inside the distillation column	Optimized as cost, part of the *profit*
Gross profit	Profit of VAM product (JPY/h)	Optimized as *gross profit* (part of the *profit*)

(b) Control units of investigated task

Control unit	Description	Effect
Flow controller 1	Control superheated stream flow rate for the reboiler	Column temperature rises/declines Affect VAM yield and quality Steam amount increases/decreases Steam generation is part of cost
Pressure controller 1	Maintain top pressure of the column with N_2 and gas purge	Acetic acid density rises/declines N_2 density rises/declines
Pressure controller 2	Maintain pressure of 3rd stage of distillation column with superheated steam	Column temperature rises/declines Affect VAM yield and quality Steam amount increase/decreases Steam generation is part of cost
Flow controller 2	Control decanter feed flow temperature with cooling water	Control decanter feed flow Affect stability of process
Temperature controller	Maintain temperature profile and product quality by controlling reflux flow rate	Affect VAM yield and quality Affect stability of process Control reverse flow rate

directly related to the physical profile of the column. Flow controller 1 and pressure controllers 1 and 2 manage the temperature, the flow, and the pressure of the distillation column. Flow controller 2 and the temperature controller manage the temperature of the decanter's feed and the reflux flow. By manipulating the scalar property of the controller, its behavior is affected. In the rest of the paper, the reader shall refer to the name of the investigated controllers by their scalar properties.

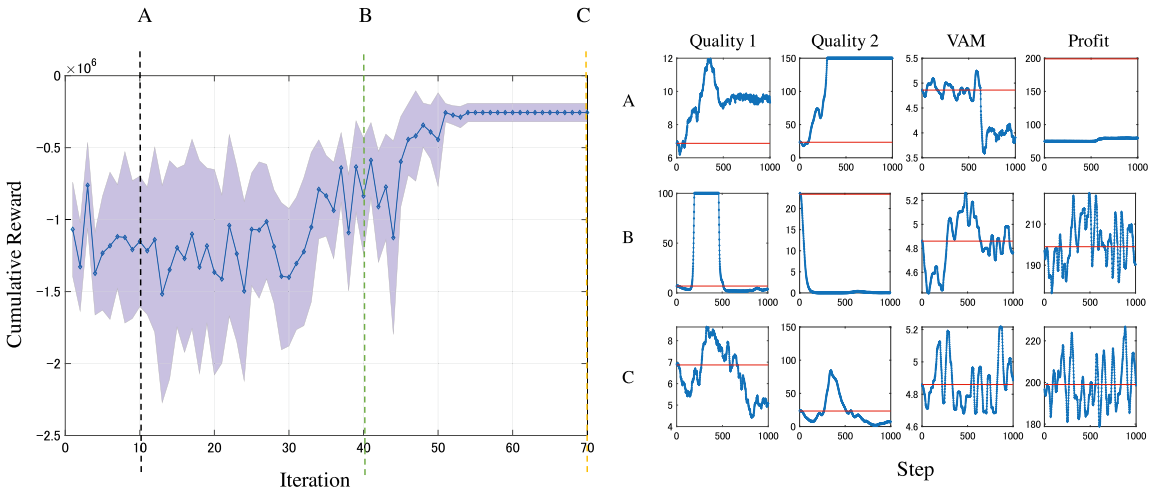


Fig. 2. Average cumulative rewards averaged over ten independent experiments with $\tau = 100$. Black dashed line marked by *A* shows performance at 10th iteration; green dashed line *B* is performance at 40th iteration; yellow line *C* is performance at 70th iteration. Right-hand subplots on *Y*-axis are *Quality 1*-(ppm), *Quality 2*-(ppm), *VAM (production rate)*-(t/h), and *Profit (gross profit)*-(10^3 yen/h). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

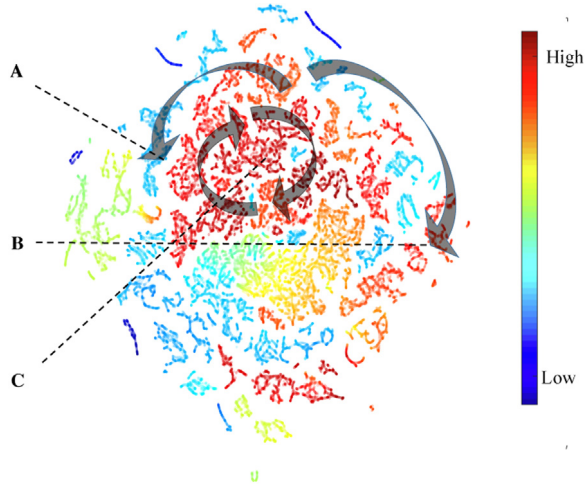


Fig. 3. T-SNE plot shows a trajectory of agent in one experiment. Undesirable states, i.e., overflow of quality sensors 1 or 2, low production rate, etc., are marked with blue, and desirable states are marked in red. Markers *A,B,C* refer to performance plots in Fig. 2. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

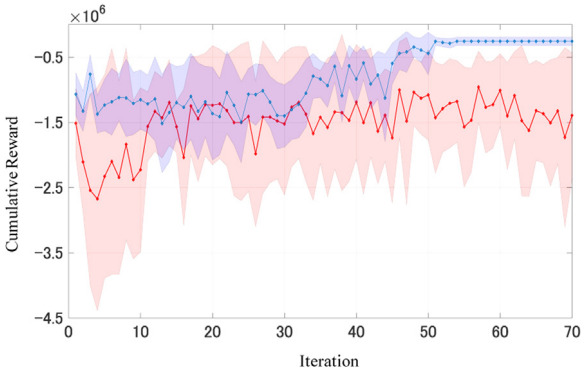


Fig. 4. Comparison between unfactorized FDPP and FFDPP. FFDPP performance is marked in blue; unfactorized FDPP is marked in red. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5. Experimental results

In this section, the setup for applying our proposed RL method for VAM process control is presented. The state, action, and reward function designs are explained in Section 5.1. Then the experimental setting is introduced in Section 5.2. The experiment results are shown in Section 5.3.

5.1. Implementation of RL on VAM process

Following the discussion in Sections 3.1, 3.2 and 4.2, the implementation of RL on the VAM process is constructed as a MDP problem with continuous state space and discrete action space defined by the entries in Tables 1(a) and 1(b). The simulator always starts from a fixed initial state $s_0 = [6.89, 23.6, 4.86, 49, 75, 76, 79, 106, 107, 109, 119, 131, 1.97 \times 10^5]$, where the order of the values corresponds to that of Table 1(a). Each factorized agent has access to the observations in Table 1(a) as well as the global value of the controller being represented. For convenience, below the quality sensor is abbreviated as *quality*, the production rate as *VAM*, temperature sensors 1–8 as *temperatures*, the gross profit as *profit*, based on the definitions of Table 1(a). Based on Eq. (24), the reward function is defined:

$$\begin{aligned}\text{Reward} = a * \text{Production rate} + b * (\text{Quality 1+Quality 2}) \\ -c * \text{Temperatures} - d,\end{aligned} \tag{25}$$

where d is a constant to ensure the reward received at each time step is always below zero. Following the assertion in the introduction that the assumption of the conditional independence tends to hold true in process control problems where distinct units are designed to be independent, the control units of different categories, e.g., steam, acetic acid, pressure, and temperature are factorized as $a^{(1)} =$ **flowcontroller1**, ..., $a^{(5)} =$ **temperaturecontroller**, according to Table 1(b). For leveraging the control units, an action set comprised of discrete values is defined so that at every time step, one agent chooses one value from the set to be added or subtracted from the control unit's global value. Addition/subtraction embodies the manipulation of the distillation column's steam flow, the decanter's reflux flow rate, and other process physical profiles.

5.2. Experimental settings

FFDPP is applied to the VAM process control problem in the Visual Modeler simulator, which was developed by the Omega Simulation

Table 2
Meta-parameters of FFDPP process control experiment.

Parameter	Description (number of)	Value
T	steps	1000
I	Iterations	70
M	Factorial agents	5
N	Discrete actions per agent	10
τ	Sampling basis functions	100

Corporation (Omega Simulation Corp.). The model-based controller derived by Machida et al. (2016) and Seki et al. (2010) is incorporated in the simulator to stably yield VAM. To prove the proposed algorithm's efficacy, we repeated and averaged ten independent experiments to collect statistical evidence. Each experiment consisted of 70 iterations, and each iteration has 1000 steps. Each time step in the simulation corresponds to around five minutes in real time. 1000 steps approximately represent two days of real-time plant operation. The proposed algorithm's performance is evaluated in terms of plant-wide stability, profit, and the computational resources needed during the two simulated days.

For each of the M control units, a corresponding action set is constructed by uniformly choosing N values from interval $[-0.01, 0.01]$. The amount of N is chosen empirically to achieve a balance between the high resolution of the discrete actions and the computational tractability. Here $N = 10$ for high resolution. According to Section 3.1, without the factorial policy setup, the size of the entire action set to be considered is $M^N = 5^{10} \gg 2^{20}$, which is intractable for efficiently exploring the common value function-based RL approaches.

Aside from maximizing the accumulated profit defined in Eq. (25), maintaining process stability is also a learning goal. Stability is defined by the violations of the safety ranges in Table 1(b). When a violation occurs, a corresponding penalty is added to the reward. The experiment was conducted on a PC with a processor i7-8700k, 3.70-GHz, and 32-GB memory.

The meta-parameters of the experiment are summarized in Table 2. Parameter τ controls the balance of the Fast-food approximation accuracy and the learning speed and the needs are specified by the user. Here $\tau = 100$. See Rahimi and Recht (2008) for technical details.

5.3. Experiment results

5.3.1. FFDPP performance

The evolving learning performance is shown on the left of Fig. 2, averaged over ten independent learning results for statistical evidence. The blue curve shows the mean reward, and the transparent blue area depicts the variance. The y-axis is the accumulated reward given in Eq. (25), summed over 1000 steps of every iteration. On the right, subplots illustrate the performance that evolves after 10, 40, and 70 iterations. After trial-and-error learning and without any prior knowledge about the model, the learned FFDPP policy successfully obtained a control policy to maximize the reward, solving the process control problem of a 13-dimension state space and a 5-dimension action space in 70 iterations within 70,000 samples.

To visualize the correspondence between the states and the learning progress, t-SNE (van der Maaten, 2014) compresses the 13-dimensional state vectors into two dimensions to draw the scatter plot in Fig. 3. By trial-and-error learning the agent explores the state space and finds a path that cycles around the optimal region indicated by deep red. Intermediate iterations might take the agent through some desirable states without controlling it within the optimal regions. Likewise, optimal states might be reached by some policies, but undesirable states will be experienced along the paths. An optimal policy is derived by circling regularly within the optimal region.

The performance of the learned policies is scrutinized in Fig. 5. To verify FFDPP's robustness and stability, we performed independent experiments. A rollout was conducted with a length of 2000 steps, 100% more than the length of the learning phase. We compared the performance of the learned policies and the model-based controller proposed in Machida et al. (2016) and Seki et al. (2010). The policies are shown here in blue solid lines, red dotted lines, and green dashed lines. The solid, straight red lines indicate the performance under the model-based controller that has minor oscillation, and the three curves show the adaptations made by the learned policies. In each subfigure, the black dashed lines indicate the mean value of the three curves averaged over 2000 steps. Figs. 5(a) and 5(b) show the readings of two product quality measures. Their readings are inversely proportional to the product quality. The product quality controlled by the learned policy is better on average than that of the model-based controller in the simulated period. Figs. 5(c) and 5(d) show the VAM production rate and profit per time unit. As the result adapts to the process dynamics, the curves change, hence the mean values must be compared with the performance of the model-based controller in terms of the VAM produced and the profit.

Without any model knowledge, in the simulated period that corresponds to around five actual days, the FFDPP agent successfully learned a policy that yields comparative performance to the state-of-the-art model-based controller.

5.3.2. Comparison to other RL methods

The computational mitigation brought by the factorial framework and the Fast-food kernel approximation is examined in Table 3 to serve as an ablation test, which compares the computational resources required by DPP-based algorithms. Note that in the original DPP paper (Azar et al., 2012), DPP was compared with classic model-free algorithm Q-learning and demonstrated superior performance. Hence in this comparison only DPP-based algorithms are considered. The KDPP and FKDPP configurations respectively follow the definitions in Cui et al. (2017a) and Cui et al. (2018), except that the sample threshold was set to $\eta = 0.9$, which is equivalent to discarding 90% of the samples and retaining the remaining 10% most informative ones. For the ablation test, unfactorized Fast-food DPP is also evaluated (see Section 5.3.3 for details). All the parameters of the compared algorithms were empirically tuned to yield the best performance.

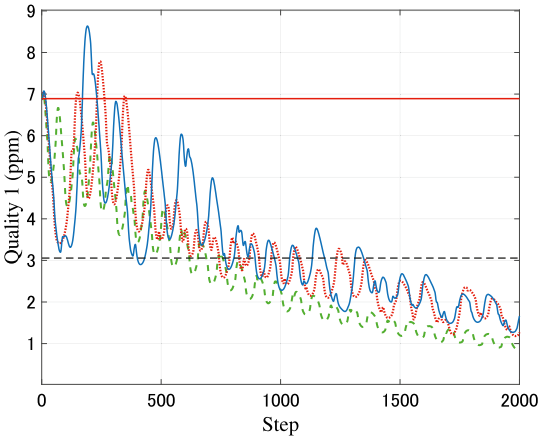
In our described experiment, *Success* is defined as after 70 iterations' learning, and the learned policy does not violate any constraint and performed comparable to the model-based controller (with cumulative reward approximately equals or exceeds 1.97×10^5 units). An indicator *Error* is also defined as a situation where the computing software reports error due to intractable computational time or running out of memory from a large amount of high-dimensional samples. The second column of Table 3 shows the maximum number of samples that each algorithm can handle before triggering the *error* indicator. The computation and policy evaluation times are evaluated with a maximum acceptable number of samples. For the factorial algorithms, since the time costs are recorded for one agent, the total time cost is that amount multiplied by the number of agents.

Note that a large acceptable number of FFDPP samples not only relies on Fast-food but also on the factorial policy for decomposing the one-time evaluation of a huge number of action preferences to several sequential evaluations, as we will verify in the next section.

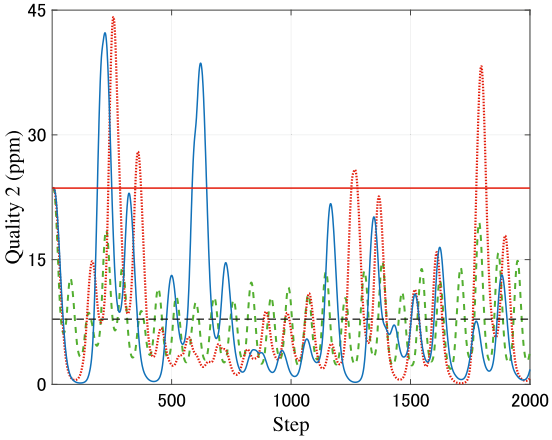
5.3.3. Unfactorized fast-food DPP

In this section the efficacy of factorial policy is empirically verified. Unfactorized Fast-food DPP (UFDPP) is run on the same task to obtain empirical evidence. As shown in Fig. 4, UFDPP's failure might be due to ineffective exploration, insufficient samples, or too many action preferences to learn. The learning curve was averaged over ten independent experiments.

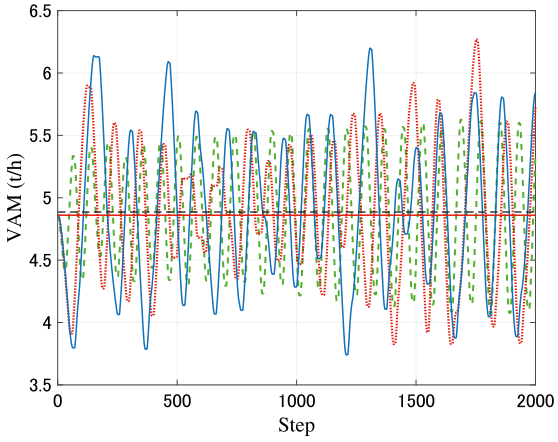
In summary, RL on a plant-wide process control requires many samples for building a robust controller. The KDPP and FKDPP bottleneck lies in the failure to process such a huge number of samples. On the other hand, the ability to efficiently search in high-dimensional



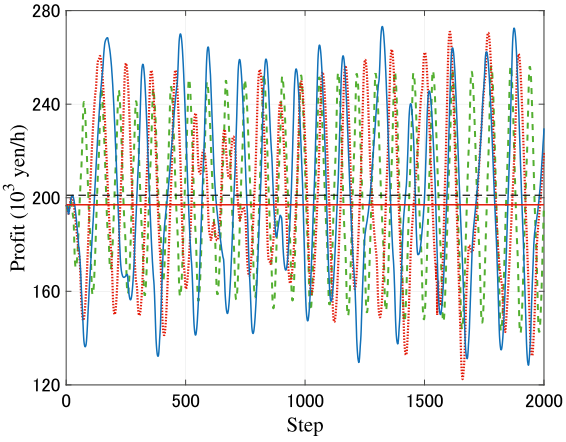
(a) Readings of quality sensor 1 under three learned policies. Reading is inversely proportional to product quality.



(b) Readings of quality sensor 2 under three learned policies. Reading is inversely proportional to product quality.



(c) VAM production rate under three learned policies



(d) Profit per time unit under three learned policies

Fig. 5. Comparison between learned three RL policies and model-based controller derived by Machida et al. (2016) and Seki et al. (2010). Red solid lines show performance of model-based controller. Blue solid lines, red dotted lines, and green dashed lines show performance of three learned policies, and black dashed lines show mean of respective policy lines. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
Performance comparison between DPP, KDPP, FKDPP, Fast-food DPP, and FFDPP on investigated task. Acceptable number of samples indicates maximum capacity of each algorithm before triggering *Error* indicator. Maximum computation and policy evaluation times are evaluated for maximum acceptable number of samples.

Algorithm	Acceptable number of samples	Maximum computation time (s)	Maximum policy evaluation time (s)	Success	Error
DPP	5000	552.5	0.0152	No	Yes
KDPP ($\eta = 0.9$)	20000	586.8	0.0015	No	Yes
FKDPP ($\eta = 0.9$)	30000	178.5 × 5	0.0014 × 5	No	Yes
FDPP (unfactorized)	> 70000	285.9	0.1405	No	No
FFDPP	> 70000	31.9 × 5	0.0044 × 5	Yes	No

actions space is indispensable, even if equipped with the capability of processing such a large number of samples. In this experiment, Fast-food approximation is exploited to tackle the first problem; factorial policy is leveraged to handle the latter.

6. Discussion

FFDPP was successfully applied to a commercial VAM simulator that is comprised of eight sub-systems for materials feeding, reacting, and recycling. We solved the process control problem by an FFDPP agent that observed 13-dimension states and manipulated five controllers with 5^{10} possible actions to improve the production and product quality while maintaining the process stability. As the first step toward real-world applications, our proposed algorithm successfully learned a model-free policy whose performance is comparable to the state-of-the-art model-based controller within 70 iterations, demonstrating the potential of FFDPP applications on the autonomous control of large-scale chemical plants.

From an algorithmic perspective, we proposed a novel RL algorithm suitable for large-scale system control problems. FFDPP's efficacy relies on the synergy of three components: Fast-food for transforming the problem of obtaining enough samples in high-dimensional space into sampling in the frequency domain; a factorial policy for decomposing

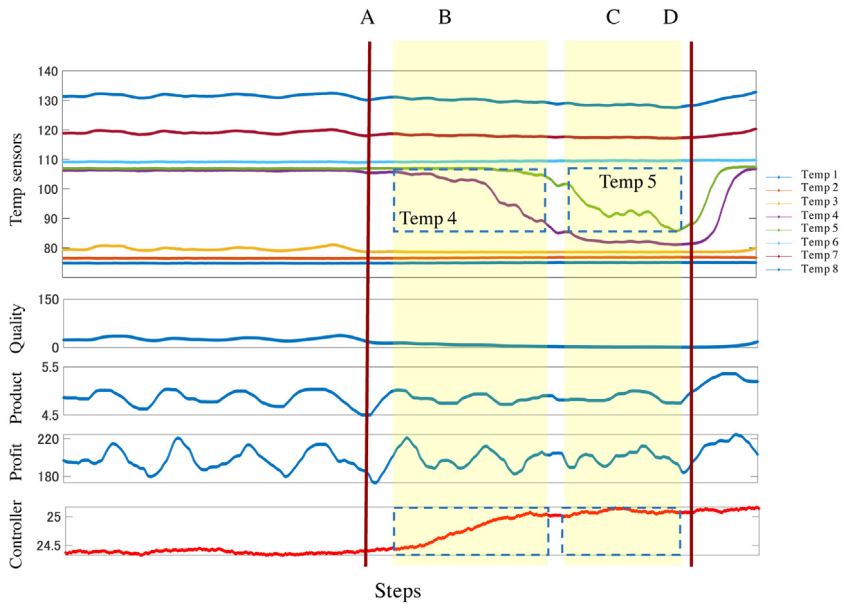


Fig. 6. Analysis of controlled behavior using learned FFDPP policy. Top subplot shows eight temperature sensors' reading during rollout. Middle subplots mark changes of main observations in blue. Bottom subplot depicts common adaptation made by FFDPP agent to maintain process stability and high profit. Label *Quality* refers to quality sensor 2 since it shares similar trend with sensor 1. *Controller* refers to flow controller 2. Agent's behavior greatly changes starting from time step *A* and goes through two stages of change in periods marked by yellow rectangles *B* and *C*. After time step *D* agent manages to restore normal temperature distribution of distillation column.

large action spaces into independent ones by assuming conditional independence between control units, which holds for process control problems since the units are designed to be independent; and DPP for ensuring smoothness and convergence in large-scale systems by exploiting the Kullback–Leibler relative entropy between the baseline and current policies.

One logical question is whether the performance can be reproduced or improved using DRL techniques. Sample complexity may be a major obstacle. To recover three units from malfunctioning, Kubosawa et al. (2018) used 4500 iterations, each of which consisted of a 30-minute simulation (135,000 min in total). For improving the product quality, production rate, and process stability in our complicated plant-wide control scenario that leveraged 18 units, we used 70 iterations, each of which consisted of around 2000 min (140,000 min in total), which is almost the same as their used time. Their simple task and the long induced time suggest extremely high sample complexity of DRL approaches, which is the reason they were excluded from consideration in plant-wide control.

The performance curves of the learned FFDPP policy in *subfigure C* of Fig. 2 changed as the result of the choice of reward function, because the reward function in Eq. (25) of the investigated control problem has no terms that restrict the variance of performance. However, the robustness and stability of the proposed algorithm can be seen from the multiple learned policies in Fig. 5.

In previous work (Cui et al., 2018), FKDPP was proposed to solve the VAM process control problem. However, such a huge number of high-dimensional samples severely limited the effectiveness of FKDPP. To alleviate the computation burden, assumptions about unlimited materials feeding and the decanter part of the simulation were made. On the other hand, in this paper, those unrealistic assumptions were successfully removed by activating all eight parts of the process and introducing only a constant amount of raw materials at each time step, which contributed to FFDPP's efficiency.

The performance of the learned FFDPP policy was also analyzed from the process control perspective. Specifically, the relationship between the actions made and the state changes of one rollout was examined in Fig. 6, and Temps 1–8 refer to the temperature sensors in Table 1(a). At the time step marked by the straight line *A*, the VAM production rate and the profit reach their minima. After this point, the behavior of the FFDPP policy changes greatly; all five PID controllers marked in red are actuated to increase the VAM production. Due to this attempt, in the period marked by the long, yellow rectangle *B* the temperature distribution of the distillation column changes, and *Temp 4* plummets because the agent increased the amount of VAM reflux to the top of the distillation column using *flow controller 2* (marked by the blue dashed rectangle in period *B*). As *Temp 4* enters a steady zone in period *C*, the amounts of reflux and steam are almost constant. Since *Temp 4* remains at a lower level than the normal behavior, *Temp 5* is also affected, quickly dropping to around the same level as *Temp 4*. To restore normal temperature distribution, *flow controller 1* is actuated to feed more steam into the distillation column.

The operation sequence of the learned FFDPP policy on the temperature distribution of the distillation column resembles an operator who has the ability to look inside the column and monitor the temperature distribution in real time.

7. Conclusion

The contribution of this research is twofold. From an algorithmic perspective, we proposed a novel RL method called FFDPP that is applicable to problems with high-dimensional state space and a large-scale discrete action set. FFDPP adopts the following:

1. a Fast-food method that efficiently approximates the state value function of high dimensionality, significantly reducing the computational complexity;
2. a factorial framework that divides a large set of discrete actions and share samples between agents, reducing the computational complexity without affecting the exploration range;
3. a smooth policy update based on the Kullback–Leibler divergence to unify the Fast-food and factorial framework to achieve both high sample efficiency and learning stability.

Our future work will focus on two directions. First, it is promising to apply FFDPP to a real-world VAM process to achieve model-free control. Real-world applications are more complicated because such factors as safety, reading errors, and control errors are possible. On the other hand, exploiting RL in a realistic environment might be beneficial for attaining insights into model-based design and applying RL to other

real-world problems. Our second direction is to automatically determine the number and the values of suitable discrete actions. Instead of empirically choosing a suitable N, it might be promising to estimate it from a sample trajectory of other relevant information.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix. Relationship with kernel DPP

In previous work (Cui et al., 2018), a factorial policy was applied to be combined with kernel DPP (KDPP). As a solution to the intractable computational cost of DPP with high-dimensional state space, Cui et al. (2017a) proposed KDPP to relieve the computation burden by selecting kernel subset $\mathcal{D}_{kernel}$:

$$K \approx K_{NN'} K_{N'N'}^{-1} K_{NN'}^{T}, \tag{A.1}$$

where $K_{N'N'} = \bar{\Phi}\bar{\Phi}^T$. Feature map $\bar{\Phi}$, which is computed based on selected kernel subset $\mathcal{D}_{kernel}$, is distinguished from $\hat{\Phi}$ in this paper, which is approximated using Fast-food. The computation relief of KDPP is ensured by the size of $\mathcal{D}_{kernel}$, which is smaller than the total number of samples: $N' \ll N$. KDPP, which can search for high-dimensional state space with limited samples, is applied to a 32-dimension, robot-arm manipulation (Cui et al., 2017a).

On the other hand, the advantages of KDPP are effaced by the almost linear growth of $\mathcal{D}_{kernel}$ during learning that results from high-dimensional state space, which renders the feature map intractable to compute and limits KDPP's utility in multi-controller problems. In the investigated task of this paper, both KDPP and FKDPP failed to address the large amount of samples in reasonable time, as seen from the comparison in Table 3 of Section 5.3.2.

References

Azar, M. G., Gómez, V., & Kappen, H. J. (2011). Dynamic policy programming with function approximation. In *International conference on artificial intelligence and statistics* (pp. 119–127).

Azar, M. G., Gómez, V., & Kappen, H. J. (2012). Dynamic policy programming. *Journal of Machine Learning Research (JMLR)*, *13*(1), 3207–3245.

Badgwell, T. A., Lee, J. H., & Liu, K.-H. (2018). Reinforcement learning – overview of recent progress and implications for process control. In *13th international symposium on process systems engineering. Computer and chemical engineering* (pp. 71–85).

Bellman, R. E. (2003). *Dynamic Programming*. New York, NY, USA: Dover Publications, Inc.

Chen, R., Dave, K., McAvoy, T. J., & Luyben, M. (2003). A nonlinear dynamic model of a vinyl acetate process. *Industrial and Engineering Chemistry Research*, *42*(20), 4478–4487.

Cui, Y., Matsubara, T., & Sugimoto, K. (2017a). Kernel dynamic policy programming: applicable reinforcement learning to robot systems with high dimensional states. *Neural Networks*, *94*, 13–23.

Cui, Y., Matsubara, T., & Sugimoto, K. (2017b). Pneumatic artificial muscle-driven robot control using local update reinforcement learning. *Advanced Robotics*, 1–16.

Cui, Y., Zhu, L., Fujisaki, M., Kanokogi, H., & Matsubara, T. (2018). Factorial kernel dynamic policy programming for vinyl acetate monomer plant model control. In *IEEE international conference on automation science and engineering* (pp. 304–309).

Dotoli, M., Fay, A., Miśkowicz, M., & Seatzu, C. (2015). A survey on advanced control approaches in factory automation. *IFAC-PapersOnLine*, *48*(3), 394–399.

Ernst, D., Glavic, M., Geurts, P., & Wehenkel, L. (2005). Approximate value iteration in the reinforcement learning context. Application to electrical power system control. *International Journal of Emerging Electric Power Systems*, *3*(1), 1–35.

Harp, S. A., Brignone, S., Wollenberg, B. F., & Samad, T. (2000). Sepia. a simulator for electric power industry agents. *IEEE Control Systems Magazine*, *20*(4), 53–69.

Hoskins, J. C., & Himmelblau, D. M. (1992). Process control via artificial neural networks and reinforcement learning. *Computers and Chemical Engineering*, *16*(4), 241–251.

Kano, M., & Nakagawa, Y. (2008). Data-based process monitoring, process control, and quality improvement: recent developments and applications in steel industry. *Computers and Chemical Engineering*, *32*(1), 12–24.

Kubosawa, S., Onishi, T., & Tsuruoka, Y. (2018). Synthesizing chemical plant operation procedures using knowledge, dynamic simulation and deep reinforcement learning. In *SICE annual conference* (pp. 1376–1379).

Le, Q. V., Sarlos, T., & Smola, A. (2013). Fastfood-computing hilbert space expansions in loglinear time. In *Proceedings of the 30th international conference on machine learning* (pp. 244–252).

Lee, J. H., & Wong, W. (2010). Approximate dynamic programming approach for process control. *Journal of Process Control*, *20*(9), 1038–1048.

Liu, W., Tan, Y., & Qiu, Q. (2010). Enhanced Q-learning algorithm for dynamic power management with performance constraint. In *The conference on design, automation and test in Europe* (pp. 602–605).

Luyben, W. L. (2011). Design and control of a modified vinyl acetate monomer process. *Industrial and Engineering Chemistry Research*, *50*(17), 10136–10147.

Luyben, M. L., & Tyréus, B. D. (1998). An industrial design/control study for the vinyl acetate monomer process. *Computers and Chemical Engineering*, *22*(7–8), 867–877.

van der Maaten, L. (2014). Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research (JMLR)*, *15*, 3221–3245.

Machida, Y., Ootakara, S., Seki, H., Hashimoto, Y., Kano, M., Miyake, Y., et al. (2016). *Vinyl acetate monomer (VAM) plant model: a new benchmark problem for control and operation study: Vol. 49* (pp. 533–538). International Federation of Automatic Control (IFAC).

Matsubara, T., Gómez, V., & Kappen, H. J. (2014). Latent Kullback-Leibler control for continuous-state systems using probabilistic graphical models. In *Conference on uncertainty in artificial intelligence* (pp. 583–592).

Metzger, M., & Polakow, G. (2011). A survey on applications of agent technology in industrial process control. *IEEE Transactions on Industrial Informatics*, *7*(4), 570–581.

Ng, Y. S., & Srinivasan, R. (2010). Multi-agent based collaborative fault detection and identification in chemical processes. *Engineering Applications of Artificial Intelligence*, *23*(6), 934–949.

Olsen, D. G., Svrcek, W. Y., & Young, B. R. (2005). Plantwide control study of a vinyl acetate monomer process design. *Chemical Engineering Communications*, *192*(10), 1243–1257.

Omega Simulation Corp. Visual Modeler. https://www.omegasim.co.jp/contents_e/product/vm/.

Qian, Y., Huang, Q., Lin, W., & Li, X. (2000). An object/agent based environment for the computer integrated process operation system. *Computers and Chemical Engineering*, *24*(2), 457–462.

Rahimi, A., & Recht, B. (2008). Random features for large-scale kernel machines. In *Advances in neural information processing systems* (pp. 1177–1184).

Seki, H., Ogawa, M., Itoh, T., Ootakara, S., Murata, H., Hashimoto, Y., et al. (2010). Plantwide control system design of the benchmark vinyl acetate monomer production plant. *Computers and Chemical Engineering*, *34*(8), 1282–1295.

Spielberg, S. P. K., Gopaluni, R. B., & Loewen, P. D. (2017). Deep reinforcement learning for process control. In *International symposium on advanced control of industrial processes* (pp. 201–206).

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems* (pp. 1038–1044).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT press Cambridge.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems* (pp. 1057–1063).

Syafiie, S., Tadeo, F., & Martinez, E. (2007). Model-free learning control of neutralization processes using reinforcement learning. *Engineering Applications of Artificial Intelligence*, *20*(6), 767–782.

Todorov, E. (2006). Linearly-solvable Markov decision problems. In *Advances in neural information processing systems* (pp. 1369–1376).

Tsurumine, Y., Cui, Y., Uchibe, E., & Matsubara, T. (2019). Deep reinforcement learning with smooth policy update: application to robotic cloth manipulation. *Robotics and Autonomous Systems*, *112*, 72–83.

Uchibe, E. (2018). Model-free deep inverse reinforcement learning by logistic regression. *Neural Processing Letters*, *47*(3), 891–905.

Wang, H., & Zhang, Y. (2008). Multi-agent based chemical plant process monitoring and management system. In *2008 4th international conference on wireless communications, networking and mobile computing* (pp. 1–4).

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3–4), 279–292.

Yang, A., Braunschweig, B., Fraga, E., Guessoum, Z., Marquardt, W., Nadjemi, O., et al. (2008). A multi-agent system to facilitate component-based process modeling and design. *Computers and Chemical Engineering*, *32*(10), 2290–2305.

Zhang, X., Kano, M., & Li, Y. (2017). Locally weighted kernel partial least squares regression based on sparse nonlinear features for virtual sensing of nonlinear time-varying processes. *Computers and Chemical Engineering*, *104*, 164–171.