Master's Thesis

Reinforcement Learning Algorithms for Large-scale Process Control: Application to Vinyl Acetate Monomer Processes

Lingwei Zhu

August 2, 2019

Graduate School of Information Science
Nara Institute of Science and Technology

A Master's Thesis
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
MASTER of ENGINEERING

Lingwei Zhu

Thesis Committee:

Professor Kenji Sugimoto	(Supervisor)
Professor Tsukasa Ogasawara	(Co-supervisor)
Associate Professor Takamitsu Matsubara	(Co-supervisor)

Reinforcement Learning Algorithms for Large-scale Process Control: Application to Vinyl Acetate Monomer Processes*

Lingwei Zhu

Abstract

Industrial process control is a very important topic as it forms the basis of modern era of technology. However, controlling large-scale industrial processes is complex in terms of methodology. To maintain the stable yield of a chemical plant experienced and expensive operators are required. Further, traditional model-based methods entail a complex model which we inevitably suffer from modeling error and computational complexity.

In this master research, a model-free reinforcement learning approach towards the automatic control of a chemical manufacturing process is explored: Vinyl Acetate Monomer (VAM) process. Reinforcement learning is a suitable answer to the abovementioned difficulties as model-free RL does not require model knowledge realize autonomous control by gradually learning from the environment.

In process control problems, sample space is often high dimensional and continuous as a result of many continuous-valued sensors. To tackle the high dimensionality, two novel reinforcement learning algorithms are proposed and are suitable for different setups: the first is efficient in local control problems while the other is suitable for plant-wide setup. The successful application of two algorithms demonstrates their effectiveness and efficiency both in learning and yielding comparable performance to the state-of-the-art model-based control technique.

*Master's Thesis, Graduate School of Information Science, Nara Institute of Science and Technology, August 2, 2019.

Keywords:

Reinforcement learning, Model-free control, Process control, High dimensionality

大規模酢酸ビニルモノマープロセス制御のためのスケーラブル強化学習アルゴリズム*

朱　令緯

内容梗概

工業プロセス制御は, 現代技術の基礎をなすものであるため, 非常に重要なトピックである. しかし, 大規模な工業プロセスを管理することは, 方法論とオーバーヘッドの両方の観点からみるとかなり複雑な問題である. オーバーヘッドのために, 化学プラントの安定した収量を維持する必要がある。そのため, 熟練したオペレータが必要であるが, それは高価になる. 方法論の観点では, 伝統的なモデルベースの方法は複雑なモデルを必要とし, それは必然的にモデル化誤差および計算の複雑さに悩まされている.

私の主な研究では, 化学製造プロセスの自動制御に向けたモデルフリーの強化学習アプローチである酢酸ビニルモノマー（VAM）プロセスを探ります. 強化学習は, モデルを回避し, 徐々に環境から学習することによって自律的制御を実現するため, 上述の困難に対して適切である.

VAM プロセスのセンサ読取値から生じる高次元サンプルの問題に取り組むために, 2 つの新しい強化学習アルゴリズムを提案し, それらを異なるプロセス制御問題に適用して, モデルベース制御技と比較し, 学習における有効性と効率性を検証する.

キーワード

強化学習、モデルフリー制御、プロセス制御、高次元制御

*奈良先端科学技術大学院大学情報科学研究科 修士論文, 2019 年 8 月 2 日.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Research Background

Industrial manufacturing processes form a building block of the modern era of technology. Take chemical manufacturing processes for example, very complex control procedures are executed to ensure the plants function stably without causing loss of profit or safety issues. However, the procedures that safeguard the stability and profitability come at a cost: a large number of experienced operators and engineers that can handle the complexity are required to monitor the real-time status of the plants. While heuristics of the experts are a reliable solution, the cost of training experienced hands are very prohibitive, especially in developed countries like Japan due to the aging population and shortage of experts [1]. To compensate for this shortage, many automatic control strategies have been devised under the emphasis of production innonvation and process stability [2, 3].

The trends in automatic chemical process control can be mainly categorized into conventional advanced control, adaptive control, modern-control-theory control, statistical process control, and soft sensors [4, 5]. Such model-based approaches as conventional advanced control and modern-control theory control require a mathematical model of the environment and derive control laws based on the model [6]. However, it is often difficult to obtain such models for complex and large-scale chemical plants in practice. Responding to this problem, model-free controllers naturally emerge. Model-free approaches, as adaptive control and statistical process control, serve as a complement to model-based methods, e.g., heuristic rules of fuzzy control [7], usage of artificial neural networks [8] to learn a model. For monitoring important variables of the process, soft-sensors are frequently used, e.g., partial least squares (PLS) in practical problems [9, 10]. Agent-based techniques are employed to help in control systems design [11], decision making, and fault detection [12, 13, 14].

Reinforcement Learning (RL) [15] can potentially be a powerful tool in chemical process control problems since it does not need a model of plant for control design, rather it can learn a control policy by trial-and-error interactions with the environment without supervision from experienced operators. Applications

in process control field range from electricity grid control to dynamic power management [16, 17, 18, 19]. In the chemical process control context, Hoskins et al. [16] apply RL to control a continuous-stirred-tank reactor (CSTR). Syafiie et al. [20] leverage RL in the classical pH neutralization control task on a laboratory plant. In other process control problems aside from chemical plants, Ernest et al. [18] use the enhanced Q-learning algorithm to derive a policy that achieves better power-performance tradeoff on both synthetic and small-sized real workloads. Harp et al. [21] embed RL algorithm in a simulator for electricity management to learn a profitable electricity pricing policy. While these methods achieve successes in low dimensional simulations or small-scaled real-world experiments, applying RL in chemical plants has not been substantially succeeded. The aforementioned studies mostly employ Q-learning [22], which does not scale well into high dimensional problems, and is typically expensive in terms of both learning time and computational resources for high dimensional systems. Deep RL (DRL), the combination of deep learning with reinforcement learning, has been explored in the process scenario [23, 24]. However, DRL requires a prohibitively huge number of samples for learning. This may be the reason that Kubosawa et al. [24] has conducted the experiment only with some isolated components of the Vinyl Acetate Monomer (VAM) manufacturing process from malfunctioning, without considering the full context of VAM process control problems, e.g., improving the production rate or product quality.

1.2 Research Objective

The objective of this research is to develop scalable reinforcement learning algorithms to large-scale process control problems. To the best of the author's knowledge, the research area of applying RL on large-scale chemical process control has largely remained blank. The possible reasons are (1) intractable computational cost incurred by the huge number of samples required for RL algorithms to converge; (2) derivation of policies that can perform comparable to the model-based control methods and (3) poor scalability of traditional RL algorithms, e.g., Q-learning. Conventional RL methods easily incur the curse of dimensionality [25].

Given the lack of research in applying RL on large-scale chemical process

control, I am motivated to conduct research on this problem. Specifically, the proposals are inspected in VAM process proposed by [26]. As a benchmark problem for testing chemical process design, VAM process has the features of (1) realistically large process flowsheet containing standard chemical unit operations; (2) typical characteristics of recycle streams and energy integration; (3) real non-ideal chemical components. The VAM model has been further improved by [27, 28, 29, 30]. Based on those improvements, [31] developed a simulation model with the state-of-the-art model-based controller under which the process runs in stable conditions. In this thesis, the proposed algorithms are examined in a VAM simulator to verify their effectiveness.

1.3 Overview and Structure

In this thesis two novel RL algorithms applicable the large-scale process control problems by integrating several efficient approximation schemes with a powerful learning paradigm are proposed. Specifically, the author proposes Factorial Kernel Dynamic Policy Programming (FKDPP) [32] and Factorial Fast-food Dynamic Policy Programming (FFDPP) that inherit the smooth policy update of Dynamic Policy Programming (DPP) by considering Kullback-Leibler divergence between current and baseline policies as a regularization term as demonstrated in [33, 34]. To further enhance the scalability of DPP, factor-wise policy factorization is explored as motivated by [35]. Two algorithms follow different philosophy of approximation: FKDPP is motivated by [36] and inspiration for FFDPP is drawn from [37]. Their difference will be detailed in Chapter 4.2 and 4.3, respectively.

This thesis begins with an introduction to the Reinforcement Learning (RL) and the algorithm that our methods based on. The basis of RL is introduced in Chapter 2.1. We formulate DPP in Chapter 2.2. Linear function approximation, as the most frequently used method to scale up RL, is introduced in Chapter 2.3 to improve the scalability of DPP and DPP-based derivatives.

Chapter 3.1 introduces an important type of chemical manufacturing process: Vinyl Acetate Monomer process that will be used for testing the proposals. Two different problems are presented: the first one detailed in Chapter 3.2 is a local control problem which is suitable to be solved using our first proposed method,

while the second one in Chapter 3.3 focuses on plant-wide control and is more ideally tackled using our second proposal.

The first proposed algorithm is introduced in Chapter 4.2 to handle the problem of homogeneous state space. In Chapter 4.2.1 we inspect the DPP approximation using the idea of reduced kernel set. The idea of factorial policy, which is critical to the success with large discrete action space, is introduced in Chapter 4.2.2.

In Chapter 4.3 we consider a plant-wide control problem which aims to realize overall control. We extend the application of DPP derivatives to handle heterogeneous state space in Chapter 4.3.1.

The experimental setups and results are presented alone as a Chapter for the convenience of comparison in Chapter 5. We first examine the experimental setup for FKDPP on process control problem 1 in Chapter 5.1. In Chapter 5.2 of testing FKDPP, FKDPP demonstrates superior performance than the model-based controller provided several assumptions hold. Likewise, the setup details for FFDPP on problem 2 are depicted in Chapter 5.3, followed by Chapter 5.4 showing FFDPP realizes comparable performance to the state-of-the-art model-based controller under the plant-wide setup with little assumptions.

2. Preliminary

2.1 Reinforcement Learning

Reinforcement Learning (RL) models problems as Markov Decision Processes (MDPs) which can be expressed as a quintuple [15]: $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, $\mathcal{T}$ denotes the transition probability from a state s to the next state $s^{'}$ with action a taken: $\mathcal{T}^{a}_{ss'}$. $\mathcal{R}$ comprises the reward incurred by that transition: $r^{a}_{ss'}$. The discount factor $\gamma \in (0, 1)$ exponentially discount the future rewards, giving more importance to the recent rewards.

In RL, we aim to maximize long-term accumulative reward which can be written in the form of *value function*:

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

where r_{s_t} is the reward at state s_t. π is a probability distribution associated with a state over action space: $\pi(a|s) : \mathcal{S} \rightarrow (0, 1)$.

RL methods search for an optimal policy π^{*} such that the value function (1) is maximized. According to the Bellman optimality criteria, this optimal policy satisfies the recursive Bellman equations:

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

The value function V is often used for planning. In the case of control, it is convenient to associate values with state-action pairs, which leads to the state-action value function $Q_{\pi}(s, a) \in \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$.

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

2.2 Dynamic Policy Programming

Dynamic Policy Programming (DPP) [38, 39] is a value function based RL algorithm that explicitly considers the smoothness in policy update. Traditional RL algorithms seldom consider the smooth update of their policies, which might lead

to brittleness and discountinuities in the value function estimates due to drastic changes of the behavioral policy.

DPP enforces the smoothness by considering the regularization-based technique. In supervised learning, regularization is frequently used to prevent the statitical model from overfitting. One well-knowen type of regularization is ridge regression. Given a design matrix X and its observation y, we want to determine the best linear regression coefficients w by solving the least-square problem: $w^* = \arg\min_w ||Xw - y||^2$. To prevent the weight w from growing abitrarily large, instead of solving the original least square problem, we solve

$$w^* = \arg\min_w ||Xw - y||^2 + \alpha||w||^2 \tag{4}$$

where the $||w||^2$ is the regularization term, α is the weight that controls how much we want to penalize the large components of the w. Solving this problem one finds the optimal weight that minimizes the lost function and has reasonable maginitude.

Following the same idea in Eq.(4), DPP incorporates the Kullback-Leibler (KL) divergence as a regularization term into the original RL formulation. Specifically, the KL divergence takes the following form:

$$KL\big(\,\pi(\cdot|s)\,\|\,\bar{\pi}(\cdot|s)\,\big) = \sum_{a\in\mathcal{A}} \pi(a|s)\log\left(\frac{\pi(a|s)}{\bar{\pi}(a|s)}\right) \tag{5}$$

where $\bar{\pi}$ is some baseline policy. Incorporating this term into the original value function formulation:

$$V^{\pi}_{\bar{\pi}}(s) := \lim_{n\to\infty} \mathbb{E}\Big[\sum_{k=0}^{n} \gamma^k \Big(r_{t+k} - \frac{1}{\eta} KL\big(\,\pi(\cdot|s)\,\|\,\bar{\pi}(\cdot|s)\,\big)\Big)\,\Big|\, s_t = s\Big] \tag{6}$$

According to the Bellman equation, the above value function also satisfies the following Bellman optimality:

$$V_{\pi^*}(s) = \max_{\pi} \sum_{\substack{a\in\mathcal{A}\\ s'\in\mathcal{S}}} \pi(a|s)\left[\mathcal{T}^a_{ss'}\big(r^a_{ss'} + \gamma V_{\pi^*}(s')\big) - \frac{1}{\eta}\log\left(\frac{\pi(a|s)}{\bar{\pi}(a|s)}\right)\right] \tag{7}$$

where η is the temperature term controlling the maginitude of regularization. The optimal policy π^* simultaneously satisfies the Bellman optimality and acheives

maximum distance to the baseline policy $\bar{\pi}$. As the ridge regression prevents the weight from growing infinitely, DPP prevents the policy from taking drastic steps in the update.

Following [40, 38], the regularized value function and policy also satisfy the Bellman recursive equations:

$$V_{\bar{\pi}}^{\pi^*}(s) = \frac{1}{\eta} \log \sum_{a \in \mathcal{A}} \bar{\pi}(a|s) \exp \left[\eta \sum_{s' \in \mathcal{S}} \mathcal{T}_{ss'}^{a} \big(r_{ss'}^{a} + \gamma V_{\bar{\pi}}^{\pi^*}(s') \big) \right], \tag{8}$$

$$\bar{\pi}^*(a|s) = \frac{\bar{\pi}(a|s) \exp \left[\eta \sum_{s' \in \mathcal{S}} \mathcal{T}_{ss'}^{a} \big(r_{ss'}^{a} + \gamma V_{\pi}^{\pi*}(s') \big) \right]}{\exp \big(\eta V_{\pi}^{\pi^*}(s) \big)}. \tag{9}$$

According to the property of Bellman operator [41], the optimal value function and policy need to be iterated infinitely to converge to their optimality. Hence we repeatedly replace the baseline policy in above equations to obtain optimal value function and policy. This can be done by using two loops: the inner loop updates value function, and the outer loop updates the policy. However, we can simplify the two-loop algorithm using only one loop by defining *action preference function* [42]:

$$P_{t+1}(s,a) = \frac{1}{\eta} \log \bar{\pi}^t(a|s) + \sum_{s' \in \mathcal{S}} \mathcal{T}_{ss'}^{a} \big(r_{ss'}^{a} + \gamma V_{\bar{\pi}}^{t}(s') \big) \tag{10}$$

where t indicates tth iteration during learning. By comparing Eqs.(10) with (8) and (9), we see by simple algebra we can write the value function and policy using only the action preference:

$$V_{\bar{\pi}}^{t}(s) = \frac{1}{\eta} \log \sum_{a \in \mathcal{A}} \exp \big(\eta P_t(s,a) \big), \tag{11}$$

$$\bar{\pi}^t(a|s) = \frac{\exp \big(\eta P_t(s,a) \big)}{\sum_{a' \in \mathcal{A}} \exp \big(\eta P_t(s,a') \big)}. \tag{12}$$

This suggests that, by using action preference function, we avoids the need for two-loop computation and can generate actions by directly computing action preference for each action given the state: $a^* = \arg\max_a P(s,a)$.

Substituting Eqs.(11), (12) back into Eq.(7), the one-loop update rule for action preference can be derived as:

$$\begin{aligned} P_{t+1}(s,a) &= \mathcal{O}P_t(s,a) \\ &= P_t(s,a) - \mathcal{M}_\eta P_t(s) + \sum_{s' \in \mathcal{S}} \mathcal{T}^a_{ss'}(r^a_{ss'} + \gamma \mathcal{M}_\eta P_t(s')) \end{aligned} \tag{13}$$

where $\mathcal{M}_\eta P_t = \sum_{a \in \mathcal{A}} \frac{\exp(\eta P_t(s,a))}{\sum_{a' \in \mathcal{A}} \exp(\eta P_t(s,a'))}$ is the Boltzmann softmax operator defined to replace the log-partition-sum in (11). $\mathcal{O}$ is the DPP operator, and $\mathcal{O}P \in |\mathcal{S}||\mathcal{A}| \times 1$ is the action preference vector for all state-action pairs. The optimality of the preference function is obtained when $t \to \infty$.

2.3 DPP with Function Approximation

The original DPP is only applicable to small-scale problems with model knowledge. To improve the scalability of DPP, function approximation is used to approximate the action preference [43]. Specifically, linear function approximation is adopted so that the action preference function can be approximated by $P = \Phi\theta$, Φ=$[\phi(x_1), \ldots, \phi(x_N)]^T$ is the feature map containing row vectors ϕ, and N denotes the number of samples. The feature vector comprises basis functions that take state-action pair (s,a) denoted as x as input: $\phi(x_n)$=$[\varphi_1(x_n), \ldots, \varphi_M(x_n)] \in \mathcal{R}^M$. One frequently used basis function is the Radial Basis Function (RBF) that takes a Gaussian shape:

$$\varphi(x_t) = \exp\left(-\frac{||x - x_t||^2}{\sigma^2}\right) \tag{14}$$

where σ is the user-specified width parameter. The RBF takes a bell shape in 3-D space as shown in Figure 1. It serves the function of projecting data from low-dimensional space to high dimensional space. In problems where the data in the original spaces are nonlinearly separable, RBFs are often used to project them to the infinite dimensional space. The RBF is infinite dimensional because of the well known fact that exponential function can be written as an infinite series of polynomials.

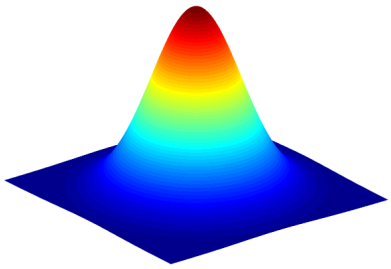


Figure 1. 3D plot of an RBF function

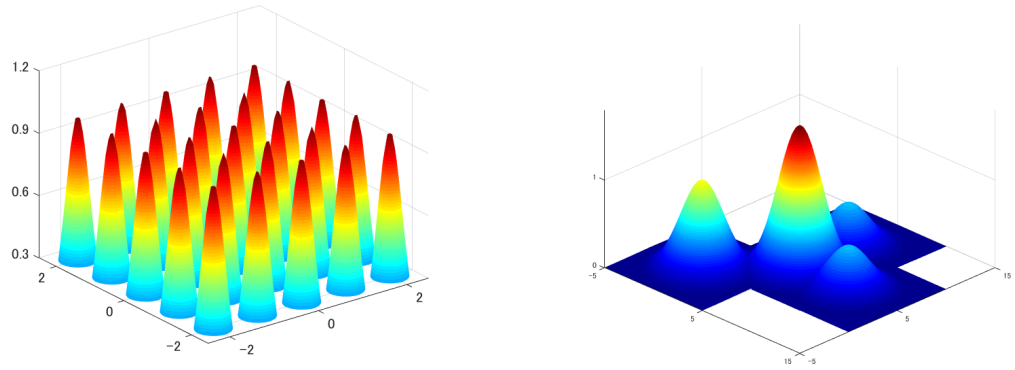
Given the basis function matrix Φ_t formed of data samples collected up to iteration t, the action preference is approximated by Φ and the weight vector: $P_t = \Phi\theta_t$. The weight vector θ_t is obtained by iteratively solving the least-square problem that minimizes the loss between the true action preference and the approximation $J \triangleq \|\mathcal{O}P - \Phi\theta\|^2$:

$$\theta_t^* = \arg\min_{\theta_t} J(\theta_t) = [\Phi_t^T\Phi_t + \sigma^2 I]^{-1}\Phi_t^T\mathcal{O}\hat{P}_t \tag{15}$$

where σ is the regularization term that avoids overfitting due to limited number of samples. For simplicity we drop the subscript t of the weight vector for approximating the action preference in later sections, i.e., $P_t = \Phi\theta$.

Conventional function approximation schemes evaluate the feature map Φ by setting up a grid uniformly over the state or action space, then evaluate basis functions on some of the grid points, usually with equal distance, as shown in Figure 2a. The characteristics of the target function are approximated by using a weight θ. At grid points where the target function has high values, large θ_i are associated with the grid points to raise the value of basis functions, where i denotes the ith component of the vector θ. Likewise, at grid points where the target function has low values, near-zero weight components are assigned to the basis functions, this effect can be seen in Figure 2b.

While the grid method works fine for small-dimensional problems, the number of basis functions required to uniformly cover the space grows exponentially with



(a) The RBFs set uniformly over the grid in 3D space.

(b) The fitted RBFs to some target function.

Figure 2. The grid basis function approximation method

the dimensionality. Suppose that we set 10 basis functions for each dimension, then a common problem of 3D space would require 10^3 basis functions. This number becomes astronomical in the problem of more than 10 dimensions, the resulting huge number of basis functions quickly renders computation intractable. This is referred to as *the curse of dimensionality*.

It is obvious more sophisticated function approximation methods are needed. Two approaches that improve the scalability of DPP are detailed in Chapter 4.2.1 and 4.3.1, respectively.

3. VAM Process and Control Problems

3.1 VAM Process Description

The VAM manufacturing process has been a popular benchmark problem for many chemical tests due to its realistic complexity and appropriate design. As a powerful tool for training operators, the Visual Modeler simulator can simulate the VAM manufacturing process. Following the process diagram shown in Figure 3, the functions of each part are briefly introduced. Further details can be found in [27, 31].

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 byproduct.

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

$C_2H_4 + 3\,O_2 \longrightarrow 2\,CO_2 + 2\,H_2O$

Here the = indicates a double chemical bond.

Part 3 comprises 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 comprises an absorber to extract gaseous VAM and discharge it to the buffer tank (Part 6).

Part 5 is the gas-purge system that maintains the pressure of the process.

Part 6 is the buffer tank for storing the VAM semi-product which then is fed to the distillation column.

Part 7 is the azeotropic distillation column extracting the VAM-water mixture and cycling acetic acid back to the parts 1 and 4.

Part 8 is the decanter that decants the final VAM product.

The VAM manufacturing process serves the objective of producing high quality VAM product while maintaining plant-wide stability. This is achieved by operators manually coordinating *control* units, mainly PID controllers from all parts of the process. Process stability is defined in terms of readings of several indicators that monitor anomalies. To verify effectiveness of the proposals, all

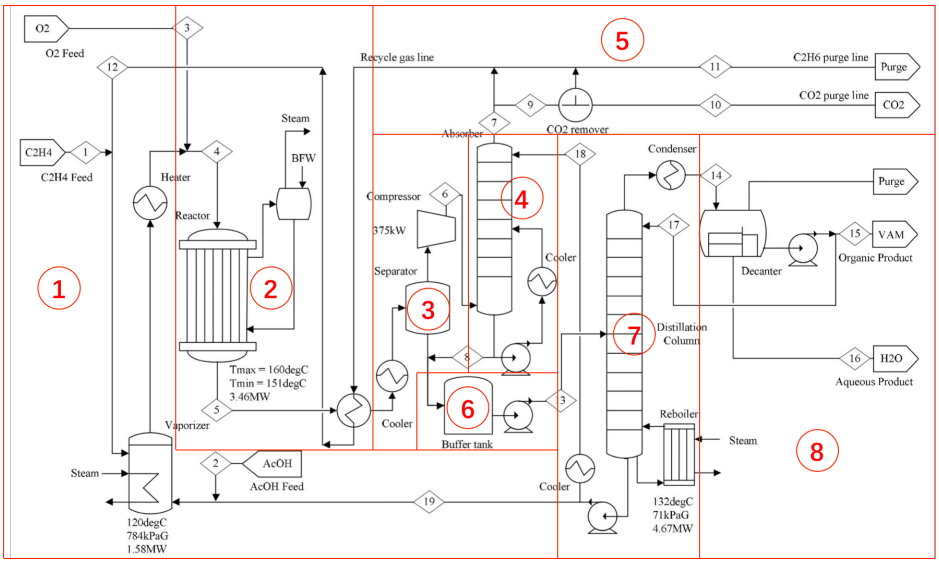


Figure 3. Flow diagram of the VAM process

proposals in this thesis are inspected in the VAM process simulator proposed by [26]. As a benchmark problem for testing chemical process design, VAM process has the features of (1) realistically large process flowsheet containing standard chemical unit operations; (2) typical characteristics of recycle streams and energy integration; (3) real non-ideal chemical components. The VAM model has been further improved by [27, 28, 29, 30]. Based on those improvements, [31] developed a simulation model with the state-of-the-art model-based controller under which the process runs in stable conditions.

Due to the complex nature the of process, it is desirable to first optimize parts with less complications, as a step towards real-world plant-wide control. In this thesis, two respective control problems in Chapter 3.2 and 3.3 are focused on using units from part 6 and part 7, while the second control problem consider global dynamics and the first does not.

3.2 VAM Process Control Problem 1

The VAM manufacturing process serves the objective of producing high quality VAM product while maintaining plant-wide stability. This is achieved by oper-

ators manually coordinating *control* units, mainly PID controllers from all parts of the process. Process stability is defined in terms of readings of several indicators that monitor anomalies. In this section, we consider VAM process control problem 1 that involves two parts of the plant.

Specifically, we focus on observing and controlling the decanter tank (Part 8) and the distillation column (Part 7), with the overall objective of optimizing VAM yield and its quality while maintaining stability as the first step towards the optimal control of the whole VAM plant in Fig. 3 by reinforcement learning. This problem features a nine-dimensional state space including observed values and control parameters detailed in Table 1 and 2, respectively. A finite discrete action set is defined as increasing/decreasing each control parameter. The initial parameters of the plant are provided to start from equilibrium. The specific objective is optimizing the VAM production's quality and quantity while keeping the level of the decanter tank and the flow rate from the distillation column within safe limits. This is done by manipulating the decanter's reflux-flow rate and feed-flow temperature, and the distillation column' s steam-flow rate and pressure.

One of the objectives 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 and the decanter. The optimization objective is defined by *profit*:

$$\text{Profit} = a * \text{yield} + b * \text{quality} - c * \text{cost} \tag{16}$$

where a, b, c are constants. A model-based controller is derived in [27, 31] under which the VAM process runs in stable conditions and outputs a constant amount of VAM with consistent quality. Without causing confusion, in this paper we denote the state-of-the-art model-based controller in [27, 31] as *the model-based controller*. Our target is to apply RL to learn a model-free policy that yield profit comparative to the model-based controller.

Name	Description	Control criteria
FI560.F	Flow rate of VAM	To be optimized as VAM yield, part of the *profit*
LI550.L	Level(%) of the decanter	$< 100\%$
s407.F	flow rate from distillation column to decanter tank	> 0
QI531.Q	Acetic acid density	< 100 *ppm*
QI560.Q	VAM density (inversely proportional to the quality)	< 150 *ppm*

Table 1. Observation units of the investigated problem 1.

PID controller	Description	Effect
PC501	Maintain top pressure of the column with N_2 and gas purge	Acetic acid density rises/declines N_2 density rises/declines
TC501	Maintain pressure of the 3rd stage of the distillation column with superheated steam	Column temperature rises/declines Affect VAM yield and quality Steam amount increase/decreases Steam generation is part of cost
TC540	Control decanter feed flow temperature with cooling water	Control Decanter feed flow Affect stability of the process
FC550	Maintain temperature profile and product quality by controlling the reflux flow rate	Affect VAM yield and quality Affect stability of the process Control reverse flow rate

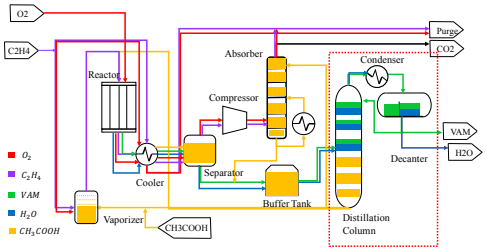
Table 2. Control units of the investigated problem 1.

3.3 VAM Process Control Problem 2

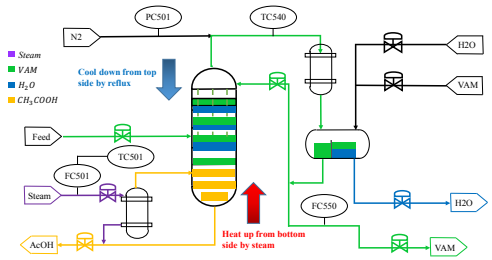
Unlike the VAM process control problem 1 in Chapter 3.2 that focused only on two parts, the VAM process control problem 2 has more control and observation units and all eight parts of the plant illustrated in Figure 3 are activated. This plant-wide activation setup brings further complications, since the plant becomes more sensitive as any perturbation from a single control unit might quickly propagate to all other parts and induce unpredictable changes.

The detailed definition of the observation and control units investigated in this paper are given in Table 1 and 2, while the complementary description of the units are depicted here. The product quality measurers *QI531.Q* and *QI560.Q* in Table 1 whose readings are inversely proportional to the product quality are constrained to stay in a tighter range compared to our previous research. Eight sensors of *T501.Temp* monitor the temperature inside the distillation column. The steam amount is part of the cost and is directly related to the physical profile of the column. PID controllers *FC501*, *PC501* and *TC501* take control of temperature, flow and pressure of the distillation column. *TC540*, *FC550* control the temperature of the feed and reflux flow of the decanter. By manipulating a scalar property of one PID controller, the behavior of that PID controller is affected. We refer the name of the investigated controllers to the scalar property without causing confusion.

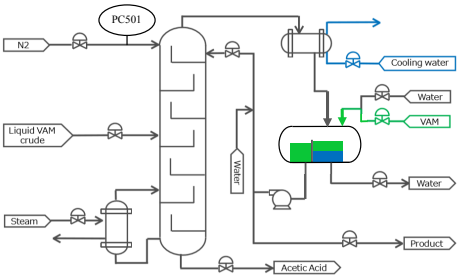
The effects of the control units located in part 7 and part 8 listed in Table 4 are illustrated in Figure 4. Figure 4a shows the normal working conditions of the process. Main materials and corresponding pipelines are marked with different colors. When malfunction occurs, some of the colors may turn black to indicate the pipelines are disabled. In this research we focus on the units located in the distillation column and the decanter, which are marked by the red dashed rectangle in Figure 4a. The summary of control units and their respective effects/positions are illustrated in Figure 4b. The temperature profile of the distillation column is adjusted by cooling using reflux from the top, or by heating with steam from the bottom. VAM and water are stored in the upper part of the column, whereas acetic acid is stored in the lower part. Control units are marked with ellipses. Figures (4c) to (4f) show the phase transitions resulted from the activation of each of the five control units.



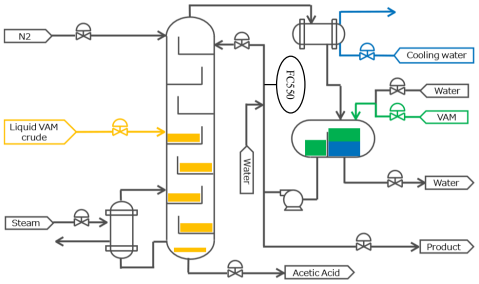
(a) The VAM process in normal working conditions under plant-wide activation. The control units are located in the distillation column and decanter parts, shown in the red dashed rectangle.



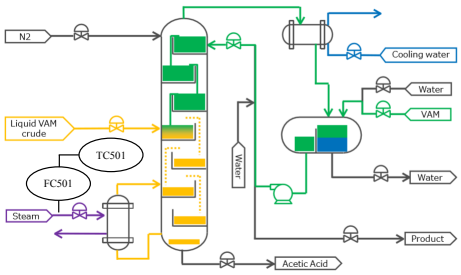
(b) The distillation column and decanter in normal working conditions under plant-wide activation. All five control units are marked with ellipses.



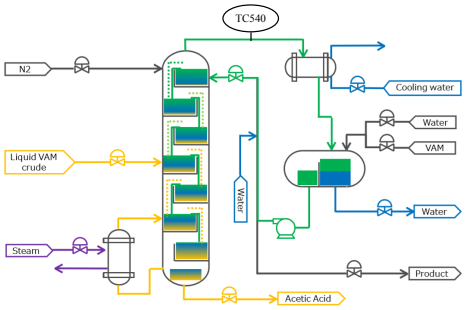
(c) Before the process is activated, VAM-water mixture is stored in the decanter. FC501 located in the top left controls N_2 feed rate to the top of the distillation column.



(d) The process is activated, VAM crude stream indicated by yellow is fed to the column via the pipeline connecting the distillation column and previous parts. FC550 located in the right pipeline to the top of the distillation column controls the flow rate VAM discharged from the decanter.



(e) FC501 indicated as purple in lower left controls the steam amount fed to the distillation column. At a slightly higher position, TC501 shown in yellow controls the temperature of the middle part of the distillation column.



(f) The normal working conditions of the distillation column and the decanter. Cooling water marked by blue is introduced to the column to maintain its temperature profile.

Figure 4. Phase transitions of VAM process normal working.

Name	Description	Control criteria
FI560.F	Flow rate of VAM	To be optimized as VAM yield, part of the *profit*
LI550.L	Level(%) of the decanter	$< 100\%$
QI531.Q	Acetic acid density	< 100 *ppm*
QI560.Q	VAM density (inversely proportional to the quality)	< 150 *ppm*
T501.Temp[1] T501.Temp[4] T501.Temp[7] T501.Temp[10] T501.Temp[11] T501.Temp[14] T501.Temp[17] T501.Temp[20]	Temperature inside the distillation column	To be optimized as cost, part of the *profit*

Table 3. Observation units of the investigated problem 2.

PID controller	Description	Effect
FC501	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
PC501	Maintain top pressure of the column with N_2 and gas purge	Acetic acid density rises/declines N_2 density rises/declines
TC501	Maintain pressure of the 3rd stage of the distillation column with superheated steam	Column temperature rises/declines Affect VAM yield and quality Steam amount increase/decreases Steam generation is part of cost
TC540	Control decanter feed flow temperature with cooling water	Control Decanter feed flow Affect stability of the process
FC550	Maintain temperature profile and product quality by controlling the reflux flow rate	Affect VAM yield and quality Affect stability of the process Control reverse flow rate

Table 4. Control units of the investigated problem 2.

4. RL Algorithms for Large-scale Plant Control

4.1 Overview

In this section we examine the two proposed algorithms: Factorial Kernel Dynamic Policy Programming (FKDPP) and Factorial Fast-food Dynamic Policy Programming (FFDPP). In Chapter 4.2 the theory of FKDPP is introduced. Specifically, the concept of KDPP [36] is first detailed. In Chapter 4.3 we introduce the other setup which replaces the 'Kernel' with 'Fast-food'.

FKDPP works with the assumption of homogeneous state spaces as opposed to FFDPP which works with heterogeneous state spaces. The proposal of two complementary algorithms enables one to deal with a wide range of process control problems in real-world large-scale systems.

4.2 Factorial Kernel Dynamic Policy Programming

In Chapter 2.3 the limitation of DPP with linear function approximation is examined. To be applicable to broader range of real-world problems, DPP awaits more sophisticated approximation schemes. In this sub-Chapter the theory of FKDPP is detailed. Specifically, to handle large state space induced by the continuous sensor readings, the idea of reduced kernel set is used. The large discrete action space formed from multiple controllers is tackled by exploiting the factorization of action space into smaller subspaces.

It is worth noting that as demonstrated in the 32-DOF robot arm manipulation task [36], KDPP is capable of dealing with very large amounut of samples when the state space is *homogeneous* (i.e., values from different dimensions do not differ much). This enables FKDPP to be applicable to local control problems where the sensor readings do not change drastically, as shown in [32]. The assumption of homogeneous state space is opposed to the assumption posed in Chapter 4.3.

One other advantage of FKDPP is the absence of need to tune the number of basis functions since FKDPP incrementally adds basis functions to the reduced kernel set. Theoretically, given enough time and computational resources, a problem will eventually be solved since there will be enough number basis functions

recovering the state space, as opposed to algorithms need manual tuning of this parameter.

4.2.1 Kernel Dynamic Policy Programming

Besides some early applications in inverse RL [44] and robot control [33], DPP is still unable to be fully utilized in many real-world applications. This is because the curse of dimensionality mentioned in previous chapter.

One better solution than setting a grid uniformly over the space is to apply kernel trick. Starting from the action preference approximation in Chapter 2.3:

$$P_{t+1} = \phi(x)^T\theta = \phi(x)^T[\Phi^T\Phi + \sigma^2 I]^{-1}\Phi^T\mathcal{O}P_t, \tag{17}$$

by using the Woodbury identity, the equation is transformed to:

$$P_{t+1} = \phi(x)^T\alpha = \phi(x)^T\Phi^T[\Phi\Phi^T + \sigma^2 I]^{-1}\mathcal{O}P_t. \tag{18}$$

where α is the $N\times 1$ dual variable vector of the $M\times 1$ weight vector θ.

Defining a $N\times N$ Gram matrix $K := \Phi\Phi^T$ such that $[K]_{ij} = <\phi(x_i), \phi(x_j)> = k(x_i, x_j)$ and a $N\times 1$ vector $k(x) = [k(x, x_1), \ldots, k(x, x_N)]^T$, the kernel trick is implicitly used. It is a simple way to generate features for algorithms that only depend on the inner product between pairs of input points. Applying this technique greatly simplifies the computation of high dimensional feature maps. Hence, the approximated action preferences function represented by the kernel is obtained as:

$$P_{t+1} = k(x)^T[K + \sigma^2 I]^{-1}\mathcal{O}P_t. \tag{19}$$

Value function with kernel methods are explored in [45, 46] with relatively low dimensionality in simulations. To further extend the scalability, [36] proposed to approximate the kernel matrix and kernel vector using a reduced kernel set $\mathcal{D}_{kernel}$ that contains N' most informative samples instead of all N samples:

$$K \approx K_{NN'}K_{N'N'}K_{NN'}^T \tag{20}$$

$$\phi(x) \approx [\varphi(x_1), \ldots, \varphi(x_{N'})]\boldsymbol{\alpha} \tag{21}$$

where $K^{-1}_{N'N'}$ is a $N' \times N'$ matrix that is computed using $\mathcal{D}_{kernel}$. The vector $\boldsymbol{\alpha} = K^{-1}_{N'N'} k_{N'}(x)$ is the weight that minimizes the approximation error of $||\phi(x) - [\varphi(x_1), \ldots, \varphi(x_{N'})]\boldsymbol{\alpha}||^2$, $k_{N'}(x)$ represents vector $[k(x, \bar{x_1}), \ldots, k(x, \bar{x_M})]^T$. Note the difference between $k_{N'}(x)$ and $k(x)$ is that $k(x) \in \mathbb{R}^N$ is the original *true* kernel vector, while $k_{N'}(x) \in \mathbb{R}^M$ is the *approximated* kernel vector.

For every sample encountered during learning, the information criterion is computed according to:

$$\delta^*(x) = k(x, x) - k_{N'}(x)^T K^{-1}_{N'N'} k_{N'}(x). \tag{22}$$

Then the computed δ^* is compared with the user-specified δ to be determined whether or not be added to the reduced kernel set $\mathcal{D}_{kernel}$.

Reducing the evaluation of kernel matrix from all samples to a subset of most informative samples is a dramatic improvement, and the size of $\mathcal{D}_{kernel}$ is kept relatively small in problems with homogeneous state space (features have values of alike maginitudes), as demonstrated in the 32-D robot arm manipulation in [36].

4.2.2 Factorial Policy KDPP

KDPP has shown its efficient learning in the control of a pneumatic muscle-driven robotic hand with a 32-dimensional state space [36], while other conventional methods such as LSPI [47] could not. On the other hand, calculating $\mathcal{M}_\eta P_t(s)$ in Eq. (10) through a huge discrete action set $\mathcal{A}$ is intractable. For example, define M discrete actions for one control parameter. To control N parameters, the entire set of all possible actions becomes $|\mathcal{A}| = M^N$, which is intractable for large M and N. Typical methods employ a carefully hand-coded action set that allows only one action move at a time as in the experiment in [36]. However, this significantly limits the controllability of the system. Further, it necessitates prior knowledge on designing the action set, when there is no prior knowledge about the action, hand-coded action set might deteriorate the performance of RL algorithms.

To address this issue, the factorial policy version of KDPP is proposed to learn

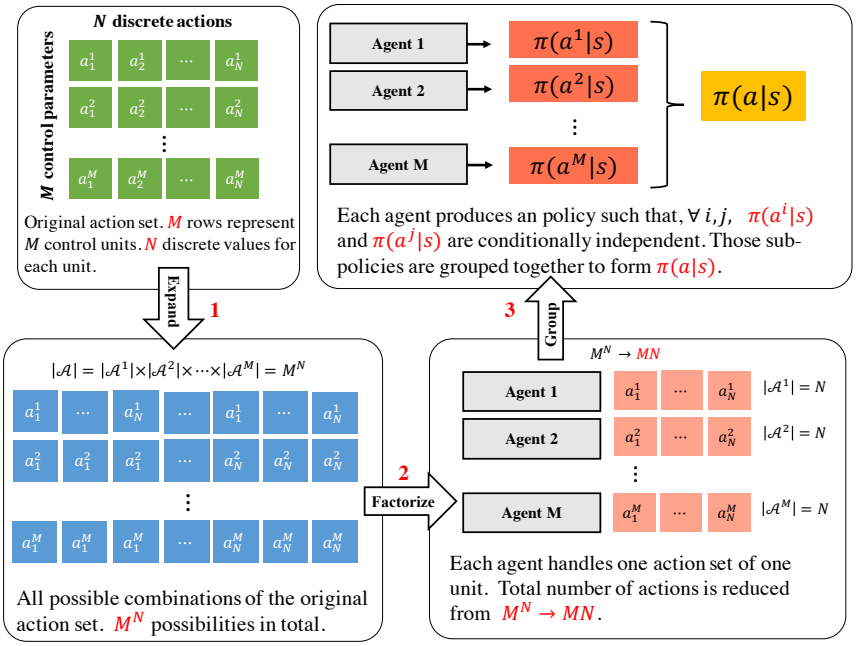


Figure 5. Illustration of factorial policy

action space dimension by dimension separately under the following assumption:

$$\begin{aligned} &\hat{\pi}(a|s) = \prod_{n=1}^{N} \pi(a^{(n)}|s), \\ \boldsymbol{s.t.} \quad &\pi(a^{(i)}|s) \perp \pi(a^{(j)}|s), \quad \forall i, j \\ &\pi(a|s) \approx \hat{\pi}(a|s) \end{aligned} \tag{23}$$

i.e., we assume that the policy can be factored into components that comprise only a small subset of actions, s.t. the conditional independence conditioned on states between sub-policies holds. The symbol $\perp$ denotes independence between two random variables. The illustration of factoring M^N candidates is in Figure 5. By factorizing the action set into several independent sub-action sets, we define a multi-agent learning framework under which the each agent learns different value function $V_\pi^{(n)}$ and policy $\pi_t^{(n)}$, where the superscript n denotes the agent handling the nth subset. The factorized value functions and policies are the factored counterparts to Eqs. (8) and (9). Hence the factorized action preferences can be expressed as:

$$P_{t+1}^{(n)}(s,a) = \frac{1}{\eta}\log \bar{\pi}^t(a^{(n)}|s) + \sum_{s^{'}\in\mathcal{S}} \hat{\mathcal{T}}_{ss^{'}}^{a}\big(r_{ss^{'}}^{a} + \gamma V_{\bar{\pi}}^{t^{(n)}}(s^{'})\big) \tag{24}$$

where $\hat{\mathcal{T}}_{ss^{'}}^{a}$ is a transition model different from the standard definition (10). The difference is resulted from the factorial policy, since instead of taking an action a and transitioning to the next state $s^{'}$, we now generate n sub-actions independently and group them together for the transition.

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

$$\bar{\pi}_t^{(n)}(a|s) = \frac{\exp\big(\eta P_t^{(n)}(s,a)\big)}{\sum_{a^{'}\in\mathcal{A}}\exp\big(\eta P_t^{(n)}(s,a^{'})\big)}. \tag{25}$$

Recall in Chapter 2.3 we approximate the action preference using linear function approximation $P_t = \Phi\theta$. By using factorial policy the feature map Φ of size $|\mathcal{M}|\times|\mathcal{S}||\mathcal{A}|$ can be factorized into N smaller ones each with size $|\mathcal{M}|\times|\mathcal{S}||\mathcal{A}^{(n)}|$. Specifically,

$$\Phi \in \mathcal{R}^{M\times\mathcal{S}\mathcal{A}} \xRightarrow{\text{Factorize}} \Phi^{(n)} \in \mathcal{R}^{M\times\mathcal{S}\mathcal{A}^{(n)}},\ n\in[1,2,\ldots,N] \tag{26}$$

where $\Phi^{(n)}$ represents the feature map of the nth agent. Thus with function approximation the action preference for the nth action set is updated by:

$$P_{t+1}^{(n)}(s,a) = P_t^{(n)}(s,a) + r_{ss^{'}}^{a} - \mathcal{M}_\eta P_t^{(n)}(s) + \gamma\mathcal{M}_\eta P_t^{(n)}(s^{'}). \tag{27}$$

Consequently, we solve N regression problems similar to Eq. (15) to obtain the N weight vectors at iteration t :

$$\theta_t^{(n)^*} = \underset{\theta_t^{(n)}}{\arg\min}\, J(\theta_t^{(n)}) = [{\Phi_t^{(n)}}^T\Phi_t^{(n)} + \sigma^2 I]^{-1}{\Phi_t^{(n)}}^T\mathcal{O}\hat{P}_t^{(n)}. \tag{28}$$

In the context of KDPP, the dual weight vector $\boldsymbol{\alpha}$ is updated rather than θ. Following the same logic as above, reduced kernel sets $\mathcal{D}_{kernel}^{(n)}$ and dual weight vector $\boldsymbol{\alpha}^{(n)}$ is maintained for each of the agents. At iteration t, the dual vectors

Algorithm 1: Factorial Kernel Dynamic Policy Programming

Input: $N, \gamma^{(n)}, \eta^{(n)}, T, I, \delta^{(n)}, n = 1, \ldots, N$

Output: weight vectors $\theta^{(n)}, n = 1, \ldots, N$

Divide $\mathcal{A}$ into $\{\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(\mathcal{N})}\}$

Initialize policy $\pi_{explore} = [\pi^{(1)}_{random}, \ldots, \pi^{(N)}_{random}]^T$

Initialize reduced kernel sets $\mathcal{D}^{(n)}_{kernel} = \emptyset$

for $i = 1, \ldots, I$ **do**

  for $t = 1, \ldots, T$ **do**

    for $n = 1, \ldots, N$ **do**

      $x^{(n)} = [s^i_t, a^{(n)}]^T$

      compute $\delta^{*(n)}$ *using Eq.(22) and compare with* $\delta^{(n)}$

      collect $x^{(n)}$ *in* $\mathcal{D}^{(n)}_i$ *if* $\delta^{*(n)} > \delta^{(n)}$

      compute $\hat{P}^{(n)}_t$ *following Eq. (25)*

      sample action $a^{(n)}_t$ *from Eq.(27)*

    end

  end

  for $n = 1, \ldots, N$ **do**

    compute $K^{(n)}_t$ *using Eq.(20)*

    Update $\boldsymbol{\alpha}^{(n)}_{t+1}$ *using Eq.(29)*

  end

end

are updated according to the new reduced kernel sets $(\mathcal{D}^{(n)}_{kernel})_t$ and approximated Gram matrix $K^{(n)}_t$:

$$\boldsymbol{\alpha}^{(n)}_{t+1} = [K^{(n)}_t + \sigma^2 I]\mathcal{O}P^{(n)}_t, \tag{29}$$

where the DPP operator on the action preference $\mathcal{O}P^{(n)}_t$ is defined in Eq.(27). The algorithm of FKDPP is illustrated in Algorithm 1.

The idea of factorial policy somewhat resembles the actor-critic method [48, 49, 50] in the sense that the actor, represented by factorial policies, perform in individual ways. But they are critized by the same performance criteria, i.e., the reward representing the product quantity and quality.

It is worth noting that factorial policy is a rather general solution to large discrete action space that does not rely specific choice of basis RL algorithm: it can be DPP or Q-learning or some other genre of algorithms. However, due to

the nature of tempered update of DPP, factorial policy coporated with DPP are easier to obtain convergence as empirically verified.

4.3 Factorial Fast-food Dynamic Policy Programming

At the heart of KDPP is the concept to retain only the most informative samples and discard the rest. This idea has been previously explored as discarding individual entries in the Gram matrix [51] or entire rows [52, 53]. While these approximations produce good low-rank or sparse approximations to the true kernel matrix, they seem to be bounded to low-dimensional problems with limited datapoints. Further, when the assumption of homogeneous state space does not hold, the efficiency of KDPP and thereupon FKDPP may no longer exist. Motivated by the idea of developing an algorithm that works with *heterogeneous* state space, we develop the algorithm of FFDPP, which leads to the change of approximation scheme in the previous chapter from FKDPP to FFDPP. We demonstrate that FFDPP is a more powerful learning paradigm when the state space is highly heterogeneous.

The proposal of FFDPP acts as a complementary part for FKDPP which is applicable to heterogeneous state space, as opposed to FKDPP efficient in homogeneous state space. Equipped with the two algorithms, one can tackle a variety of process control problems.

4.3.1 Fast-food Approximation

There are many approaches focus on extending the scalability of kernel method, in the sense of computing the Gram matrix K. One of the most important research recently is the *Random Kitchen Sink* (RKS) [54]. RKS projects the input data into random low-dimensional feature space and apply existing linear methods.

As briefly introduced in Chapter 4.2.1, the kernel trick is a simple way for the algorithms that rely only on the inner products of input datapoints, e.g., Support Vector Machine (SVM). This technique relies on the theorem due to Mercer more than a hundred years ago. For the clarity of the following explanation, the theorem is stated here.

Mercer's Theorem *Any kernel k:* $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *satisfying*

$\int k(x,x^{'})f(x)f(x^{'})dxdx^{'} \geq 0$ *for all measurable functions* f *can be expanded into the following summation:*

$$k(x,x^{'}) = \sum_{j} \lambda_j \phi_j(x)\phi_j(x^{'}) \tag{30}$$

where $\lambda_j > 0$ and ϕ_j are orthonormal.

Mercer's theorem states that any positive definite kernel can be computed as a summation of weighted inner products of feature vectors. This in turn implies that lifted datapoints can be computed as $K(x,y) = k(x,y) = <\phi_j(x), \phi_j(y)>$, where $x, y \in \mathbb{R}^d$ are the input datapoints, $\phi_j : \mathbb{R}^d \rightarrow \mathbb{R}^q$ defines the lifting, where $q >> d$.

Rahimi proposed that, instead of relying on the implicit lifting defined by the kernel trick, it might be better to explicitly map the data to a low-dimensional Euclidean distance space using a randomized feature map $z : \mathbb{R}^d \rightarrow \mathbb{R}^D$, such that the inner product between a pair of lifted datapoints approximates their kernel counterpart:

$$k(x,y) = <\phi(x), \phi(y)> \approx z(x)^T z(y) \tag{31}$$

where unlike the high dimensional mapping ϕ, z is low-dimensional. The low-dimensionality of z provides us with the benefit that can be used directly with the transformed input to approximate the desired nonlinear kernel machine.

Specifically, to obtain the desired kernel, an important theorem from harmonic analysis called *Bochner's theorem* is used, which further complements the Mercer theorem in the sense that a kernel is valid only if its Fourier transform is a valid probability measure, i.e., a proper probability distribution.

Bochner's Theorem *A continuous kernel* $k(x,y) = k(x-y)$ *is positive definite if and only if* $k(\delta)$ *is the Fourier transform of a non-negative measure, i.e.,*

$$k(x-y) = \int p(w) e^{jw^T(x-y)} dw = \mathbb{E}_w \left[\varphi_w(x) \varphi_w(y) \right] \tag{32}$$

where $\varphi_w(x) = e^{jw^T x}$. This states that, if w is drawn from the distribution $p(w)$, then we have an unbiased estimate of original kernel $k(x,y) = k(x-y)$. Note that by writing so we mean that the kernel should be shift-invariant, as is the case of common kernel functions including RBF, Cauchy and Laplacian, etc.

Two more steps to obtain the desired kernel function as used in Chapter 2.3, 4.2.1. The first is that, RBF and the corresponding Gaussian distribution are real, hence the complex components in Eq.(32) can be replaced by cosine function $z_w(x) = \sqrt{2}\cos(w^T x + b)$ and $p(w)$ set to be Gaussian. Second, to manipulate the *width* σ^2 of the desired feature map, one simply samples from the Gaussian with zero mean and variance σ^2. The approach of RKS is summarized below:

input Width σ^2, n, d
  Sample entries of $Z \in \mathbb{R}^{n \times d}$ i.i.d. from $\mathcal{N}(0, \sigma^{-2})$
for all x,
  Compute the feature map using Eq.(32)
  $\phi_j = \frac{1}{\sqrt{n}} \exp(i[Zx]_j)$
end for

The RKS approximation is a probabilistic approximation that converge with high probability and at the rate of independent empirical averages [54]. However, one shortcoming is that one needs to compute Zx for every observation, leading to $O(nd)$ operations and storage for each input x. In the application considered in this thesis, input comes as a stream of observations resulted from interacting with the environment. This stream of observations can be arbitrarily many if one wants to collect as many samples as possible. Hence it is desirable to further reduce the cost of computation and storage.

Le et al. proposed Fast-food approximation [37] to improve the computation time of RKS from $O(nd)$ to $O(n \log d)$ and storage from $O(nd)$ to $O(n)$. The improvement is crucial for applications with very big number of features, e.g., image processing.

Specifically, fast-food achieves the improvement by considering a matrix formed from multiplications of several diagonal simple matrices instead of Z. The fast-food matrix is denoted as V in the following equation:

$$V = \frac{1}{\sigma\sqrt{d}} SHG\Pi HB \tag{33}$$

where S, G, B are diagonal matrices. More specifically, $G_{ii} \sim \mathcal{N}(0,1), B_{ii} \in \{\pm 1\}$, the scaling matrix S can be adjusted to produce different kinds of kernels desired.

For the RBF case, it follows Chi-distribution, namely $S_{ii} \sim r^{d-1}e^{-\frac{r^2}{2}}$. $\Pi \in \{0,1\}$ is a permutation matrix, H is the Walsh-Hadamard matrix of the form:

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{and} \quad H_{2d} = \begin{bmatrix} H_d & H_d \\ H_d & -H_d \end{bmatrix}$$

Matrices S, G, B are once computed and stored. The Walsh-Hadamard matrix H can be obtained via the fast Walsh-Hadamard transform.

The detail of generating fast-food matrix is provided below, in which the important properties are explained.

Every row of V is a Gaussian variable

This statement comprises two facts :

(1) The rows of matrix V excluding S are i.i.d. Gaussian variables.

(2) Based on (1), multiplying with the scaling matrix S, rows of the fast-food matrix S are Gaussian, but no longer independent.

This first fact is because every entry of V excluding S takes the form of $[HG\Pi HB]_{ij} = B_{jj}H_i^T G\Pi H_j$, which is a sum of zero-mean independent Gaussian variables. According to the fact that adding d independent variables each of $\mathcal{N}(0,1)$ yields a Gaussian variable of $\mathcal{N}(0,d)$, the variance of each entry is d. The binary matrix B ensures that different entries in a row $[HG\Pi HB]_i$ have zero correlation, hence guaranteeing that they are i.i.d Gaussian variables.

The second fact is because of that, scaling rows of $HG\Pi HB$ into a unit-ball using entries of $S_{ii} = s_i||G||_{Frob}^{-\frac{1}{2}}$, $s_i \sim (2\pi)^{-\frac{d}{2}}A_{d-1}^{-1}r^{d-1}e^{-\frac{r^2}{2}}$ serves the purpose of making Gaussian random vectors distributed uniformly on the unit-ball. It provides information on sampling, since knowing the radius of the ball one knows that which regions are more likely to be concentrated with distributed points, hence the independence no longer holds. □

Rows of $HG\Pi HB$ have the same length

This is because the length can be written as $l^2 = [HG\Pi HB(HG\Pi HB)^T]_{ii} = [HG^2H]_j d = \sum_i H_{ij}^2 G_{ii}^2 d = ||G||_{Frob}^2 d$. □

The fast-food method preserves the advantages of RKS, while improves the scalability in the sense that instead of storing Z computing Zx with complexity $O(nd)$ for every observation encountered, now the complexity of computing Vx drops to $O(n \log d)$ and storing V drops to $O(n)$. This is because S, G, B are diagonal matrices that cost only $3n$ storage. The Hadamard matrix H is implicitly

represented using the recursive formula. These two facts show that storage of V is $O(n)$. To demonstrate that the computation costs $O(n \log d)$, one notes that the computation relies on the Hadamard transform which costs $O(n \log d)$ and evaluation of n basis functions which costs $O(n)$, hence the cost in total is $O(n \log d)$.

We produce fast-food feature maps using the follow procedure:

input Width σ, n, d
  Compute fast-food matrix V using Eq.(33)
for all x,
  Compute the feature map by $\phi_j = \frac{1}{\sqrt{n}} \exp\left(i[Vx]_j\right)$
end for

4.3.2 Factorial Fast-food Dynamic Policy Programming

Given the fast-food approximation in Chapter 2.3, 4.2.2 and 4.3.1, the second proposed algorithm: Factorial Fast-food Dynamic Policy Programming (FFDPP) is constructed in this section. The feature map using fast-food approximation is denoted as $\hat{\Phi}$ to be distinguished from its exact counterpart. According to the proof in [54, 37], $\hat{\Phi}$ recovers Φ with exponentially decreasing error with the increase of number of sampling basis functions τ. Typically, in most of problems the accuracy obtained by setting $\tau = 100$ suffices, choosing lower values of τ is mainly for tradeoff with computational resources.

Following the notation in previous chapters about iteration and factorial policy, the nth sub-action is sampled from the factorial fast-food policy:

$$\bar{\pi}_t^{(n)}(a|s) = \frac{\exp\left(\eta P_t^{(n)}(s,a)\right)}{\sum_{a' \in \mathcal{A}} \exp\left(\eta P_t^{(n)}(s,a')\right)}. \tag{34}$$

where $\hat{P}_t^{(n)} = \hat{\Phi}_t^{(n)}\theta$. The Fast-food DPP update rule is now given by:

$$\hat{P}_{t+1}^{(n)}(s,a) = \hat{P}_t^{(n)}(s,a) + r_{ss'}^a - \mathcal{M}_\eta \hat{P}_t^{(n)}(s) + \gamma \mathcal{M}_\eta \hat{P}_t^{(n)}(s'). \tag{35}$$

Solving the least-square problem analogous to Eq.(28) yields:

$$\hat{\theta}^{*(n)} = \arg\min_{\hat{\theta}} J(\hat{\theta}) = [\hat{\Phi}^{(n)^T}\hat{\Phi}^{(n)} + \sigma^2 I]^{-1}\hat{\Phi}^{(n)^T}\mathcal{O}\hat{P}_t^{(n)}. \tag{36}$$

Algorithm 2: Factorial Fast-food Dynamic Policy Programming

Input: $N, \tau, \gamma, \eta, \sigma, T, I$

Output: weight vectors $\theta^{(n)}, n = 1, \ldots, N$

1 Divide $\mathcal{A}$ into $\{\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(\mathcal{N})}\}$
2 Initialize policy $\pi_{explore} = [\pi^{(1)}_{random}, \ldots, \pi^{(N)}_{random}]^T$
3 Compute Fast-food matrix V using τ
4 **for** $i = 1, \ldots, I$ **do**
5 **for** $t = 1, \ldots, T$ **do**
6 **for** $n = 1, \ldots, N$ **do**
7 $x^{(n)} = [s^i_t, a^{(n)}]^T$
8 *compute* $\phi^{(n)}(x)$ *using Fast-food*
9 *compute* $\hat{P}^{(n)}_t$ *following Eq.(35)*
10 *sample action* $a^{(n)}_t$ *from Eq.(34)*
11 *collect* $x^{(n)}$ *in sample buffer* $\mathcal{D}^{(n)}_i$
12 **end**
13 **end**
14 **for** $n = 1, \ldots, N$ **do**
15 *compute* $\hat{\Phi}^{(n)}_t$ *using* $\mathcal{D}^{(n)}_i$
16 *compute nth weight vector* $\hat{\theta}^{(n)}_{t+1}$ *using Eq.(36)*
17 **end**
18 **end**

The aforementioned components are unified in Alg. 2. Lines 1 and 2 establish the factorial multi-agent framework and initialize each agent with exploration policy. To obtain information about the environment, the exploration policy is set to random search, accomplished by all agents independently choosing random actions $\pi^{(n)}_{random}$. The Fast-food approximation feature map is computed using the user-design parameter τ in line 3. The user needs to specify the number of episodes I for the learning loop beginning at line 4. For every time step t and every agent n, a sample is encountered and the corresponding feature vector, action preference are computed. Then the sample is stacked in the sample pool $\mathcal{D}^{(n)}_i$ for policy update after the episode. Lines 14 to 17 correspond to the FFDPP training (36) to produce weight vectors that are approximated counterparts of (15). After learning, all factorial weight vectors are grouped together to produce the learned near-optimal policy:

$$\pi^*(a|s) = [\pi^*(a^{(1)}|s), \ldots, \pi^*(a^{(N)}|s)]^T. \tag{37}$$

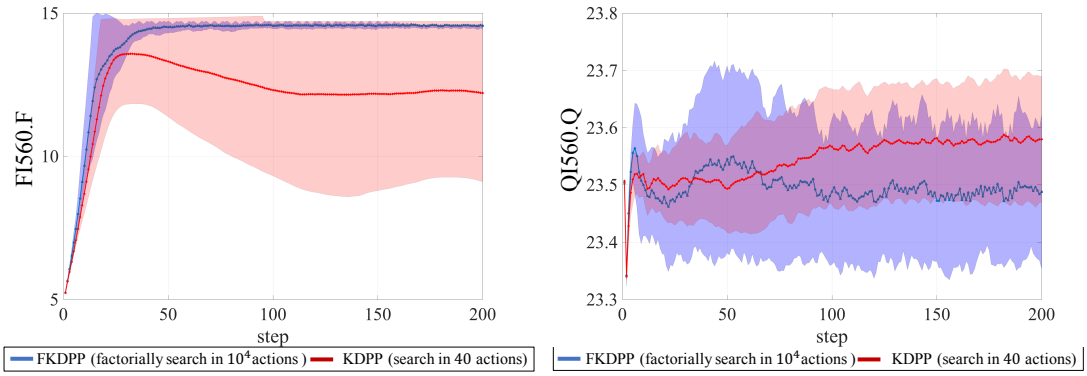
5. Experiment

5.1 Setup for Problem 1 using FKDPP

In this section, FKDPP is applied to the task introduced in previous chapter. To demonstrate the effectiveness of factorial policy, KDPP is also tested to yield a comparison with FKDPP. According to Table 1 and 2, the state space has nine dimensions including control state (FI560.F, LI550.L, s407.F, QI531.Q and QI560.Q) and control parameters (FC550.SVM, TC540.SVM, TC501.SVM and PC501.SVM). The discrete action set is defined as increasing/decreasing four control parameters by 10 discrete actions ($M = 10$) following: $a_{\text{FC550}} \in [-2, 2]$, $a_{\text{TC540}} \in [-4, 4]$, $a_{\text{TC501}} \in [-10, 10]$ and $a_{\text{PC540}} \in [-2, 2]$. The total number of actions is close to 10^4, resulting in an intractable computation in Eq. (10). Therefore, FKDPP consider only $M \times N = 40$ action combinations. For FKDPP, the action space with 10^4 actions is factorized by four agents. For each agent, there are $M = 10$ actions. The VAM plant simulation used in this experiment is detailed in [31], which is implemented on the commercial dynamic simulator, Visual Modeler, developed by Omega Simulation Co., Ltd. Each algorithm is trained over 30 iterations, with each iteration consisting of 200 steps. Each step simulates approx. 30 minutes due to lengthy chemical processes. The reward function used by both methods is defined as:

$$\begin{aligned} R = {} & 30 \times \text{FI560.F} - 2 \times \text{LI550.L} - 5 \times \text{QI560.Q} \\ & - 20 \times \text{QI531.Q}. \end{aligned} \tag{38}$$

It follows the general definition of reward in Eq.(16) of giving a high reward when FI560.F is increased (VAM quantity up) or QI560.Q, QI531.Q are decreased (VAM quality up). Note that the quality measurer readings are minus terms because they are inversely related to the product quality. LI550.L is the stability regularizer that penalizes unstable performance that causes its level to get close to 100%. If any critical condition in Table 1 is violated, the reward is sharply decreased. The experiment is conducted on a PC with processor i7-8700k, 3.70GHz and 32GB memory.



(a) Mean and variance of VAM yield during 30 iterations' learning.

(b) Mean and variance of VAM quality (QI560.Q) during 30 iterations' learning. Lower values indicate a higher quality.

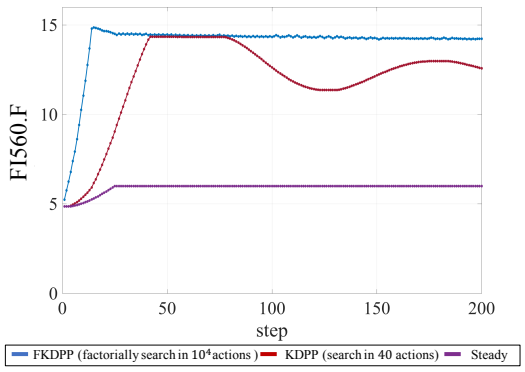
Figure 6. Mean and variance comparison between FKDPP and KDPP

5.2 Result of FKDPP on Problem 1

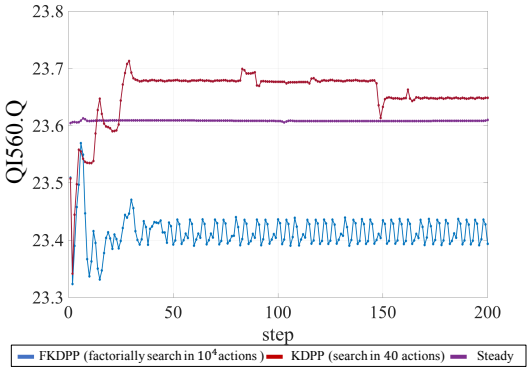
Figure 6a and 6b show the mean and variance of VAM yield (FI560.F) and quality (QI560.Q) for each 200-step period during 30 iterations of learning. FKDPP quickly converged to a good policy that maximized production (Fig. 6a), while KDPP resulted in a lower mean and larger variance due to system instability (e.g. depleting the tank or causing other catastrophic failure scenarios). FKDPP also has a lower mean of QI560.Q during learning compared with KDPP, as shown in Fig. 6b.

After 30 learning iterations, the policies of FKDPP and KDPP are tested. According to Fig. 7a and 7b, FKDPP outperformed KDPP in both VAM quantity and quality. The sequences of control parameters are shown in Fig. 8a and 8b, where FKDPP facilitates exploration with multiple control parameters at each step. On the other hand, KDPP only operates a single control parameter at each step, which is insufficient in terms of exploration, especially when each step lasts for 30 minutes. In Fig. 8b, KDPP has to use more iterations to operate each dimension of the action, resulting in worse performance. These results show that under identical learning conditions, the proposed FKDPP achieves superior performance compared with methods in which the policy is not factorized.

In terms of computational complexity for this task with its nine-dimensional state and action set for four control parameters, FKDPP took an average 0.0017

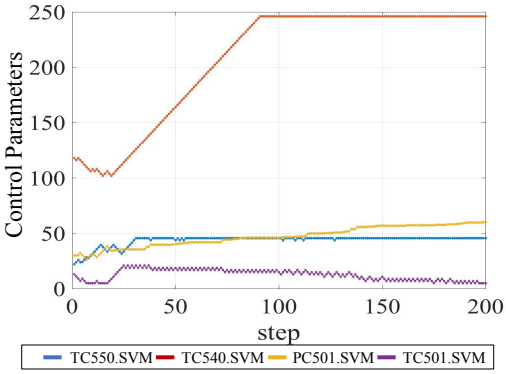


(a) VAM yield with policies after 30 iterations' learning. The purple line is the benchmark model-based controller parameters.

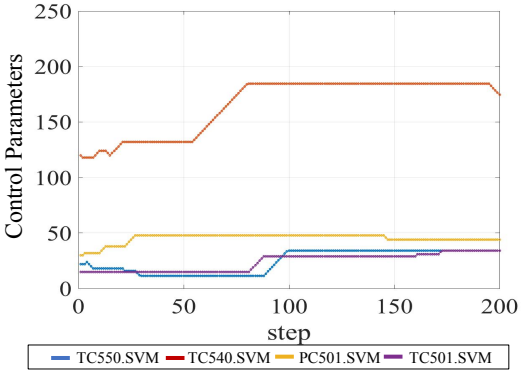


(b) VAM quality with policies after 30 iterations' learning. The purple line is the benchmark model-based controller parameters.

Figure 7. Performance comparison between FKDPP, KDPP and the model-based controller



(a) Control actions following FKDPP policies after 30 iterations' learning.



(b) Control actions following KDPP policies after 30 iterations' learning.

Figure 8. Control policy comparison between FKDPP, KDPP

s per step to search over 10^4 actions while KDPP took 0.0014 s to consider 10×4 actions. Because FKDPP reduces the computational complexity by limiting the number of action combinations (Eq. 10), computation time for FKDPP with 10 actions each is negligibly slower in comparison to running a single agent with 40 actions in the case of KDPP.

From the above comparisons it is obvious that FKDPP is more suitable for tasks that require flexible control strategies. When manipulating one control unit

at a time step is not sufficient, factorial policy might provide better performance in terms of learned policy, sufficient exploration and quick convergence.

As a concluding remark for this chapter, it is worth pointing out that KDPP has demonstrated its effectiveness and efficiency in high dimensional problems with *homogeneous* state spaces, as shown in [36, 32]. On the other hand, KDPP on high dimensional problems with *heterogeneous* state spaces, i.e., each dimension has drastically different range of values, is an unexamined topic. Since in the VAM process control problem 1 only part 6 and part 7 are optimized, with other parts being *locked*, the state space is indeed homogeneous. We point out that in the VAM process control problem 2 (Chapter 3.3), where the goal is to achieve plant-wide autonomous control, the state space is heterogeneous, and KDPP fails to achieve reasonable performance due to this property.

5.3 Setup for Problem 2 using FFDPP

Following the discussion in Chapter 3.2 and Eq.(16) for designing proper reward function, the reward to the VAM process control problem 2 is crafted as:

$$\text{reward} = Merit - 10 \times QI531.Q - 20 \times QI560.Q, \tag{39}$$

where the term *Merit* is the reading directly indicates how much profit the process is producing every hour. This setup is more general than the hand-crafted reward in Chapter 3.2.

Following the assertion in Chapter 4.2.2 that the assumption of 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 $a^{(1)} = \mathbf{FC501}$, $a^{(2)} = \mathbf{PC501}$, $a^{(3)} = \mathbf{TC501}$, $a^{(4)} = \mathbf{TC540}$, and $a^{(5)} = \mathbf{FC550}$, respectively. For leveraging control units in the FFDPP setup, again an action set comprising discrete values is defined, such that at every time step, one agent chooses one value from the set to increase or decrease the value of a control unit. The increase/decrease embodies manipulation of the distillation column's steam flow, the decanter's reflux flow rate and other process physical profiles.

To prove the efficacy of the proposed algorithm, ten independent experiments are repeated and averaged to collect statistical evidence. Each experiment consists of 70 iterations and each iteration has 1000 steps. Each time step in simulation corresponds to around five minutes in the real time. Hence 1000 steps is approximately five days of real time plant running. The performance of our algorithm is evaluated in terms of plant-wide stability, profit and computational resources needed in the period of simulated five days.

For each of the M control units, corresponding action set is constructed by uniformly choosing N values from the interval $[-0.01, 0.01]$. We set $N = 10$ to realize precise control. According to the previous discussion, 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 common value function based RL approaches to efficiently explore.

Aside from maximizing accumulated profit as defined in Eq. (39), maintaining process stability is also a learning goal. Stability is defined by violations of the

Table 5. Meta parameters of the FFDPP process control experiment.

Parameter	Description (Number of)	Value
T	steps	1000
I	iterations	70
N	factorial agents	5
M	discrete actions per agent	10
τ	sampling basis functions	100

safety ranges in Table 4. When a violation occurs, a corresponding penalty is added to the reward. The experiment is conducted on a PC with processor i7-8700k, 3.70GHz and 32GB memory.

The meta parameters of the experiement are summarized in Table 5. Parameter τ controls the balance of Fast-food approximation accuracy and learning speed and needs to be specified by the user. We set $\tau = 100$ and refer interested readers to [54] for technical details.

5.4 Result of FFDPP

The learning performance evolving is shown in the left part of Figure 9, averaged over ten independent learning results for statistical evidence. The blue curve shows the mean reward, while the transparent blue area depicts variance. The y-axis is the accumulated reward given in Eq. (38) summed over 1000 steps of every iteration. The black dashed line marked by A show the performance at 10th iteration; green dashed line B the performance at 40th iteration; yellow line C the performance at 70th iteration. On the right are subplots illustrating the performance evolving through iteration: after 10, 40 and 70 iterations. The y-axes are: *QI531*-(ppm), *QI560*-(ppm), *VAM*-(t/h), *Profit*-(10^3 yen/h).It can be seen from the figure that after trial-and-error learning and without any prior knowledge about the model, the learned FFDPP policy successfully learns a control policy to maximize the reward, solving the process control problem of 13-dimension state space and 5-dimension action space in 70 iterations within 70,000 samples.

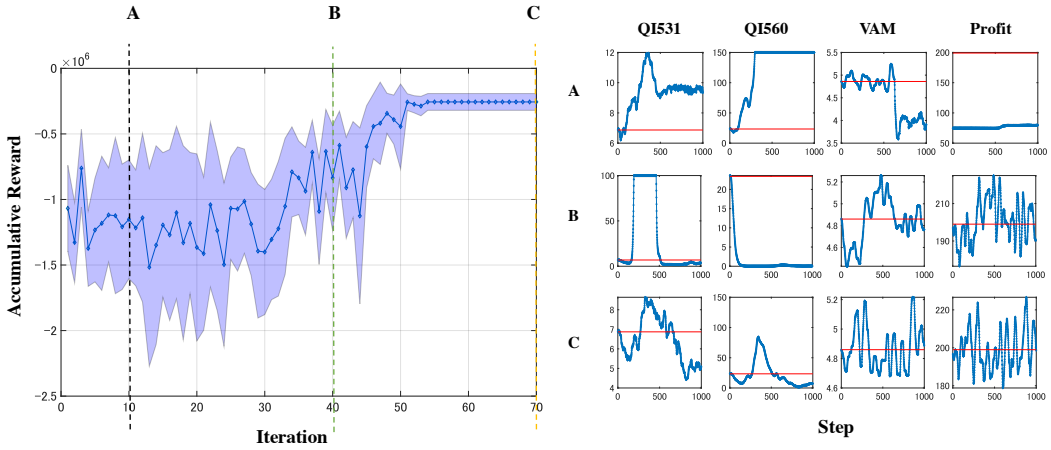


Figure 9. Average cumulative rewards averaged over ten independent experiments with $\tau = 100$.

To visualize the correspondence between states and learning progress, t-SNE [55] is applied to compress the 13 dimensional state vectors into two dimensions to draw the scatterplot in Figure 10. Undesirable states, i.e., overflow of QI531 or QI560, low production rate, etc., are marked with blue, desirable states are marked in red. Markers *A,B,C* refer to the performance plots in Figure 9. By trial-and-error learning the agent explores 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, but cannot control it to be within the optimal regions. Likewise, optimal states might be reached by some policies, but undesirable states will be experienced along the paths. Optimal policy is derived as circling regularly within the optimal region.

The performance of the learned policy is further examined in Figure 11. Comparison is made between performance of the learned RL agent and of the model-based controller proposed in [27, 31]. The red lines indicate performance under the model-based controller which have minor oscillation, while the blue curves show adaptations made by RL agent. The black dashed lines are the mean values of corresponding blue curves. Figure 11a and 11b show the readings of two product quality measurers. Their readings are inversely proportional to the product quality. The product quality controlled by the learned policy better on average

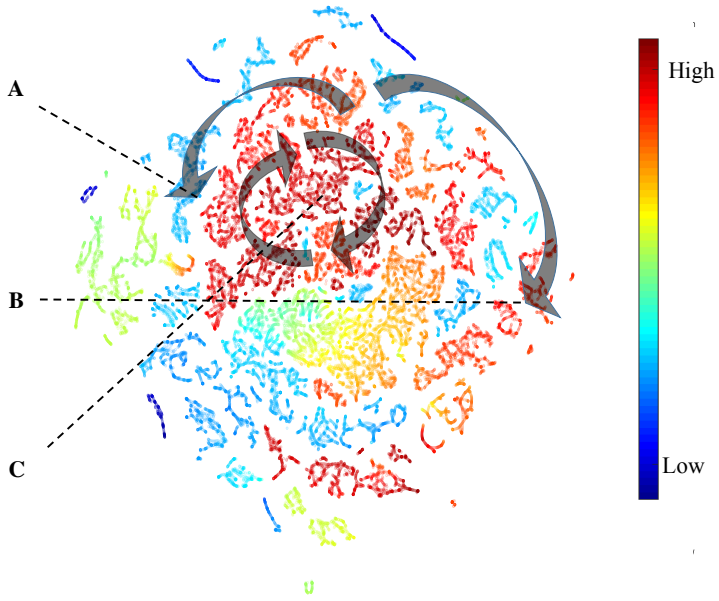
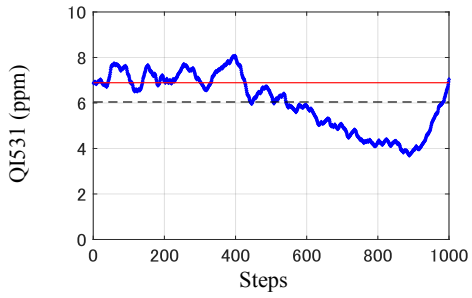


Figure 10. The t-SNE plot shows a trajectory of the agent in one experiment.

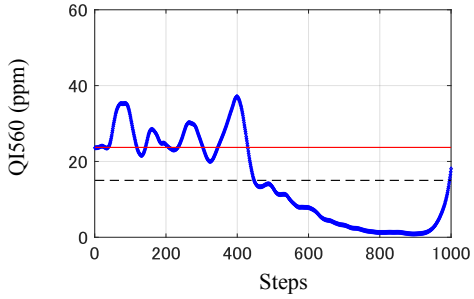
than that of the model-based controller in the simulated period. Figures 11c and 11d show the VAM production rate and profit per time unit, respectively. As the result of adapting to the process dynamics, the curves are changing, hence the mean values are evaluated to be compared with performance of the model-based controller in terms of VAM produced and profit made.

In summary, without any model knowledge, in the simulated period corresponding to around 5 real days, the FFDPP agent succesfully learns a policy that yields comparative performance to the state-of-the-art model-based controller.

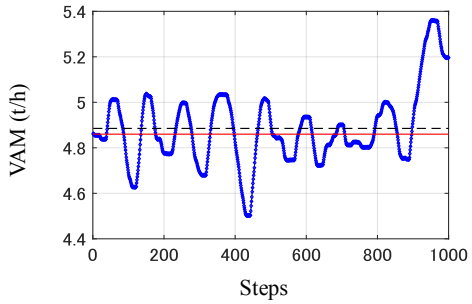
The computational mitigation brought by the factorial framework and Fastfood kernel approximation is examined in Table 6, which compares the computational resources that DPP-based algorithms require. Q-learning, a classic RL



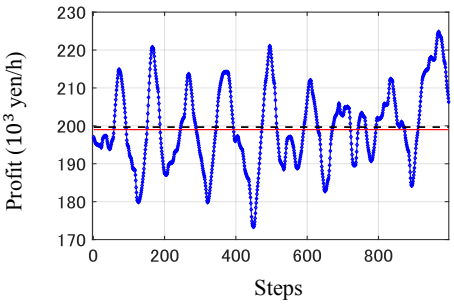
(a) Reading of the product quality measurer QI531 under learned policy. Reading is inversely proportional to product quality.



(b) Reading of the product quality measurer QI560 under learned policy. Reading is inversely proportional to product quality.



(c) VAM production rate under the learned policy.



(d) Profit per time unit under the learned policy.

Figure 11. The comparison between learned RL policy and the model-based controller.

Table 6. Performance comparison between Q-learning, DPP, KDPP, FKDPP and FFDPP on the problem 2.

Algorithm	Acceptable number of samples	Maximum computation time (s)	Maximum policy evaluation time (s)	Success	Error
Q-Learning	500	2287.5	1.2	No	Yes
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
FFDPP	>70000	31.9 × 5	0.0044 × 5	Yes	No

algorithm, is also evaluated to emphasize the importance of more scalable methods. The configurations for KDPP and FKDPP respectively follow the definition in [36] and [32], with the exception the sample threshold is set to $\eta = 0.9$, which

is equivalent to discarding 90% of samples and retaining the rest 10% most informative ones. All parameters of the compared algorithms were tuned to yield best performance empirically.

We define *success* as the capability to learn a policy that does not violate any constraint define in Table 3 within the preset 70 iterations and vice versa. An indicator *Error* is also defined as the situation where the computing software reports error due to intractable computational time or memory usage resulted from the large amount of high dimensional samples. The second column of Table 6 shows the maximum number of samples that each algorithm can handle before triggering the *Error* indicator. The computation time, policy evaluation time are evaluated with the maximum acceptable number of samples. For factorial algorithms, time costs are recorded for one agent, hence the total time cost is that cost multiply by the number of agents.

It is worth noting that the large acceptable number of samples of FFDPP not only relies on Fast-food, but also on the factorial policy for decomposing the one-time evaluation of huge number of action preferences to several sequential evaluations.

The performance of the learned FFDPP policy is also analyzed from the process control perspective. Specifically, we inspect the relationship between actions made and state changes of one rollout in Figure 12. 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 FFDPP policy changes greatly: all five PID controllers marked with red are actuated to increase VAM production. Due to this attempt, in the period marked by the long, yellow rectangle B the temperature distribution of the distillation column changes, *T501.Temp[10]* drops quickly as the result of agent increasing the the amount of VAM reflux to the top of the distillation column using *FC550* (marked by the blue dashed rectangle in period B). As the *T501.Temp[10]* enters a steady zone in the period C, the amount of reflux and steam are almost constant. As a result of *T501.Temp[10]* staying at a lower level than the normal behavior, *T501.Temp[11]* is also affected, quickly dropping to around the same level as *T501.Temp[10]*. To restore the normal temperature distribution, controller *FC501* is actuated to feed more steam into the distillation column, as can be seen from the blue dashed rectangle of *FC501*

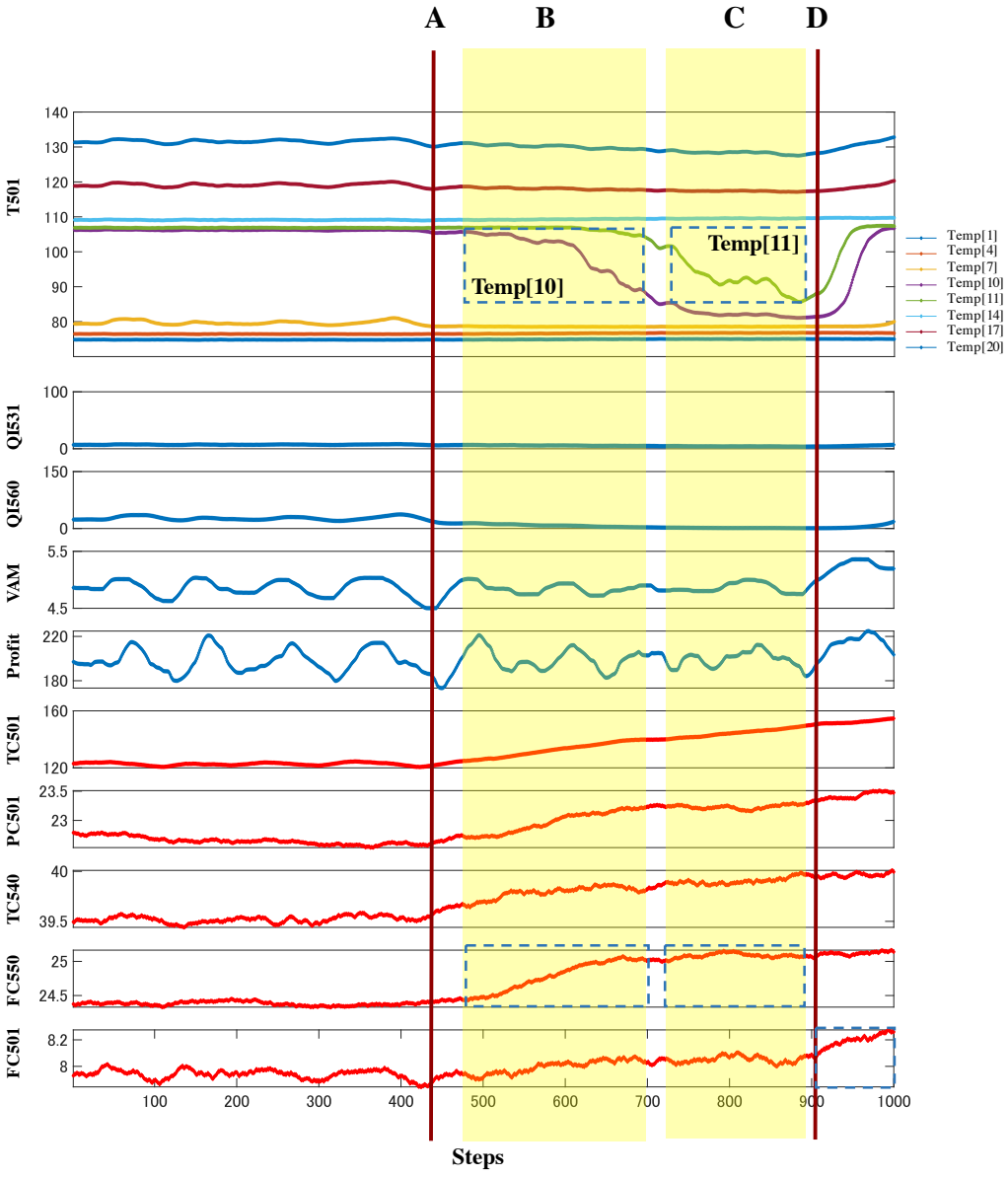


Figure 12. The analysis of controlled behavior using the learned FFDPP policy.

right after the straight line D.

In summary, the operation sequence of the learned FFDPP policy on the temperature distribution of the distillation column is considered to be similar to an operator who has the ability to look inside the column and monitor the temperature distribution in real time.

6. Discussion

Application. From the application viewpoint, FKDPP and FFDPP are successfully applied to a commercial VAM simulator that comprises eight subsystems for materials feeding, reacting and recycling. Two process control problems were solved by FKDPP agent and FFDPP agent. FKDPP solved Problem 1 featuring a 9-dimension state space and four controllers manipulating the part 6 and part 7 of the VAM process, with total 4^{10} possible actions. FFDPP further improves the scalability in terms of number of samples capable of handling, computational resources required and approximation error. FFDPP was successfully applied to Problem 2 featuring a plant-wide control problem with 13-dimensional state space and five controllers manipulating the dynamics changes in the same parts as in Problem 1, with 5^{10} action candidates in total. However, situations handled by FFDPP were far more difficult since tiny perturbations may cause significant changes to the environment for the learning agent. In both problems the aim was to improve the production and product quality while maintaining the process stability. As the first step towards real-world application, the proposed algorithms successfully learns a model-free policy that performs comparatively to the state-of-the-art model-based controller within given number of iterations, demonstrating the potential of FKDPP/FFDPP application on the autonomous control of large-scale chemical plants.

Algorithm. From the algorithmic perspective, two novel RL algorithms suitable for large-scale system control problems were proposed. Since FKDPP has large in common with FFDPP, the summary of the advantages of FFDPP is discussed here. The efficacy of FFDPP 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; factorial policy for decomposing large action space into independent ones by assuming the conditional independence between control units, which holds for process control problems since the units are designed to be independent; and DPP for ensuring the smoothness and convergence in large-scale system by exploiting the Kullback-Leibler relative entropy between the baseline policy and the current policy.

Performance. The performance curves of the learned FFDPP policy in Figure 11 change as the result of the choice of reward function, this is because

the reward function Eq. (39) of the investigated control problem does not involve terms that restrict the variance of performance. On the other hand, designing a generally proper reward function for large-scale process control problems remain an open issue. We leave the investigation of variance-related reward function to future work.

7. Conclusion

The contributions of this research thesis are twofold. From the algorithmic perspective, two novel RL algorithms that are applicable to problems with high dimensional state space and large-scale discrete action set. FFDPP adopts:

1. Fast-food method to efficiently approximate state value function of high dimensionality, significantly reducing the computational complexity.

2. Factorial framework that divides a large set of discrete actions and share samples between agents, reducing computational complexity without affecting the exploration range.

3. Smooth policy update based on the Kullback-Leibler divergence to unify the Fast-food and the factorial framework, realizing both high sample efficiency and learning stability.

FKDPP shares point (1) and (3) in common and replaces the fast-food approximation scheme with the concept of reduced kernel set.

The future work naturally focuses on applying the proposed algorithms on a real-world VAM process to realize model-free control. Real-world application is more complicated and since various factors, e.g., safety, reading errors, control errors might happen and there are many more complications needed to be considered. On the other hand, exploiting RL in a realistic environment could be beneficial for attaining insights for model-based design and applying RL on other real-world problems.

Acknowledgements

I would like to thank my supervisor Professor Sugimoto, Professor Matsubara for supervision they provided me during my master course. Many thanks also go to Dr.Cui for his help and guidance. I also would like to extend my heartfelt thanks for my family for their care.

Sincere thanks also go to Yokogawa Inc. for their technical support and the commercial simulator which we based our experiments on. Secretary Hayashi, Fujimoto, Nishimura helped a lot on handling official matters and free us from tons of paperworks.

References

[1] M. Kano and M. Ogawa, "The State of the Art in Advanced Chemical Process Control in Japan," *IFAC Proceedings Volumes*, vol. 42, no. 11, pp. 10–25, 2009.

[2] M. Dotoli, A. Fay, M. Miśkowicz, and C. Seatzu, "Advanced control in factory automation: a survey," *International Journal of Production Research*, vol. 55, no. 5, pp. 1243–1259, 2017.

[3] M. Metzger and G. Polakow, "A survey on applications of agent technology in industrial process control," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 570–581, 2011.

[4] K. J. Aström and P. Kumar, "Control: A perspective," *Automatica*, vol. 50, no. 1, pp. 3 – 43, 2014.

[5] K. Fujiwara, M. Kano, S. Hasebe, and A. Takinami, "Soft-sensor development using correlation-based just-in-time modeling," *AIChE Journal*, vol. 55, no. 7, pp. 1754–1765, 2009.

[6] E. F. Camacho and C. A. Bordons, *Model Predictive Control in the Process Industry*. Berlin, Heidelberg: Springer-Verlag, 1997.

[7] R.-E. Precup and H. Hellendoorn, "A survey on industrial applications of fuzzy control," *Computers in Industry*, vol. 62, no. 3, pp. 213 – 226, 2011.

[8] K. Hunt, D. Sbarbaro, R. Żbikowski, and P. Gawthrop, "Neural networks for control systems–a survey," *Automatica*, vol. 28, no. 6, pp. 1083 – 1112, 1992.

[9] M. Kano and Y. Nakagawa, "Data-based process monitoring, process control, and quality improvement: Recent developments and applications in steel industry," *Computers and Chemical Engineering*, vol. 32, no. 1, pp. 12 – 24, 2008.

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

[11] A. Yang, B. Braunschweig, E. Fraga, Z. Guessoum, W. Marquardt, O. Nadjemi, D. Paen, D. Piñol, P. Roux, S. Sama, M. Serra, and I. Stalker, "A multi-agent system to facilitate component-based process modeling and design," *Computers and Chemical Engineering*, vol. 32, no. 10, pp. 2290 – 2305, 2008.

[12] Y. Qian, Q. Huang, W. Lin, and X. Li, "An object/agent based environment for the computer integrated process operation system," *Computers and Chemical Engineering*, vol. 24, no. 2, pp. 457 – 462, 2000.

[13] H. Wang and Y. Zhang, "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, 2008.

[14] Y. S. Ng and R. Srinivasan, "Multi-agent based collaborative fault detection and identification in chemical processes," *Engineering Applications of Artificial Intelligence*, vol. 23, no. 6, pp. 934 – 949, 2010.

[15] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[16] J. C. Hoskins and D. M. Himmelblau, "Process control via artificial neural networks and reinforcement learning," *Computers and Chemical Engineering*, vol. 16, no. 4, pp. 241–251, 1992.

[17] J. H. Lee and W. Wong, "Approximate dynamic programming approach for process control," *Journal of Process Control*, vol. 20, no. 9, pp. 1038–1048, 2010.

[18] W. Liu, Y. Tan, and Q. Qiu, "Enhanced Q-learning algorithm for dynamic power management with performance constraint," in *the Conference on Design, Automation and Test in Europe*, pp. 602–605, 2010.

[19] D. Ernst, M. Glavic, P. Geurts, and L. Wehenkel, "Approximate Value Iteration in the Reinforcement Learning Context. Application to Electrical Power System Control," *International Journal of Emerging Electric Power Systems*, vol. 3, no. 1, pp. 1–35, 2005.

[20] S. Syafiie, F. Tadeo, and E. Martinez, "Model-free learning control of neutralization processes using reinforcement learning," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 6, pp. 767 – 782, 2007.

[21] S. A. Harp, S. Brignone, B. F. Wollenberg, and T. Samad, "Sepia. a simulator for electric power industry agents," *IEEE Control Systems Magazine*, vol. 20, no. 4, pp. 53–69, 2000.

[22] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] S. P. K. Spielberg, R. B. Gopaluni, and P. D. Loewen, "Deep reinforcement learning approaches for process control," in *International Symposium on Advanced Control of Industrial Processes (AdCONIP)*, pp. 201–206.

[24] S. Kubosawa, T. Onishi, and Y. Tsuruoka, "Synthesizing chemical plant operation procedures using knowledge, dynamic simulation and deep reinforcement learning," in *SICE Annual Conference*, pp. 1376–1379, 2018.

[25] R. E. Bellman, *Dynamic Programming*. New York, NY, USA: Dover Publications, Inc., 2003.

[26] M. L. Luyben and B. D. Tyréus, "An industrial design/control study for the vinyl acetate monomer process," *Computers and Chemical Engineering*, vol. 22, no. 7-8, pp. 867–877, 1998.

[27] H. Seki, M. Ogawa, T. Itoh, S. Ootakara, H. Murata, Y. Hashimoto, and M. Kano, "Plantwide control system design of the benchmark vinyl acetate monomer production plant," *Computers and Chemical Engineering*, vol. 34, no. 8, pp. 1282–1295, 2010.

[28] R. Chen, K. Dave, T. J. McAvoy, and M. Luyben, "A nonlinear dynamic model of a vinyl acetate process," *Industrial and Engineering Chemistry Research*, vol. 42, no. 20, pp. 4478–4487, 2003.

[29] D. G. Olsen, W. Y. Svrcek, and B. R. Young, "Plantwide control study of a vinyl acetate monomer process design," *Chemical Engineering Communications*, vol. 192, no. 10, pp. 1243–1257, 2005.

[30] W. L. Luyben, "Design and control of a modified vinyl acetate monomer process," *Industrial & Engineering Chemistry Research*, vol. 50, no. 17, pp. 10136–10147, 2011.

[31] Y. Machida, S. Ootakara, H. Seki, Y. Hashimoto, M. Kano, Y. Miyake, N. Anzai, M. Sawai, T. Katsuno, and T. Omata, "Vinyl Acetate Monomer (VAM) Plant Model: A New Benchmark Problem for Control and Operation Study," *International Federation of Automatic Control (IFAC)*, vol. 49, no. 7, pp. 533–538, 2016.

[32] Y. Cui, L. Zhu, M. Fujisaki, H. Kanokogi, and T. Matsubara, "Factorial kernel dynamic policy programming for vinyl acetate monomer plant model control," in *IEEE International Conference on Automation Science and Engineering (IEEE-CASE)*, pp. 304–309, 2018.

[33] Y. Cui, T. Matsubara, and K. Sugimoto, "Pneumatic artificial muscle-driven robot control using local update reinforcement learning," *Advanced Robotics*, pp. 1–16, 2017.

[34] Y. Tsurumine, Y. Cui, E. Uchibe, and T. Matsubara, "Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation," *Robotics and Autonomous Systems*, vol. 112, pp. 72 – 83, 2019.

[35] T. Matsubara, V. Gómez, and H. J. Kappen, "Latent Kullback-Leibler Control for Continuous-State Systems using Probabilistic Graphical Models," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 583–592, 2014.

[36] Y. Cui, T. Matsubara, and K. Sugimoto, "Kernel Dynamic Policy Programming: Applicable Reinforcement Learning to Robot Systems with High Dimensional States," *Neural networks*, vol. 94, pp. 13–23, 2017.

[37] Q. V. Le, T. Sarlos, and A. Smola, "Fastfood-computing hilbert space expansions in loglinear time," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

[38] M. G. Azar, V. Gómez, and H. J. Kappen, "Dynamic policy programming with function approximation," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 119–127, 2011.

[39] M. G. Azar, V. Gómez, and H. J. Kappen, "Dynamic policy programming," *The Journal of Machine Learning Research (JMLR)*, vol. 13, no. 1, pp. 3207–3245, 2012.

[40] E. Todorov, "Linearly-solvable Markov decision problems," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 1369–1376, 2006.

[41] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. 2005.

[42] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 1038–1044, 1996.

[43] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 1057–1063, 1999.

[44] E. Uchibe and K. Doya, "Inverse reinforcement learning using dynamic policy programming," in *International Conference on Development and Learning and on Epigenetic Robotics*, pp. 222–228, IEEE, 2014.

[45] X. Xu, D. Hu, and X. Lu, "Kernel-Based Least Squares Policy Iteration for Reinforcement Learning," *IEEE Transactions on Neural Networks*, vol. 18, no. 4, pp. 973–992, 2007.

[46] T. Jung and D. Polani, "Kernelizing lspe," in *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 338–345, April 2007.

[47] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *The Journal of Machine Learning Research*, vol. 4, no. 44, pp. 1107–1149, 2003.

[48] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *SIAM Journal on Control and Optimization*, pp. 1008–1014, MIT Press, 2000.

[49] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7, pp. 1180–1190, 2008.

[50] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor-critic algorithms," *Automatica*, vol. 45, no. 11, pp. 2471 – 2482, 2009.

[51] D. Achlioptas, F. McSherry, and B. Schölkopf, "Sampling techniques for kernel methods," in *NIPS*, 2001.

[52] A. Blum, "Random projection, margins, kernels, and feature-selection," in *Subspace, Latent Structure and Feature Selection* (C. Saunders, M. Grobelnik, S. Gunn, and J. Shawe-Taylor, eds.), (Berlin, Heidelberg), pp. 52–68, Springer Berlin Heidelberg, 2006.

[53] A. Frieze, R. Kannan, and S. Vempala, "Fast monte-carlo algorithms for finding low-rank approximations," *Journal of the ACM*, vol. 51, no. 6, pp. 1025–1041, 2004.

[54] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 1177–1184, 2008.

[55] L. van der Maaten, "Accelerating t-sne using tree-based algorithms," *Journal of Machine Learning Research*, vol. 15, pp. 3221–3245, 2014.