2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Learning Switched Koopman Models for Control of Entity-Based Systems

Madeline Blischke and João P. Hespanha

Abstract— **One problem that arises in control engineering is that of controlling a system for which the dynamics are unknown. Such a problem favors a data-driven approach, such as can be done through the use of the Koopman operator. We present a switched Koopman model, applicable to systems with discrete sets of inputs, that gives rise to an optimal control problem with a piecewise affine value function. This structure provides an efficient representation and enables a heuristic pruning algorithm that avoids the exponential complexity of finding the true optimal solution.**

We use density-like observables that are defined through the notion of entity-based systems: systems whose state is composed of a possibly varying number of entities that can be grouped into classes of like-entities. This encompasses many systems, including arcade games, a common benchmark used in reinforcement learning. We find that our approach requires much less training than commonly used for reinforcement learning. The Koopman approach also has the advantage of being agnostic to the control objective, which allows the objective to be changed without needing to retrain the model.

I. Introduction

Many dynamical systems have dynamics whose model is either unknown, or otherwise is too complicated for use in control design. Nonetheless, the need arises to implement controllers for such systems. Such scenarios favor a model-free, data-driven approach. One commonly used method is reinforcement learning, in which an agent learns a value function via interacting with the system. An alternative is to learn a model from data, and then to apply model-based control design. The latter is the approach taken here, through the use of the Koopman operator.

The Koopman operator, introduced in [1], provides a linear representation for nonlinear systems via "lifting" of the dynamics into an infinite-dimensional function space. For use in numerical computation and analysis, a finite matrix representation of the operator can be approximately learned from data [2]. The Koopman operator has seen a great deal of interest in recent years, including extensions to systems with control [3]–[6]. While much work has considered systems with continuous inputs, resulting in models with linear [7], [8] or bilinear [9] forms, comparatively little literature exists on systems with discrete inputs. In [10], a continuous input is discretized to construct a switched Koopman model for use in control. Both discrete inputs and states are considered in [11] in the context of hybrid systems.

We formalize a novel class of systems that we refer to as *entity-based systems*, which is amenable to kernel-based observables. The key feature of these systems is that their state is composed of some (possibly varying) number of entities that can be grouped into classes of like-entities. This can describe many systems, such as players in strategy games, vehicles in traffic control problems, data packets in computer networks, or individuals in a dynamic population. The observables we define rely on evaluating kernel functions between entities and fixed points in their state spaces. These are well-suited to capturing the dynamics of entity-based systems, as they share the symmetry of the underlying system with respect to permutations of like-entities, and remain well-defined as the number of entities present changes. As will be seen in an example, these observables can be effective even when the entities are high-dimensional.

The Koopman model we obtain is a switched linear system, for which standard control approaches apply (see [12] for a survey). It is most natural when dealing with the Koopman operator to consider a cost that is linear in the lifted state, such as would be obtained if the cost were to be included as a Koopman observable. Linear cost functions yield useful properties not shared by nonlinear cost functions, yet have not been studied extensively in the literature. They have been considered, for example, in [13], which uses properties of the matrix trace to develop a branch and bound algorithm.

We show that, for a linear switched system with a linear cost function, the optimal value function is piecewise linear on the state space. This is similar to results on partially observed Markov processes [14], which also yield a piecewise linear value function. As the Koopman model we obtain will have some error, we show that the worst-case cost is still piecewise affine in the presence of small perturbations to the dynamics. There may also be error in measuring the state or in performing the lifting, which we show also leads to a worst-case cost that is piecewise affine. Due to the discrete nature of the problem, the computational complexity of find the true optimal solution is exponential in the length of the horizon considered. We introduce an algorithm that finds an approximate solution to the optimal control problem via use of a heuristic that dynamically prunes input sequences using a combination of exploration and exploitation.

It is useful to compare the Koopman operator approach with that of reinforcement learning (see [15] for a survey). Whereas reinforcement learning focuses on simultaneously

M. Blischke and J. P. Hespanha are with the University of California, Santa Barbara. Email: {blischke,hespanha}@ucsb.edu. This material is based upon work supported by the National Science Foundation under grant no. 2229876 and is supported in part by funds provided by the National Science Foundation, by the Department of Homeland Security, and by IBM. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

learning the value function and exploiting it to maximize an accrued reward, the Koopman approach used here separates these two aspects. One advantage of the combined approach is that a control policy can be incrementally improved, which in some environments may be necessary to explore regions of the state space where higher rewards can be achieved. On the other hand, an advantage of the separate approach is that the model can be trained from any data, and is agnostic of the intended objective. Thus, one can implement controllers that optimize different objectives without needing any additional training.

The rest of the paper is structured as follows. Section II presents background on the Koopman operator framework. Section III formalizes the notion of an entity-based system and introduces an appropriate type of observable for such systems. Section IV discusses properties of the optimal control problem and presents a heuristic search algorithm. Section V demonstrates our methods for a simple *Pong*-like game and for the Atari 2600 game *Assault*. Section VI provides concluding remarks.

II. The Koopman Operator

We consider dynamical systems of the form

$$x^+ = f(x,u), \qquad y = g(x), \tag{1}$$

for state $x \in \mathcal{X}$, controlled output $y \in \mathbb{R}$, and input $u \in \mathcal{U}$, where $\mathcal{U}$ is a finite set of size $n_c := |\mathcal{U}|$, and x^+ denotes the state at the next time step. The Koopman operator acts on a function space $\mathcal{F}$ of mappings from $\mathcal{X}$ into $\mathbb{R}$, referred to as *observables*. Formally, we define a family of Koopman operators, parameterized by u, as

$$\mathcal{K}_u \phi(x) := \phi(f(x,u)), \qquad \phi \in \mathcal{F}.$$

That is, each $\mathcal{K}_u$ maps observables to their composition with the state dynamics (1) when input u is applied. We say that a subspace $\tilde{\mathcal{F}} \subseteq \mathcal{F}$ is *Koopman-invariant* if

$$\mathcal{K}_u \phi \in \tilde{\mathcal{F}}, \qquad \forall \phi \in \tilde{\mathcal{F}},\ \forall u \in \mathcal{U},$$

and we say that a so-called *dictionary* of observables $\mathbf{\Psi} : \mathcal{X} \to \mathbb{R}^N$ is Koopman-invariant if its elements span a Koopman-invariant subspace. Since evaluation of the dictionary often involves a "lifting" of a low-dimensional state vector into a higher dimensional space, the vector $\mathbf{\Psi}(x)$ is commonly referred to as the *lifted state*.

We briefly summarize key properties of the Koopman operator used in this paper. Each $\mathcal{K}_u$ is a linear operator on $\mathcal{F}$, since

$$\begin{aligned}\mathcal{K}_u(\alpha_1\phi_1(x) + \alpha_2\phi_2(x)) &= \alpha_1\phi_1(f(x,u)) + \alpha_2\phi_2(f(x,u)) \\ &= \alpha_1\mathcal{K}_u\phi_1(x) + \alpha_2\mathcal{K}_u\phi_2(x).\end{aligned}$$

When applying a finite input sequence $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_k) \in \mathcal{U}^k$ from an initial state x_0, we can compose the $\mathcal{K}_u$ to obtain

$$\phi(x_k) = \mathcal{K}_{\mu_k}\mathcal{K}_{\mu_{k-1}} \cdots \mathcal{K}_{\mu_1}\phi(x_0). \tag{2}$$

Given a Koopman-invariant dictionary $\mathbf{\Psi}$, linearity of $\mathcal{K}_u$ guarantees that there exist matrices $A_u \in \mathbb{R}^{N\times N}$ such that

$$\mathcal{K}_u \mathbf{\Psi}(x) = \mathbf{\Psi}(f(x,u)) = A_u \mathbf{\Psi}(x). \tag{3}$$

Composition as in (2) then yields

$$\mathbf{\Psi}(x_k) = A_{\mu_k}A_{\mu_{k-1}} \cdots A_{\mu_1}\mathbf{\Psi}(x_0). \tag{4}$$

When the output map g is in the span of $\mathbf{\Psi}$, then there exists a vector $c \in \mathbb{R}^N$ such that

$$g(x) = \mathbf{c}^T \mathbf{\Psi}(x). \tag{5}$$

While every system has some Koopman-invariant subspace, (e.g. the space of constant functions), it is generally hard to find a subspace (and associated dictionary $\mathbf{\Psi}$) that is "useful" in the sense of both being Koopman-invariant and containing the output map g. In practice, this means that often one needs to work with dictionaries for which (3) and (5) can not hold exactly, but instead can hold approximately with some error that depends on how close $\mathbf{\Psi}$ is to being Koopman-invariant.

A. Data-Driven Approximation

The matrices A_u and vector $\mathbf{c}$ in (3) and (5) can be constructed from data through an algorithm called *extended dynamic mode decomposition* (EDMD) [2].

The EDMD algorithm, slightly modified here to include inputs and outputs, requires some number of data snapshots

$$\{x_i, \hat{x}_i, y_i, u_i\}_{i=1}^M$$

satisfying

$$\hat{x}_i = f(x_i, u_i), \qquad y_i = g(x_i),$$

and a dictionary of observables $\mathbf{\Psi}$. The snapshots may be taken from one long trajectory (in which case we have $\hat{x}_i = x_{i+1}$), or from several shorter trajectories. The ordering of the snapshots is irrelevant.

We define the sets $\mathcal{I}(u) = \{i : u_i = u\}$, which can be used to partition the data snapshots into n_c subsystems, each corresponding to one fixed input $u \in \mathcal{U}$. We then apply the EDMD algorithm to each of these subsystems individually by solving

$$A_u = \operatorname*{arg\,min}_{A \in \mathbb{R}^{N\times N}} \sum_{i \in \mathcal{I}(u)} \|A\mathbf{\Psi}(x_i) - \mathbf{\Psi}(\hat{x}_i)\|^2 \tag{6}$$

for each u. These solutions can be obtained without storing the full set of snapshots, by computing

$$A_u = H_u G_u^\dagger,$$

where

$$\begin{aligned}G_u &= \sum_{i \in \mathcal{I}(u)} \frac{1}{|\mathcal{I}(u)|} \mathbf{\Psi}(x_i)\mathbf{\Psi}(x_i)^T, \\ H_u &= \sum_{i \in \mathcal{I}(u)} \frac{1}{|\mathcal{I}(u)|} \mathbf{\Psi}(\hat{x}_i)\mathbf{\Psi}(x_i)^T,\end{aligned}$$

and $G_u^\dagger$ denotes the Moore-Penrose pseudoinverse. The vector $\mathbf{c}$ for approximating the output function can be similarly obtained as the solution to

$$\mathbf{c} = \operatorname*{arg\,min}_{\mathbf{c} \in \mathbb{R}^N} \sum_{i=1}^M \|\mathbf{c}^T\mathbf{\Psi}(x_i) - y_i\|^2. \tag{7}$$

III. Entity-Based Systems

We consider systems for which the state x of the overall system (1) includes the states of a finite number of entities, which may be grouped into distinct classes of like-entities. Each class can be identified with a set-valued mapping $\mathcal{E}_i : \mathcal{X} \rightrightarrows \mathbb{R}^{p_i}$ such that $\mathcal{E}_i(x)$ is the set of states of entities within that class. By encoding the entities as a set, $\mathcal{E}_i(x)$ does not encode which states correspond to which entities, and so would remain unchanged if the states were to be permuted among the entities. If the system is such that two entities may share the same state, then $\mathcal{E}_i$ should instead be considered multiset-valued, so that states are unordered but with multiplicities preserved.

We say that the system (1), together with entity mappings $\mathscr{E} = (\mathcal{E}_1, \ldots, \mathcal{E}_{n_\mathcal{E}})$, is an *entity-based system* if there exist functions F and G such that

$$\mathscr{E}(f(x,u)) = F(\mathscr{E}(x), u), \tag{8}$$

$$g(x) = G(\mathscr{E}(x)). \tag{9}$$

The key property described by (8) and (9) is that the sets of entities are sufficient to both predict their evolution and to compute the output. That is, the evolution of the system depends on the sets of states of the entities, but not on which specific state belongs to which entity. An advantage of this representation over a representation of x as a vector in Euclidean space is that it allows for systems where the number of entities may vary over time.

Note that no special structure or properties are required of $\mathcal{X}$, and so entity-based systems can describe systems with arbitrary state spaces. If we do have $\mathcal{X} \subseteq \mathbb{R}^n$ for a fixed n then we can always describe the system as an entity-based system, such as by a single entity with $\mathcal{E}_1(x) = \{x\}$, or by n entities $\mathcal{E}_i(x) = \{x_i\}$. However, finding a nontrivial characterization may still have utility in terms of dimensionality reduction as a result of exploiting symmetries among entities within the same class.

To emphasize the importance of the choice of $\mathscr{E}$, consider a system with state $x \in \mathbb{R}^3$, $u \in \mathbb{R}$, where

$$f(x,u) = \begin{bmatrix} x_1 + u \\ ux_3 \\ ux_2 \end{bmatrix}, \qquad g(x) = x_1 - x_2 x_3.$$

This is not an entity-based system for, say, $\mathcal{E}_1(x) = \{x_1, x_2\}$ and $\mathcal{E}_2(x) = \{x_3\}$, as this choice erases the necessary distinction between x_1 and x_2. However, it is an entity-based system for $\mathcal{E}_1(x) = \{x_1\}$ and $\mathcal{E}_2(x) = \{x_2, x_3\}$, as x_2 and x_3 behave indistinguishably. Explicitly, we have

$$\begin{aligned} \mathcal{E}_1(f(x,u)) &= \mathcal{E}_1(x) + u, \\ \mathcal{E}_2(f(x,u)) &= u\mathcal{E}_2(x), \\ g(x) &= \mathcal{E}_1(x) - \prod_{z \in \mathcal{E}_2(x)} z. \end{aligned}$$

Another example of an entity-based system that highlights these properties is the game of chess, for which same-color pieces of the same type can be grouped into classes of like-entities. The positions of pieces within each class define the evolution of the game, and the number of pieces present decreases over time. Additionally, whereas pieces from different classes cannot be interchanged (e.g. a pawn and a queen), nothing would change if, say, we permuted the locations of the white pawns.

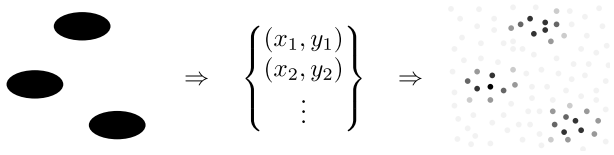


Fig. 1. Construction process of the lifted state $\mathbf{\Psi}(x)$. Entity coordinates are extracted from the state (e.g. a game screen), from which densities are computed at each of a fixed set of basis centers.

A. Choice of Dictionary

Due to the dependence of the evolution of entity-based systems on the state through $\mathscr{E}(x)$ as described by (8) and (9), it suffices to consider dictionaries that share this dependence, i.e.

$$\mathbf{\Psi}(x) = \tilde{\mathbf{\Psi}}(\mathscr{E}(x))$$

for some function $\tilde{\mathbf{\Psi}}$. We propose observables based on kernel functions between entity states and some set of fixed points $\{\mathbf{b}_{ij}\}_{j=1}^{N_{b,i}} \subseteq \mathbb{R}^{p_i}$ that are chosen via a K-means clustering over some number of trajectories. For each $\mathbf{b}_{ij}$, we define an observable

$$\psi_{ij}(x) = \sum_{\mathbf{e} \in \mathcal{E}_i(x)} w_{ij}(\mathbf{e}) \kappa_i(\mathbf{e}, \mathbf{b}_{ij}), \tag{10}$$

where w_{ij} is a chosen weighting function and κ_i is a chosen kernel function. The dictionary $\mathbf{\Psi}$ is then constructed by stacking the ψ_{ij} for each entity class $\mathcal{E}_i$.

One appropriate choice of kernel is the Gaussian radial basis function, defined as

$$\kappa(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\ell^2}\right)$$

where ℓ is a chosen lengthscale parameter. This kernel produces larger values for entities that are nearer to the given basis center. The resulting observables then capture relative densities of the entities, such as is illustrated in Figure 1. An appropriate weighting function may be set as 0 when $\mathbf{e}$ is far from $\mathbf{b}_{ij}$, and 1 otherwise, to promote sparsity in the dictionary $\mathbf{\Psi}$.

IV. Optimal Control Problem

Given an initial state x_0, we treat the outputs $y_k = g(x_k)$ as cumulative costs at each stage that we wish to minimize over some horizon h:

$$\min_{\boldsymbol{\mu} \in \mathcal{U}^h} \quad \sum_{k=0}^{h} g(x_k) \tag{11a}$$

$$\text{s.t.} \quad x_k = f(x_{k-1}, \mu_k), \quad k = 1, \ldots, h. \tag{11b}$$

Using (3) and (5), this optimization problem can be expressed equivalently in the lifted state space as

$$\min_{\boldsymbol{\mu}\in\mathcal{U}^h} \quad \sum_{k=0}^{h} \mathbf{c}^T\boldsymbol{\xi}_k \tag{12a}$$

$$\text{s.t.} \quad \boldsymbol{\xi}_0 = \boldsymbol{\Psi}(x_0), \tag{12b}$$

$$\boldsymbol{\xi}_k = A_{\mu_k}\boldsymbol{\xi}_{k-1}, \quad k = 1,\dots,h. \tag{12c}$$

In practice, (3) and (5) may not hold exactly, so that there may be a gap between the values of (11) and (12).

For simplicity of presentation, we consider a cost function that is stage-invariant and depends only on the current state x_k. However, this can be easily extended to stage-dependent and input-dependent cost functions, i.e. those of the form $g_k(x_k, u_k)$, by replacing the $\mathbf{c}$ vectors with appropriate $\mathbf{c}_k(u_k)$ vectors.

We define the optimal cost-to-go at stage k from state $\boldsymbol{\xi}_k$ by

$$J_k(\boldsymbol{\xi}_k) := \min_{\boldsymbol{\mu}^k\in\mathcal{U}^{h-k}} \sum_{\ell=k}^{h} \mathbf{c}^T\boldsymbol{\xi}_\ell \qquad \text{s.t. (12c)} \tag{13}$$

where $\boldsymbol{\mu}^k := (\mu_{k+1},\dots,\mu_h)$ denotes the tail of the sequence $\boldsymbol{\mu}$ from stage k, with the understanding that $\boldsymbol{\mu}^h$ is the empty sequence and $\boldsymbol{\mu}^0 = \boldsymbol{\mu}$. Then we have the following result, which shares similarities to results on the optimal control of partially observable Markov processes [14], which also yield a piecewise linear value function.

Theorem 1: Let $J_k(\boldsymbol{\xi}_k)$ denote the optimal cost-to-go of (12) from state $\boldsymbol{\xi}_k$ at stage k. Then J_k is piecewise linear and given by

$$J_k(\boldsymbol{\xi}_k) = \min_{\boldsymbol{\mu}^k\in\mathcal{U}^{h-k}} \mathbf{d}(\boldsymbol{\mu}^k)^T\boldsymbol{\xi}_k \tag{14}$$

where, for $\boldsymbol{\mu}^k = (\mu_{k+1},\dots,\mu_h)$,

$$\mathbf{d}(\boldsymbol{\mu}^k) = \sum_{\ell=k}^{h} A_{\mu_{k+1}}^T \cdots A_{\mu_\ell}^T \mathbf{c}. \tag{15}$$

Moreover, the optimal control to (12) can be obtained as any $\boldsymbol{\mu}^0$ that minimizes (14) for $k = 0$.

Proof: For a given sequence $\boldsymbol{\mu}^k = (\mu_{k+1},\dots,\mu_h)$, recursive application of (12c) yields the cost at each stage as $\mathbf{c}^T\boldsymbol{\xi}_\ell = \mathbf{c}^T A_{\mu_\ell}\cdots A_{\mu_{k+1}}\boldsymbol{\xi}_k$. Summing these costs for stages k to h and minimizing over $\boldsymbol{\mu}^k$ thus yields the desired form for J_k. That the optimal control to (12) can be obtained as any $\boldsymbol{\mu}^0$ that minimizes (14) for $k = 0$ then follows directly from the definition of the cost-to-go in (13). ∎

Note also that the $\mathbf{d}(\boldsymbol{\mu}^k)$ vectors can be constructed recursively by initializing $\mathbf{d}(\boldsymbol{\mu}^h) = \mathbf{c}$, and then computing

$$\mathbf{d}(\boldsymbol{\mu}^{k-1}) = A_{\mu_k}^T\mathbf{d}(\boldsymbol{\mu}^k) + \mathbf{c}, \qquad k \in \{h,\dots,1\}. \tag{16}$$

A. Incorporating Robustness

As discussed previously, using the Koopman model for prediction introduces error unless $\boldsymbol{\Psi}$ is Koopman-invariant, which is generally not the case. As such, it is likely desirable to incorporate some notion of robustness to our objective function to mitigate this error. The form we consider here is to optimize for the worst-case cost when (12c) holds only approximately, with an input-dependent bound on the error that satisfies $\|\boldsymbol{\xi}_k - A_{\mu_k}\boldsymbol{\xi}_{k-1}\| \le \varepsilon_{\mu_k}$. With this change, the problem (12) becomes

$$\min_{\boldsymbol{\mu}\in\mathcal{U}^h} \quad \max_{\mathbf{v}_1,\dots,\mathbf{v}_h\in\mathbb{R}^N} \sum_{k=0}^{h} \mathbf{c}^T\boldsymbol{\xi}_k \tag{17a}$$

$$\text{s.t.} \quad \boldsymbol{\xi}_0 = \boldsymbol{\Psi}(x_0), \tag{17b}$$

$$\boldsymbol{\xi}_k = A_{\mu_k}\boldsymbol{\xi}_{k-1} + \mathbf{v}_k, \quad k = 1,\dots,h, \tag{17c}$$

$$\|\mathbf{v}_k\| \le \varepsilon_{\mu_k}, \quad k = 1,\dots,h. \tag{17d}$$

Theorem 2: Let $J_{\varepsilon,k}(\boldsymbol{\xi}_k)$ denote the optimal worst-case cost-to-go of (17) from state $\boldsymbol{\xi}_k$ at stage k. Then $J_{\varepsilon,k}$ is piecewise affine and given by

$$J_{\varepsilon,k}(\boldsymbol{\xi}_k) = \min_{\boldsymbol{\mu}^k\in\mathcal{U}^{h-k}} \mathbf{d}(\boldsymbol{\mu}^k)^T\boldsymbol{\xi}_k + \sum_{\ell=k+1}^{h} \varepsilon_{\mu_\ell}\|\mathbf{d}(\boldsymbol{\mu}^\ell)\|$$

where $\mathbf{d}(\boldsymbol{\mu}^k)$ is as in (15).

Proof: For a given $\boldsymbol{\mu}^k = (\mu_{k+1},\dots,\mu_h)$ and $\mathbf{v}_{k+1},\dots,\mathbf{v}_h \in \mathbb{R}^N$, recursive application of (17c) yields the cost at each stage as

$$\mathbf{c}^T\boldsymbol{\xi}_\ell = \mathbf{c}^T A_{\mu_\ell}\cdots A_{\mu_{k+1}}\boldsymbol{\xi}_k + \sum_{m=k+1}^{\ell} \mathbf{c}^T A_{\mu\ell}\cdots A_{\mu_{m+1}}\mathbf{v}_m.$$

Summing these costs for stages k to h then yields

$$\sum_{\ell=k}^{h} \mathbf{c}^T\boldsymbol{\xi}_\ell = \mathbf{d}(\boldsymbol{\mu}^k)^T\boldsymbol{\xi}_k + \sum_{m=k+1}^{h} \mathbf{d}(\boldsymbol{\mu}^m)^T\mathbf{v}_m. \tag{18}$$

Now we maximize this sum subject to (17d). Note that the contribution of each $\mathbf{v}_k$ is decoupled, so that maximizing (18) is achieved by maximizing each $\mathbf{d}(\boldsymbol{\mu}^m)^T\mathbf{v}_m$ separately. For any vector $\mathbf{d}$, we have

$$\max_{\|\mathbf{v}\|\le\varepsilon} \mathbf{d}^T\mathbf{v} = \max_{\|\mathbf{v}\|=1} \varepsilon\mathbf{d}^T\mathbf{v} = \varepsilon\mathbf{d}^T\frac{\mathbf{d}}{\|\mathbf{d}\|} = \varepsilon\|\mathbf{d}\|, \tag{19}$$

and therefore

$$\max_{\substack{\mathbf{v}_{k+1},\dots,\mathbf{v}_h\in\mathbb{R}^N \\ \text{s.t. (17d)}}} \sum_{\ell=k}^{h} \mathbf{c}^T\boldsymbol{\xi}_\ell = \mathbf{d}(\boldsymbol{\mu}^k)^T\boldsymbol{\xi}_k + \sum_{m=k+1}^{h} \varepsilon_{\mu_m}\|\mathbf{d}(\boldsymbol{\mu}^m)\|.$$

Minimization over $\boldsymbol{\mu}^k$ thus completes the proof. ∎

One can alternatively consider robustness to error between $\boldsymbol{\xi}_0$ and $\boldsymbol{\Psi}(x_0)$. Such error may be introduced in evaluating $\boldsymbol{\Psi}$, or in measuring the state x_0 itself. We now optimize the worst-case when (12b) holds only approximately as $\|\boldsymbol{\xi}_0 - \boldsymbol{\Psi}(x_0)\| \le \varepsilon$. This yields the problem

$$\min_{\boldsymbol{\mu}\in\mathcal{U}^h} \quad \max_{\mathbf{v}\in\mathbb{R}^N} \sum_{k=0}^{h} \mathbf{c}^T\boldsymbol{\xi}_k \tag{20a}$$

$$\text{s.t.} \quad \boldsymbol{\xi}_0 = \boldsymbol{\Psi}(x_0) + \mathbf{v}, \tag{20b}$$

$$\|\mathbf{v}\| \le \varepsilon, \tag{20c}$$

$$\boldsymbol{\xi}_k = A_{\mu_k}\boldsymbol{\xi}_{k-1}, \quad k = 1,\dots,h. \tag{20d}$$

Note that the cost-to-go at stage $k \ge 1$ is equal to that of (12), and only the cost from stage $k = 0$ differs.

Corollary 1: The solution to (20) is equivalent to that of

$$\min_{\boldsymbol{\mu}\in\mathcal{U}^h} \mathbf{d}(\boldsymbol{\mu})^T\boldsymbol{\Psi}(x_0) + \varepsilon\|\mathbf{d}(\boldsymbol{\mu})\|. \tag{21}$$

where $\mathbf{d}(\boldsymbol{\mu})$ is as in (15).

Proof: By Theorem 1, the initial cost-to-go has the form (14) with respect to $\boldsymbol{\xi}_0 = \boldsymbol{\Psi}(x_0) + \mathbf{v}$. By (19), the worst-case with respect to $\mathbf{v}$ is then (21). ■

B. Dynamic Pruning Algorithm

One approach to computing the optimal control is to exhaustively precompute the $\mathbf{d}(\boldsymbol{\mu})$ vectors for each possible input sequence, yielding ${n_c}^h$ vectors. The problem (12) would then be solved by finding the $\boldsymbol{\mu}$ that minimizes $\mathbf{d}(\boldsymbol{\mu})^T\boldsymbol{\Psi}(x_0)$. However, due to the exponential dependence on h, this approach quickly becomes intractable for longer horizon lengths. One alternative is to use a branch and bound method as is described in [13], but this algorithm is too slow for use in real-time control.

We propose a suboptimal method, described in Algorithm 1, that approximates the value function by computing only a tractable subset of the $\mathbf{d}(\boldsymbol{\mu})$ vectors. First, a set Υ_h is constructed containing only the empty sequence. Then, starting with $k = h-1$, we construct the set Υ_k by prepending each input $u \in \mathcal{U}$ to each sequence $\boldsymbol{\mu}^{k+1} \in \Upsilon_{k+1}$, and concurrently compute the corresponding vectors $\{\mathbf{d}(\boldsymbol{\mu}^k) : \boldsymbol{\mu}^k \in \Upsilon_k\}$ using (16). If the size of the constructed set is greater than some fixed constant L, then the set is pruned via a combination of exploration and exploitation.

The exploitation phase consists of implementing a receding horizon algorithm, which chooses among the sequences in Υ_k, from an initial state $\boldsymbol{\xi}_0$. Starting at $\ell = 0$, we compute

$$(\mu_{k+1},\dots,\mu_h) = \arg\min_{\boldsymbol{\mu}^k\in\Upsilon_k} \mathbf{d}(\boldsymbol{\mu}^k)^T\boldsymbol{\xi}_\ell, \tag{22}$$
$$\boldsymbol{\xi}_{\ell+1} = A_{\mu_{k+1}}\boldsymbol{\xi}_\ell,$$

which obtains the best sequence in Υ_k at state $\boldsymbol{\xi}_\ell$ and applies it to obtain the next state $\boldsymbol{\xi}_{\ell+1}$. We also construct a set $\Upsilon_{K\text{-best},\ell}$ containing the K sequences in Υ_k that yield the K lowest values in (22), which will be included in the pruned set. The above process is repeated for each $\ell \in \{0, h-1\}$, resulting in sets $\Upsilon_{K\text{-best},\ell}$ containing up to hK input sequences in total.

Ideally, Υ_k would contain the best input sequences of length $h-k$ for states $\boldsymbol{\xi}$ in the reachable region of the lifted state space. If this were the case, then (22) would find the optimal control to (12) from $\boldsymbol{\xi}$ at stage k, and its value would be equal to $J_k(\boldsymbol{\xi})$ as defined in (13). It is likely not the case that the best input sequence to every state can be saved, but we hope that some input sequence achieves close to the optimal value. Since the input sequences in Υ_k are used as a starting point from which to construct the sets Υ_j for $j < k$, this should lead to good performance from stage k onward even if the performance up to that point is poor.

In the exploration phase, R random input sequences are chosen from Υ_k (regardless of how they perform in terms of the minimization in (22)). This allows the algorithm to find sequences that initially look "bad" but that turn out good over the full horizon. The pruned set Υ_k is then taken as the union of this set and of the sets $\Upsilon_{\text{best},\ell}$. The parameters should thus be chosen so that $hK+R \le L$, so that the pruned set is guaranteed to contain no more than L sequences.

The above process is repeated for each $k \in \{h-1,\dots,0\}$, as shown in Algorithm 1. The input sequence to a given state x_0 is then chosen as $\boldsymbol{\mu} = \arg\min_{\boldsymbol{\mu}\in\Upsilon_0} \mathbf{d}(\boldsymbol{\mu})^T\boldsymbol{\Psi}(x_0)$.

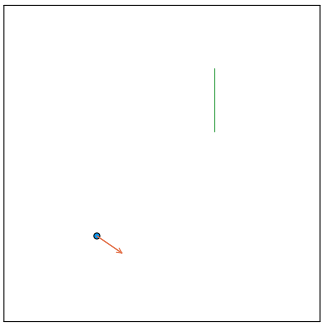


Fig. 2. Illustration of the simple Pong game. The blue dot is the ball's position, and the orange arrow is its velocity. The green line is the paddle.

Algorithm 1 Dynamic Pruning

$\Upsilon_h \leftarrow \{\boldsymbol{\mu}^h\}$ ▷ empty sequence
for $k \in \{h-1,\dots,0\}$ **do**
  $\Upsilon_k \leftarrow \{(u, \boldsymbol{\mu}^{k+1}) : u \in \mathcal{U}, \boldsymbol{\mu}^{k+1} \in \Upsilon_{k+1}\}$
  Compute $\{\mathbf{d}(\boldsymbol{\mu}^k) : \boldsymbol{\mu}^k \in \Upsilon_k\}$ using $\{\mathbf{d}(\boldsymbol{\mu}^{k+1}) : \boldsymbol{\mu}^{k+1} \in \Upsilon_{k+1}\}$ and (16)
  if $|\Upsilon_k| > L$ **then**
    for $\ell \in \{0,\dots,h-1\}$ **do**
      $(\mu_{k+1},\dots,\mu_h) \leftarrow \arg\min_{\boldsymbol{\mu}^k\in\Upsilon_k} \mathbf{d}(\boldsymbol{\mu}^k)^T\boldsymbol{\xi}_\ell$
      $\boldsymbol{\xi}_{\ell+1} \leftarrow A_{\mu_{k+1}}\boldsymbol{\xi}_\ell$
      $\Upsilon_{K\text{-best},\ell} \leftarrow K$ minimizers of (22)
    end for
    $\Upsilon_k \leftarrow \text{sample}(\Upsilon_k, R) \cup \bigcup_{\ell=0}^{h_{cl}-1} \Upsilon_{K\text{-best},\ell}$
  end if
end for

V. Experimental Results

We apply our techniques to two different arcade games. The first is a simple *Pong*-like game implemented in Julia. The second is the Atari 2600 game *Assault*.

A. Simple Pong

The first example we consider is a simplified version of the game *Pong*. The game consists of a square box containing a "ball" represented as a point, and a "paddle" with width w. The ball moves within the box, colliding elastically off of the walls and the paddle. Note that the ball can collide off both the "front" or the "back" of the paddle. At each time step, the player chooses to move the paddle vertically by an amount $u \in \{-w/2, 0, w/2\}$.

The state of this game is fully determined by the ball position, velocity, and paddle height. The full state is taken

as a single entity that exists in $\mathbb{R}^5$, and we choose $N_b = 500$ basis centers for the observables in (10). Whereas a grid based method for selecting the centers in $\mathbb{R}^5$ would lead to a prohibitively large lifted state due to the high dimension, the K-means clustering identifies a relatively small number of centers that turn out to be sufficient for predicting the state evolution. We also find that using a single 5-dimensional entity leads to better performance than separating the ball and paddle into separate entities, likely due to tight coupling between the two.

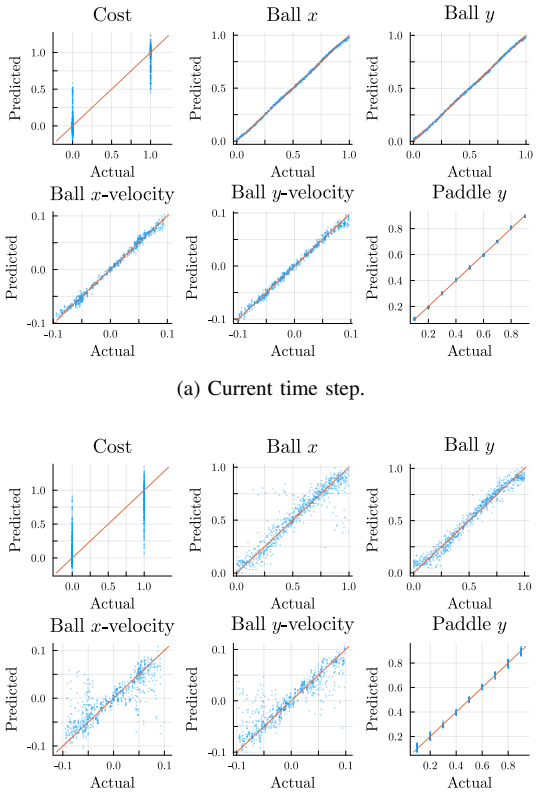
Our goal is to minimize the amount of time that the ball spends on the right-hand side of the paddle. Accordingly, we choose the cost function as an indicator function that is equal to 1 when the ball is to the right of the paddle (x-coordinate greater than $2/3$), and 0 otherwise.

We train the model by simulating, for 10 steps each, initial conditions sampled uniformly with ball position in $[0, 1]^2$ and velocity v with $|v| \in [0.05, 0.1]$ and $\angle v \in [0, 2\pi)$. We run 5,000 such trajectories for constructing the observables, and another 5,000 (equaling 50,000 snapshots) for constructing the EDMD model using (6) and (7). We train two models, one with $w = 1/3$ and one with $w = 1/5$. The prediction accuracy of the model with $w = 1/5$ is illustrated in Figure 3. We see that the general trends of the outputs are captured, both at the current and future time steps, though some error is present, particularly for the cost function. Despite the prediction error, there is still a strong positive correlation between the actual and predicted cost functions, which as we shall see is sufficient for designing an effective controller.

For each paddle width, we design a controller with horizon $h = 6$ that finds the best input sequence to (12) through an exhaustive search over the precomputed vectors $\{\mathbf{d}(\boldsymbol{\mu}) : \boldsymbol{\mu} \in \mathcal{U}^6\}$, implemented in a receding horizon algorithm. We test each controller for 5,000 initial conditions sampled with the x-coordinate equal to zero and an initial velocity with an angle in $[-\pi/3, \pi/3]$. We simulate each initial condition for 100 time steps. We test each controller for the system with the paddle width it was designed for, but also against the system with the other paddle width to observe the performance in a non-nominal system. We also implement robust versions of each controller, which minimize the worst-case cost-to-go of Theorem 2, where we fix $\varepsilon_0 = 10^{-4}$ and vary $\varepsilon_{-w/2} = \varepsilon_{w/2}$ from 10^{-4} to 10^{-5}. These values are small to avoid being overly conservative, since the actual errors are unlikely to match the worst-case errors of (17d).

The fraction of time spent on the right-hand side is shown in Figure 4. For reference, we also provide the performance of random play and of a controller that solves the original problem (11) by simulating the exact nonlinear dynamics for the same horizon of $h = 6$. Note that even the solution to (11) is suboptimal due to its limited horizon. All controllers yield significant improvement over random play for both systems. The designed controllers are also robust to perturbations in the system dynamics, as evidenced by their performance in the off-nominal cases (i.e. those with $w_{nom} \neq w$).

Surprisingly, the controller with $w_{nom} = 1/3$ always performs better than the controller with $w_{nom} = 1/5$, even



(a) Current time step.

(b) 6 time steps in the future.

Fig. 3. Actual vs. predicted outputs at the current time step, and at 6 time steps in the future, for the pong game with $w = 1/5$. Perfect predictions would lie on the line $y = x$, shown in orange. While some error is visible, the general trends are captured.

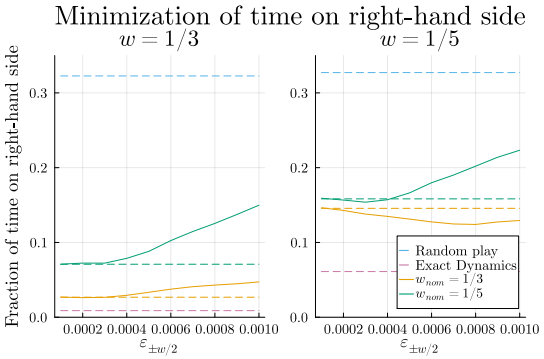


Fig. 4. Performance of various controllers for the Pong game, when the paddle width is $w = 1/3$ or $1/5$. Lower values correspond to better performance. The dashed green and red lines denote the performance achieved by the non-robust controllers, whereas the solid lines are the performance of the robust controllers with $\varepsilon_0 = 10^{-4}$ and with varying values of $\varepsilon_{-w/2} = \varepsilon_{w/2}$. For reference, the blue shows the performance of random play, and the magenta shows that of a controller that solves (11) by simulating the exact nonlinear dynamics for the same horizon $h = 6$.

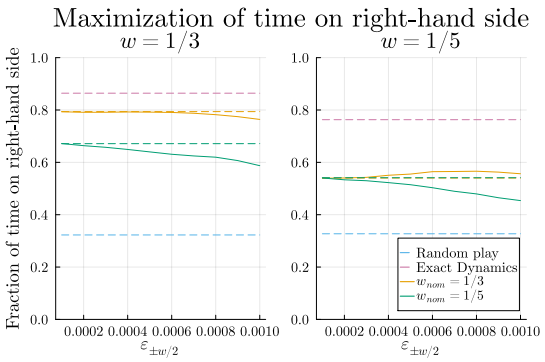


Fig. 5. Performance of various controllers for the Pong game, when the objective is reversed. Now higher values correspond to better performance. Similar trends are seen as compared to Figure 4 for the original objective.

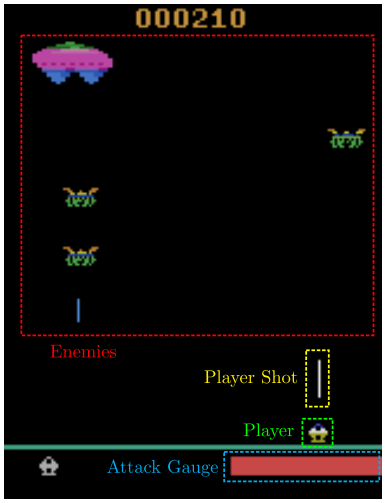


Fig. 6. A screen from the Atari 2600 game *Assault*. Four classes of like-entities (player ship, player shot, enemies, and attack gauge) are indicated.

when the actual width is $1/5$. There are multiple possible explanations for this phenomenon. One is that the larger paddle results in more interactions between the ball and paddle during training, thus producing a Koopman model with better knowledge of the dynamics at those critical points. For another, consider the case with $w = 1/5$ where the ball is just barely within reach of where the paddle can move in the time before the ball crosses to the right-hand side. If the nominal controller incorrectly believes that the ball will be instead just out of reach, then it would conclude that it is unable to prevent the ball from passing, so receives no benefit in trying. On the contrary, the off-nominal controller would believe that it is able to reach the ball in time, so it will move toward it and successfully deflect it.

We additionally see that our robust controllers yield improvements in the system with $w = 1/5$, particularly in the off-nominal case. A small improvement is also seen in the nominal case with proper tuning of the ε_u. In the system with width $1/3$, we do not see any significant improvement to either controller when robustness is incorporated. This could be expected, as the larger and faster paddle results in an easier game which is more forgiving to model errors.

Since the Koopman model was trained using random inputs, it is agnostic of the objective that we choose. As such, the objective can be changed without needing to retrain the model. We illustrate this here using the same model, but now with the objective of maximizing the time spent on the right-hand side of the paddle. The cost function is the same as before, but with the sign reversed. The resulting performance is shown in Figure 5. As before, we see that both controllers yield significant improvement over random play for both systems. We also see that for the off-nominal controller of the $w = 1/5$ system, incorporation of robustness leads to improved performance.

B. Assault

The second example is based on the Atari 2600 game *Assault*. The game consists of a player ship and a number of enemy ships. The player can move horizontally and shoot left, right, and up. When the player shoots, a gauge increases that leads to a loss of life when it is full. The objective of the game is to shoot the enemies while evading enemy fire.

The game is structured into a series of levels, each featuring different enemies. The horizontal shooting of the player is only needed from the fourth level onward, which is hardly ever reached by random play. For simplicity of the resulting model, we thus do not include horizontal shooting during either the training or testing phases.

Note that this game has stochastic elements, which do not appear in our modeling in the previous sections. In this setting, the stochastic Koopman operator maps observables to their expected value at the next time step [16], and this is the operator that is approximated by EDMD [2]. Our control methods then apply as described, with the understanding that they now minimize the *expected* cost.

We take as entities the pixels that make up various in-game features. Specifically, $\mathcal{E}_1$ corresponds to the player ship, $\mathcal{E}_2$ to the player shot, $\mathcal{E}_3$ to the enemies (encompassing the enemies, ship, and shots), and $\mathcal{E}_4$ to the attack gauge. These features can be identified by searching specific areas of the screen for specific colors of pixels. We also include delayed measurements of $\mathbf{\Psi}_{\mathcal{E}_1}$ and $\mathbf{\Psi}_{\mathcal{E}_3}$ so that the Koopman model will have awareness of the player and enemy velocities. We choose the number of basis centers as $N_{b,1} = 50, N_{b,2} = 100, N_{b,3} = 200$, and $N_{b,4} = 16$ respectively. The total size of the dictionary $\mathbf{\Psi}$ is then $N = 616$.

We simulate the game via the Arcade Learning Environment [17] with a frame skip of 2; i.e. a decision is made every other frame. We additionally use "sticky actions" as proposed in [18] with $\varsigma = 0.25$; i.e. at each frame, with probability 0.25, the current input is ignored and replaced by the previous input. We train the observables over 20 games, and train the EDMD model over 500 (totaling 662,668 snapshots, or about 1.3 million frames). We implement a controller constructed by Algorithm 1 with $h = 50$, $L = 10{,}000$, $R = 5{,}000$, and $K = 100$.

We compare the performance of this controller to that of

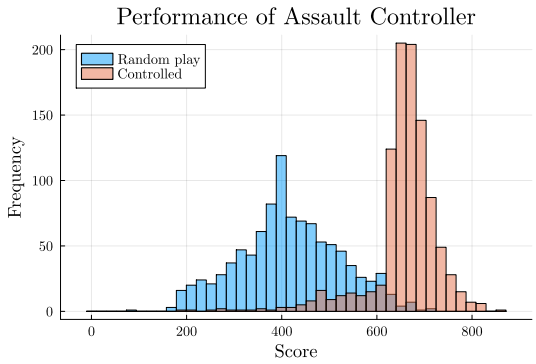


Fig. 7. Comparison of *Assault* scores between random play and a controller constructed via Algorithm 1 with $h = 50$.

random play, testing each on $1{,}000$ games. The distribution of resulting scores is shown in Figure 7. The controller provides more consistent results, and improves the mean score from 416.4 to 657.8. It should be noted that the fourth level is reached at a score of 630, at which point the player will inevitably die due to the restriction imposed on shooting horizontally. The observed performance then may be as good as one could reasonably hope to achieve under the imposed constraint preventing horizontal shooting.

We now compare this score to those achieved in the reinforcement learning literature. In terms of raw performance, we do not approach the performance of state-of-the-art learning algorithms such as [19], [20]. However, the $662{,}668$ training samples we used is much less than the tens of millions of training samples typically used in the literature. Despite this relatively small number of training samples, our controller achieved comparable performance to that of the agents in [18] after 10 million frames (2 million training samples). Also of note is that the mean score we achieve is higher than that of the best linear learner in [19], which, like us, uses a linear approximation for the value function.

VI. Conclusion

We formulated a switched Koopman model with a linear approximation of a cost function, for which we showed that the value function is piecewise linear, or piecewise affine under the worst-case of small perturbations. We also presented a heuristic search algorithm that allows for tractable consideration of longer horizons. We introduced a formalization of entity-based games, which characterize the state by collections of like-entity states, and introduced a class of kernel-based observables appropriate for such systems.

We demonstrated our approach for two arcade games. For a simple deterministic *Pong*-like game, we demonstrated accuracy of state predictions, and designed controllers that are robust to changes in the dynamics. We also demonstrated the ability to change the control objective without needing to perform any additional training. We then considered the Atari 2600 game *Assault*, which is more complicated and contains stochastic elements. For this game, we demonstrated the effectiveness of our heuristic search algorithm, and achieved good performance with a much smaller number of training points than is typically used in reinforcement learning.

References

[1] B. O. Koopman, "Hamiltonian systems and transformation in Hilbert space," *Proceedings of the National Academy of Sciences*, vol. 17, no. 5, pp. 315–318, May 1931.

[2] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A data-driven approximation of the Koopman operator: Extending dynamic mode decomposition," *Journal of Nonlinear Science*, vol. 25, no. 6, pp. 1307–1346, Dec. 2015.

[3] M. Budišić, R. Mohr, and I. Mezić, "Applied Koopmanism," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 4, Dec. 2012.

[4] A. Mauroy, I. Mezić, and Y. Susuki, Eds., *The Koopman Operator in Systems and Control*, 2020.

[5] S. E. Otto and C. W. Rowley, "Koopman operators for estimation and control of dynamical systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 59–87, Feb. 2021.

[6] S. L. Brunton, M. Budišić, E. Kaiser, and J. N. Kutz, "Modern Koopman theory for dynamical systems," *SIAM Review*, vol. 64, no. 2, pp. 229–340, May 2022.

[7] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Dynamic mode decomposition with control," *SIAM Journal on Applied Dynamical Systems*, vol. 15, no. 1, pp. 142–161, Jan. 2016.

[8] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160, July 2018.

[9] D. Bruder, X. Fu, and R. Vasudevan, "Advantages of bilinear Koopman realizations for the modeling and control of systems with unknown dynamics," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4369–4376, July 2021.

[10] S. Peitz and S. Klus, "Koopman operator-based model reduction for switched-system control of PDEs," *Automatica*, vol. 106, pp. 184–191, Aug. 2019.

[11] C. Bakker, A. Bhattacharya, S. Chatterjee, C. J. Perkins, and M. R. Oster, "Learning Koopman representations for hybrid systems," June 2020.

[12] F. Zhu and P. J. Antsaklis, "Optimal control of hybrid switched systems: A brief survey," *Discrete Event Dynamic Systems*, vol. 25, no. 3, pp. 345–364, Sept. 2015.

[13] W. Xu, Z. G. Feng, G.-H. Lin, and L. Yu, "Optimal scheduling of discrete-time switched linear systems," *IMA Journal of Mathematical Control and Information*, vol. 37, no. 1, pp. 1–18, Mar. 2020.

[14] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable Markov processes over a finite horizon," *Operations Research*, vol. 21, no. 5, pp. 1071–1088, 1973.

[15] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[16] I. Mezić and A. Banaszuk, "Comparison of systems with complex behavior," *Physica D: Nonlinear Phenomena*, vol. 197, no. 1, pp. 101–133, Oct. 2004.

[17] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, June 2013.

[18] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *Journal of Artificial Intelligence Research*, vol. 61, pp. 523–562, Mar. 2018.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[20] Y. Liang, M. C. Machado, E. Talvitie, and M. Bowling, "State of the art control of Atari games using shallow reinforcement learning," in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, ser. AAMAS '16. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, May 2016, pp. 485–493.