Markov Chain Monte Carlo for Koopman-Based Optimal Control

João Hespanha, *Fellow, IEEE*, and Kerem Çamsarı, *Senior Member, IEEE*

***Abstract*—We propose a Markov Chain Monte Carlo (MCMC) algorithm based on Gibbs sampling with parallel tempering to solve nonlinear optimal control problems. The algorithm is applicable to nonlinear systems with dynamics that can be approximately represented by a finite dimensional Koopman model, potentially with high dimension. This algorithm exploits linearity of the Koopman representation to achieve significant computational saving for large lifted states. We use a video-game to illustrate the use of the method.**

***Index Terms*—Koopman operator, optimal control, optimization, randomized algorithms, switched systems.**

I. Introduction

WHILE pioneering work is almost a century old, its use as a practical tool to model complex dynamics is much more recent and only became practical when computational tools became available for the analysis of systems with hundreds to thousands of dimensions. The use of Koopman models for control is even more recent, but has attracted significant attention in the last few years [2], [3], [4], [5], [6], [7].

The linear structure of the Koopman representation permits very efficient solutions for optimal control when the lifted dynamics are linear on the control input, as in [2], [3], [4]. However, linearity in the control severely limits the class of applicable dynamics. Bilinear representations are more widely applicable [5], [6], [7], but are also harder to control.

When the set of admissible control inputs is finite, the Koopman representation can be viewed as a switched linear system, where the optimal control selects, at each time step, one out of several admissible dynamics [9]. Solving optimal control problems for switched systems is typically difficult [10], [11], [12], [13], but when the optimization criterion is linear in the lifted state, the dynamic programming cost-to-go is concave and piecewise linear, with a simple representation in terms of a minimum over a finite set of linear functions. This observation enabled the design of efficient algorithms that combine dynamic programming with dynamic pruning [9].

This letter exploits the structure of the Koopman representation to develop efficient Markov Chain Monte Carlo (MCMC) sampling methods for optimal control. Following the pioneering work of [14], [15], [16], we draw samples from a Boltzmann distribution with energy proportional to the criterion to minimize. The original use of MCMC methods for combinatorial optimization relied on a gradual decrease in temperature, now commonly known as *simulated annealing*, to prevent the chain from getting trapped into states that do not minimize energy. An alternative approach relies on the use of multiple replicas of a Markov chain, each generating samples for a different temperature. The introduction of multiple replicas of a Markov chain to improve the mixing time can be traced back to [17]. The more recent form of *parallel tempering* (also known as *Metropolis–coupled MCMC*, or *exchange Monte Carlo*) is due to [18], [19].

Our key contribution is an MCMC algorithm that combines Gibbs sampling [16], [20] with parallel tempering to solve the switched linear optimizations that arise from the Koopman representation of nonlinear optimal control problems. This algorithm is computationally efficient due to the combination of two factors: (i) the linear structure of the cost function enables the full variable sweep need for Gibbs sampling to be performed with computation that scales linearly with the horizon length T and (ii) parallel tempering can be fully parallelized across computation cores, as noted in [21]. With regard to (i), the computational complexity of evaluation Gibbs' conditional distribution for each optimization variable is of order Tn_ψ^2, where n_ψ denotes the size of the lifted state. Our algorithm computes the conditional distributions *for all T variables* in a full Gibbs' sweep with computation still just of order Tn_ψ^2. With regard to (ii), while here we only explore parallelization across CPU cores, in the last few years hardware parallelization using GPUs and FPGAs has achieved orders of magnitude increase in the number of samples generated with MCMC sampling [22], [23].

The remaining of this letter is organized as follows: Section II shows how a nonlinear control problem can be converted into a switching linear systems optimization, using the Koopman operator. Section III provides basic background on MCMC, Gibbs sampling, and parallel tempering. Our optimization algorithm is described in Section IV and its use is illustrated in Section V in the context of a video game. While

Manuscript received 4 March 2024; revised 7 May 2024; accepted 29 May 2024. Date of publication 10 June 2024; date of current version 15 July 2024. This work was supported in part by the U.S. Office of Naval Research MURI under Grant N00014-23-1-2708, and in part by the National Science Foundation under Grant 2229876. Recommended by Senior Editor T. Oomen. *(Corresponding author: João Hespanha.)*

The authors are with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106 USA (e-mail: hespanha@ece.ucsb.edu; camsari@ece.ucsb.edu).

Digital Object Identifier 10.1109/LCSYS.2024.3411930

this letter is self-contained, details of some of the algebraic derivations are omitted, but can be found in [24].

II. Optimal Control of Koopman Models

Given a discrete-time nonlinear system of the form

$$x_{t+1} = f(x_t, u_t), \quad \forall t \in \mathcal{T},\ x_t \in \mathcal{X},\ u_t \in \mathcal{U}_t, \tag{1}$$

with the time t taking values over $\mathcal{T} := \{1, \dots, T\}$, our goal is to solve a final-state optimal control problem of the form

$$J^* := \min_{u \in \mathcal{U}} J(u), \quad J(u) := g(x_T), \tag{2}$$

where $u := (u_1, \dots, u_T) \in \mathcal{U} := \mathcal{U}_1 \times \cdots \times \mathcal{U}_T$ denotes the control sequence to be optimized, which we assume finite but potentially with a large number of elements.

For each input $u \in \mathcal{U}_t$, $t \in \mathcal{T}$, the Koopman operator K_u for the system (1) operates on the linear space of functions $\mathcal{F}$ from $\mathcal{X}$ to $\mathbb{R}$ and is defined by

$$\varphi(\cdot) \in \mathcal{F} \mapsto \varphi(f(\cdot, u)) \in \mathcal{F}.$$

Assuming there is a finite dimensional linear subspace $\mathcal{F}_{\text{inv}}$ of $\mathcal{F}$ that is invariant for every Koopman operator in the family $\{K_u : \forall u \in \mathcal{U}_t,\ t \in \mathcal{T}\}$, there is an associated family of matrices $\{A(u) \in \mathbb{R}^{n_\psi \times n_\psi} : \forall u \in \mathcal{U}_t,\ t \in \mathcal{T}\}$ such that

$$\psi_{t+1} = A(u_t)\psi_t, \quad \forall t \in \mathcal{T},\ \psi_t \in \mathbb{R}^{n_\psi},\ u_t \in \mathcal{U}_t, \tag{3}$$

where $\psi_t := \left[\varphi_1(x_t) \cdots \varphi_{n_\psi}(x_t)\right]' \in \mathbb{R}^{n_\psi}$ and the functions $\{\varphi_1(\cdot), \dots, \varphi_{n_\psi}(\cdot)\}$ form a basis for $\mathcal{F}_{\text{inv}}$ [9], [25], [26]. If in addition, the function $g(\cdot)$ in (2) also belongs to $\mathcal{F}_{\text{inv}}$, there also exists a row vector $c \in \mathbb{R}^{1 \times n_\psi}$ such that

$$g(x_T) = c\psi_T, \tag{4}$$

which enables re-writing the optimization criterion (2) as

$$J(u) := c\psi_T = cA(u_T) \cdots A(u_1)x_1. \tag{5}$$

We can thus view the original optimal control problem for the nonlinear system (1) as a switched linear control problem [9]. Note that it is always possible to make sure that (4) hold for some vector c by including $g(x_t)$ as one of the entries of ψ_t.

It is generally not possible to find a finite-dimensional subspace $\mathcal{F}_{\text{inv}}$ that contains $g(\cdot)$ and is invariant for every $\{K_u : u \in \mathcal{U}_t,\ t \in \mathcal{T}\}$. Instead, we typically work with a finite dimensional space for which (3)–(4) hold up to some error. However, to make this error small, we typically need to work with high-dimensional subspaces, i.e., large n_ψ.

III. MCMC Methods for Optimization

To solve a combinatorial minimization of the form

$$J^* := \min_{u \in \mathcal{U}} J(u). \tag{6}$$

over a finite set $\mathcal{U}$, it is convenient to consider the following *Boltzmann distribution* with energies $J(u)$, $u \in \mathcal{U}$:

$$p(u; \beta) := \frac{e^{-\beta J(u)}}{Q(\beta)},\ \forall u \in \mathcal{U}, Q(\beta) := \sum_{\bar{u} \in \mathcal{U}} e^{-\beta J(\bar{u})}, \tag{7}$$

for some constant $\beta \geq 0$. For consistency with statistical mechanics, we say that values of β close to zero correspond to high temperatures, whereas large values correspond to low temperatures. The normalization function $Q(\beta)$ is called the *canonical partition function*.

For $\beta = 0$ (high temperature), the Boltzmann distribution becomes uniform and all $u \in \mathcal{U}$ are equally probable. However, as we increase β (lower the temperature) all probability mass is concentrated on the subset of $\mathcal{U}$ that minimizes the energy $J(u)$. This motivates a procedure to solve (6): draw samples from a random variable $\boldsymbol{u}_\beta$ with Boltzmann distribution given by (7) with β sufficiently high (temperature sufficiently low) so that all samples correspond to states with the minimum energy/cost.

A. Markov Chain Monte Carlo Sampling

Consider a *discrete-time Markov chain* $\{\boldsymbol{u}[1], \boldsymbol{u}[2], \dots\}$ on a finite set $\mathcal{U}$, with *transition probability*

$$p(u'|u) = \mathrm{P}(\boldsymbol{u}[k+1] = u' \,|\, \boldsymbol{u}[k] = u),\ \forall u', u \in \mathcal{U},\ k \geq 1.$$

Combining the probabilities of all possible realizations of $\boldsymbol{u}[k]$ in a row vector $p[k]$ and organizing the values of the transition probabilities $p(u'|u)$ as a *transition matrix* P, with one $u' \in \mathcal{U}$ per column and one $u \in \mathcal{U}$ per row, enables us to express the evolution of the $p[k]$ as

$$p[k+K] = p[k]P^K, \quad \forall k, K \geq 1.$$

A Markov chain is called *regular* if that there exists an integer N such that all entries of P^N are strictly positive. In essence, this means that any $\bar{u} \in \mathcal{U}$ can be reached in N steps from any other $\tilde{u} \in \mathcal{U}$ through a sequence of transitions with positive probability $p(u'|u) > 0$, $u, u' \in \mathcal{U}$. The following result adapted from [27, Ch. IV] is the key property behind MCMC sampling:

Theorem 1 (Fundamental Theorem of Markov Chains): For every regular Markov chain on a finite set $\mathcal{U}$ and transition matrix P, there exists a vector π such that:

1) The Markov chain is *geometrically ergodic*, meaning that there exists constants $c > 0$, $\lambda \in [0, 1)$ such that

$$\|p[k]P^K - \pi\| \leq c\lambda^K, \quad \forall k, K \geq 0. \tag{8}$$

2) The vector π is called the *invariant distribution* and is the unique solution to the *global balance equation*:

$$\pi P = \pi,\ \pi \mathbf{1} = 1,\ \pi \geq 0. \tag{9}$$

To use an MCMC method to draw samples from a desired distribution $p(u; \beta)$, we construct a discrete-time Markov Chain that is regular and with an invariant distribution π that matches $p(u; \beta)$. We then "solve" our sampling problem by only accepting a subsequence of samples $\boldsymbol{u}[K+1], \boldsymbol{u}[2K+1], \dots$ with K "sufficiently large" so that the distribution $p[K+1] = p[1]P^K$ of the sample $\boldsymbol{u}[K+1]$ satisfies

$$\|p[1]P^K - \pi\| \leq \epsilon \overset{(8)}{\Longleftarrow} K \geq \frac{\log c + \log \epsilon^{-1}}{\log \lambda^{-1}}. \tag{10}$$

for some "sufficiently small" ϵ. Such K guarantees that the samples $\boldsymbol{u}[K+1], \boldsymbol{u}[2K+1], \dots$ may not quite have the desired distribution, but are away from it by no more than ϵ. Since ϵ^{-1} appears in (10) "inside" a logarithm, we can get

Algorithm 1 Gibbs Sampling

- Pick arbitrary $\boldsymbol{u}[1] \in \mathcal{U}$.
- Set $k = 1$ and repeat until enough samples are collected:
- *Variable sweep*: For each $t \in \mathcal{T}$:
  - Sample $\boldsymbol{u}_t[k+1]$ with the desired conditional distribution of $\boldsymbol{u}_t$, given $\boldsymbol{u}_{-t}[k]$:

$$\frac{p(u_t, \boldsymbol{u}_{-t}[k]; \beta)}{\sum_{\bar{u}_t} p(\bar{u}_t, \boldsymbol{u}_{-t}[k]; \beta)}, \tag{12}$$

  - Set $\boldsymbol{u}_{-t}[k+1] = \boldsymbol{u}_{-t}[k]$ and increment k.

ϵ extremely small without having to increase K very much. The factor $1/\log \lambda^{-1}$ in (10) is often called the *mixing time* of the Markov chain and is a quantity that should be small to minimize the number of "wasted samples."

To make sure that the chain's invariant distribution π matches a desired sampling distribution $p(u; \beta)$, $u \in \mathcal{U}$, we need the latter to satisfy the *global balance equation* (9), which can be re-written in non-matrix form as

$$p(u'; \beta) = \sum_{u \in \mathcal{U}} p(u'|u) p(u; \beta), \quad \forall u' \in \mathcal{U}.$$

A sufficient (but not necessary) condition for global balance is *detailed balance* [28], which instead asks for

$$p(u|u') p(u'; \beta) = p(u'|u) p(u; \beta), \quad \forall u', u \in \mathcal{U}. \tag{11}$$

B. Gibbs Sampling

Gibbs sampling starts with a function $p(u; \beta)$, $u \in \mathcal{U}$ that defines a desired multi-variable joint distribution up to a normalization constant. We use the subscript notation u_t to refer to the variable $t \in \mathcal{T}$ in u and u_{-t} to refer to the remaining variables. The algorithm operates as in Algorithm 1.

The Gibbs transition probability for a sample corresponding to the update of each variable $\boldsymbol{u}_t[k]$, $t \in \mathcal{T}$ satisfies the detailed balance equation (11) for the desired distribution $p(u; \beta)$ [28], which means that we also have global balance. Detailed balance is not preserved through the full variable sweep, but global balance is. The condition

$$p(u; \beta) > 0, \quad \forall u \in \mathcal{U}, \tag{13}$$

guarantees that for every two possible values of the Markov chain's state, there is one path of nonzero probability that takes the state from one value to the other over a full variable sweep (essentially by changing one variable at a time). This means that the Markov chain generated by Gibbs sampling is regular over the T steps of a variable sweep.

C. Parallel Tempering

Tempering decreases the mixing time of a Markov chain by creating high-probability "shortcuts" between states. It is applicable whenever we can embed the desired distribution into a family of distributions parameterized by a parameter $\beta \in [\beta_{\min}, \beta_{\max}]$, with the property that we have slow mixing for the desired distribution, which corresponds to $\beta = \beta_{\max}$, but we have fast mixing for the distribution corresponding to

Algorithm 2 Tempering

1) Pick arbitrary $\boldsymbol{u}[1] = (\boldsymbol{u}^\beta[1] \in \mathcal{U} : \beta \in \mathcal{B})$.
2) Set $k = 1$ and repeat until enough samples are collected:
   a) For each $\beta \in \mathcal{B}$, sample $\boldsymbol{u}^\beta[k+1]$ with probability $p(u'|\boldsymbol{u}^\beta[k]; \beta)$ and increment k.
   b) *Tempering sweep*: For each $j \in \{1, \ldots, M-1\}$:
   - Compute the flip probability

$$p_{\text{flip}} = f_{\text{flip}}\left(\boldsymbol{u}^{\beta_j}[k], \boldsymbol{u}^{\beta_{j+1}}[k]\right) \tag{16}$$

   and set

$$\boldsymbol{u}[k+1] = \begin{cases} \tilde{\boldsymbol{u}}[k] \text{ with prob. } p_{\text{flip}}, \\ \boldsymbol{u}[k] \text{ with prob. } 1 - p_{\text{flip}}, \end{cases}$$

   where $\tilde{\boldsymbol{u}}[k]$ is a version of $\boldsymbol{u}[k]$ with the entries $\boldsymbol{u}^{\beta_j}[k]$ and $\boldsymbol{u}^{\beta_{j+1}}[k]$ flipped; and increment k.

$\beta = \beta_{\min}$; which is typical for the Boltzmann distribution (7). The key idea behind tempering is then to select M values

$$\mathcal{B} := \{\beta_1 := \beta_{\min} < \beta_2 < \beta_3 < \cdots < \beta_M := \beta_{\max}\}$$

and generate samples from the joint distribution

$$p(u^\beta : \beta \in \mathcal{B}) := \prod_{\beta \in \mathcal{B}} p(u^\beta; \beta) \tag{14}$$

that corresponds to M independent random variables $\boldsymbol{u}^\beta$, one for each parameter value $\beta \in \mathcal{B}$. We group these variables as an M-tuple and denote the joint Markov chain by

$$\boldsymbol{u}[k] := (\boldsymbol{u}^\beta[k] : \beta \in \mathcal{B}),$$

Eventually, from each M-tuple we only use the samples $\boldsymbol{u}^\beta[k]$, $\beta = \beta_{\max}$ that correspond to the desired distribution.

1) General Algorithm: Tempering can be applied to any MCMC method associated with a regular Markov chain with transition probabilities $p(u'|u; \beta)$ and strictly positive transition matrices P_β, $\beta \in \mathcal{B}$. The algorithm uses a *flip function* defined by

$$f_{\text{flip}}(u^{\beta_j}, u^{\beta_{j+1}}) = \min\left\{\frac{p(u^{\beta_{j+1}}; \beta_j)}{p(u^{\beta_{j+1}}; \beta_{j+1})} \frac{p(u^{\beta_j}; \beta_{j+1})}{p(u^{\beta_j}; \beta_j)}, 1\right\}, \tag{15}$$

and operates as in Algorithm 2.

Step 2a corresponds to one step of a base MCMC algorithm (e.g., Gibbs sampling), for each value of $\beta \in \mathcal{B}$. For the Boltzmann distribution (7), the flip function is given by

$$f_{\text{flip}}(u^{\beta_j}, u^{\beta_{j+1}}) = \min\left\{e^{-(\beta_j - \beta_{j+1})\left(J(u^{\beta_{j+1}}) - J(u^{\beta_j})\right)}, 1\right\},$$

which means that the variables $\boldsymbol{u}^{\beta_j}[k]$ and $\boldsymbol{u}^{\beta_{j+1}}[k]$ are flipped with probability one whenever $J(\boldsymbol{u}^{\beta_j}) < J(\boldsymbol{u}^{\beta_{j+1}})$. Intuitively, the tempering sweep in step 2b quickly brings to $\boldsymbol{u}^{\beta_{\max}}[k]$ low-energy/low-cost samples that may have been "discovered" by other $\boldsymbol{u}^\beta[k]$, $\beta < \beta_{\max}$ with better mixing.

2) Balance: Since the sample extractions in step 2a are independent, the transition probability corresponding to this step is given by

$$p\left(\left(u'^\beta : \beta \in \mathcal{B}\right) | \left(u^\beta : \beta \in \mathcal{B}\right)\right) = \prod_{\beta \in \mathcal{B}} p\left(u'^\beta | u^\beta; \beta\right),$$

which satisfies the global balance equation for the joint distribution in (14). The flip function in (15) guarantees

detailed balance for all $M-1$ steps in 2b and therefore global balance for all the combined steps in 2a and 2b, within a full tempering sweep [24].

3) Regularity and Convergence: Assuming that for each $\beta \in \mathcal{B}$, the Markov chain generated by $p(u'|u;\beta)$ is regular with a strictly positive transition matrix P_β, any possible combination of states $\boldsymbol{u}[k] = (u^\beta[k] : \beta \in \mathcal{B})$ at time k can transition in one time step to any possible combination of states $\boldsymbol{u}[k+1] = (u^\beta[k] : \beta \in \mathcal{B})$ at time $k+1$ at step 2a. This means that this step corresponds to a transition matrix P with strictly positive entries. In contrast, each flip step corresponds to a transition matrix $P_{\text{flip}\,j}$ that is non-negative and right-stochastic, but typically has many zero entries. However, each matrix $P_{\text{flip}\,j}$ cannot have any row that is identically zero (because rows must add up to 1). This suffices to conclude that any product of the form

$$Q = P_{\text{flip}\,1} P_{\text{flip}\,2}, \ldots P_{\text{flip}\,M-1} P$$

must have all entries strictly positive. This transition matrix corresponds to the transition from the start of step 2b in one "tempering sweep" to the start of the same step at the next sweep and defines a regular Markov chain. Theorem 1 thus allow us to conclude that the distribution at the start of step 2b converges to the desired invariant distribution. Since every step satisfies global balance, the invariant distribution is preserved across every step, so the distributions after every step also converges to the invariant distribution.

IV. MCMC for Optimal Control

Our goal is to optimize a criterion of the form (5). In the context of MCMC sampling from the Boltzmann distribution (7), this corresponds to a Gibbs update for the control $\boldsymbol{u}_t^\beta[k+1]$ with a distribution (12) that can be computed using

$$\frac{e^{-\beta J\left(u_t, \boldsymbol{u}_{-t}^\beta[k]\right)}}{\sum_{\bar{u}_t} e^{-\beta J\left(\bar{u}_t, \boldsymbol{u}_{-t}^\beta[k]\right)}} = \frac{e^{-\beta \boldsymbol{c}_t^\beta[k] A(u_t) \boldsymbol{x}_t^\beta[k]}}{\sum_{\bar{u}_t} e^{-\beta e^{-\beta \boldsymbol{c}_t^\beta[k] A(\bar{u}_t) \boldsymbol{x}_t^\beta[k]}}},$$

and a tempering flip probability in (16) computed using

$$p_{\text{flip}} = \min\left\{ e^{-(\beta_j - \beta_{j+1}) c\left(\boldsymbol{x}_T^{\beta_{j+1}}[k]\right) - \boldsymbol{x}_T^{\beta_j}[k])}, 1 \right\},$$

where

$$\boldsymbol{c}_t^\beta[k] \coloneqq cA\left(\boldsymbol{u}_T^\beta[k]\right) \cdots A\left(\boldsymbol{u}_{t+1}^\beta[k]\right), \tag{17}$$

$$\boldsymbol{x}_t^\beta[k] \coloneqq A\left(\boldsymbol{u}_{t-1}^\beta[k]\right) \cdots A\left(\boldsymbol{u}_1^\beta[k]\right) x_1. \tag{18}$$

The following algorithm implements Gibbs sampling with tempering using recursive formulas to evaluate (17)–(18).

Computation Complexity and Parallelization: The bulk of the computation required by Algorithm 3 lies in the computation of the matrix-vector products that appear in (19), (20), and (21), each of these products requiring $O(n_\psi^2)$ floating-point operation for a total of $O(MT(2+|\mathcal{U}|)n_\psi^2)$ operations. The tempering step, does not use any additional vector-matrix multiplications. In contrast, a naif implementation of Gibbs sampling with tempering would require $MT|\mathcal{U}|$ evaluations of the cost function (5), each with computational complexity $O(Tn_\psi^2)$. The reduction in computation complexity by a factor of $T|\mathcal{U}|/(2+|\mathcal{U}|)$ is especially significant when the time horizon is large. The price paid for the computational savings is that we need to store the vectors (19), (21), with memory complexity $O(MTn_\psi)$.

The computations of the matrix-vector products mentioned above are independent across different values of $\beta \in \mathcal{B}$ and can be performed in parallel. This means that tempering across M temperatures can be computationally very cheap if enough computational cores and memory are available.

Algorithm 3 Tempering for Koopman Optimal Control

1) Pick arbitrary $\boldsymbol{u}[1] = (\boldsymbol{u}_t^\beta[1] \in \mathcal{U} : t \in \mathcal{T}, \beta \in \mathcal{B})$.
2) Set $k = 1$ and repeat until enough samples are collected:
   a) *Gibbs sampling*: For each $\beta \in \mathcal{B}$:
   - Set $\boldsymbol{x}_1^\beta[k] = x_1$, $\boldsymbol{c}_T^\beta[k] = c$ and, $\forall t \in \mathcal{T}$,

$$\boldsymbol{c}_t^\beta[k] = \boldsymbol{c}_{t+1}^\beta[k] A\left(\boldsymbol{u}_{t+1}^\beta[k]\right). \tag{19}$$

   - *Variable sweep*: For each $t \in \mathcal{T}$
     - Sample $\boldsymbol{u}_t^\beta[k+1]$ with distribution

$$\frac{e^{-\beta \boldsymbol{c}_t^\beta[k] A(u_t) \boldsymbol{x}_t^\beta[k]}}{\sum_{\bar{u}_t} e^{-\beta e^{-\beta \boldsymbol{c}_t^\beta[k] A(\bar{u}_t) \boldsymbol{x}_t^\beta[k]}}}, \tag{20}$$

     - Set $\boldsymbol{u}_{-t}^\beta[k+1] = \boldsymbol{u}_{-t}^\beta[k]$, update

$$\boldsymbol{x}_{t+1}^\beta[k+1] = A\left(\boldsymbol{u}_t^\beta[k+1]\right) \boldsymbol{x}_t^\beta[k], \tag{21}$$

     and increment k.
   b) *Tempering sweep*: For each $j \in \{1, \ldots, M-1\}$
   - Compute the flip probability

$$p_{\text{flip}} = \min\left\{ e^{-(\beta_j - \beta_{j+1}) c\left(\boldsymbol{x}_T^{\beta_{j+1}}[k]\right) - \boldsymbol{x}_T^{\beta_j}[k])}, 1 \right\},$$

   and set

$$\boldsymbol{u}[k+1] = \begin{cases} \tilde{\boldsymbol{u}}[k] & \text{with prob. } p_{\text{flip}} \\ \boldsymbol{u}[k] & \text{with prob. } 1 - p_{\text{flip}}, \end{cases}$$

   where $\tilde{\boldsymbol{u}}[k]$ is a version of $\boldsymbol{u}[k]$ with the entries $\boldsymbol{u}^{\beta_j}[k]$ and $\boldsymbol{u}^{\beta_{j+1}}[k]$ flipped; and increment k.

V. Numerical Example

We tested the algorithm proposed in this letter on a Koopman model for the Atari 2600 *Assault* video game. The goal of the game is to protect earth from small attack vessels deployed by an alien mothership. The mother ship and attack vessels shoot at the player's ship and the player uses a joystick to dodge the incoming fire and fire back at the alien ships. The player's ship is destroyed either if it is hit by enemy fire or if it "overheats" by shooting at the aliens. The player earns points by destroying enemy ships.

We used a Koopman model with $|\mathcal{U}| = 4$ control action, which correspond to "move left", "move right", "shoot up", and do nothing. The optimization minimizes the cost

$$J(u) \coloneqq \begin{cases} -\frac{1}{T}\sum_{t=1}^T r_t & \text{if player's ship not destroyed} \\ +10 - \frac{1}{T}\sum_{t=1}^T r_t & \text{if player's ship destroyed} \end{cases}$$

where r_t denotes the points earned by the player at time t. This cost balances the tradeoff between taking some risk

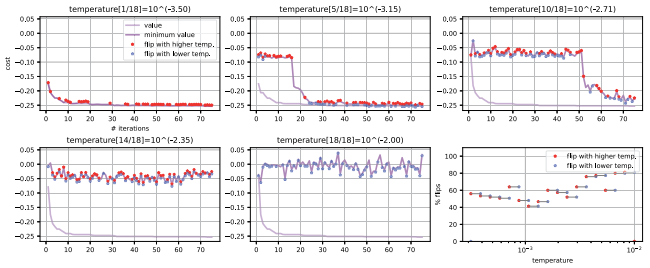

Fig. 1. Typical run of Algorithm 3: The first 5 plots show the cost (y-axis) at the end of each iteration (x-axis), as well as the minimum cost found so far, for a specific temperature β. For this run, 12 logarithmic scaled temperatures were used, but only 5 are shown here. In each plot, red and blue dots mark iterations where flips occurred during the tempering sweep. The bottom-right plot shows the total number of flips across "adjacent" temperatures (in the x-axis), as a percentage of the total number of iterations (in the y-axis).

to collect points to decrease $-\frac{1}{T}\sum_{t=1}^{T} r_t$, while not getting destroyed and incurring the $+10$ penalty. For game play, this optimization is solved at every time step with a receding horizon starting at the current time t and ending at time $t+T$, with only the first control action executed. However, because the focus of this letter is on the solution to (1)–(2), we present results for a single optimization starting from a typical initial condition. A time horizon $T = 40$ was used in this section, which corresponds to a total number of control options $|\mathcal{U}|^T \approx 1.2 \times 10^{24}$.

The state of the system is built directly from screen pixel information. Specifically, the pixels are segmented into 5 categories corresponding to the player's own ship, the player's horizontal fire, the player's vertical fire, the attacker's ships and fire, and the temperature bar. For the player's own ship and the attacker's ships/fire, we consider pixel information from the current and last screenshot, so that we have "velocity" information. The pixels of each category are used to construct "spatial densities" using the entity-based approach described in [9], with observables of the form

$$\varphi_{\ell j}(x) := \sum_i e^{-\lambda \|p_{\ell i} - c_{\ell j}\|^2}, \tag{22}$$

where the index ℓ ranges over the 5 categories above, the summation is taken over the pixels $p_{\ell i}$ associated with category ℓ, and the $c_{\ell j}$ are fixed points in the screen. The densities associated with the 5 categories are represented by 50, 50, 100, 200, 16 points $c_{\ell j}$, respectively. The observables in (22), together with the value of the optimization criterion, are used to form a lifted state ψ_t with dimension $n_\psi = 667$. The matrices $A(u)$ in (5) were estimated using 500 game traces using random inputs. Collecting data from the Atari simulator and lifting the state took about 1h45min, whereas solving the least-squares problems needed to obtain the Koopman matrices took less than 1 sec.

Figure 1 depicts a typical run of Algorithm 3, showing flipping of samples across adjacent temperatures in 40-80% of the tempering sweeps. In the remainder of this section we compare the performance of Algorithm 3 with several alternatives. All run times refer to Julia implementations on a 2018 MacBook Pro with a 2.6GHz Intel Core i7 CPU.

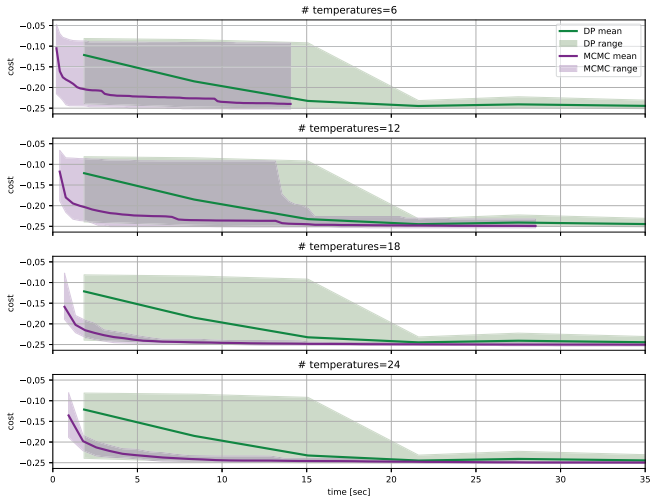

Fig. 2. Run-time comparison between Algorithm 3 (purple) and the dynamic programming algorithm in [9] (green). The x-axis denotes run-time and the y-axis the cost achieved, with the solid lines showing the average cost across 15 different runs and the shaded areas the whole range of costs obtained over those runs. The different plots correspond to different values for the number of temperatures M.

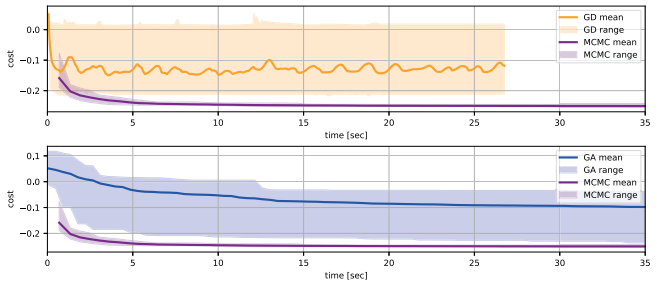

Fig. 3. Run-time comparison between Algorithm 3 with 18 temperatures (purple) and a gradient descent algorithm (top, orange) and a genetic algorithm (bottom, blue). The meaning of the solid lines and shaded areas is the same as in Figure 2.

Figure 2 compares Algorithm 3 with the algorithm in [9], which exploits the piecewise-linear structure of the cost-to-go to efficiently represent and evaluate the value function and also to dynamically prune the search tree. Due to the need for exploration, this algorithm "protects" from pruning a random fraction of tree-branches (see [9] for details). Both algorithms used 6 CPU cores. For Algorithm 3, each core executed one sweep of the tempering algorithm and for the algorithm in [9], the 6 cores were used by BLAS to speedup matrix multiplication. Both algorithms were executed multiple times and the plots show the costs obtained as a function of run time. For Algorithm 3, the run time is controlled by the number of samples drawn. For the algorithm in [9], the run time is controlled by the number of vectors used to represent the value function. Both algorithms eventually discover comparable "optimal" solutions, but Algorithm 3 consistently finds a lower cost faster.

The top plot in Figure 3 compares Algorithm 3 with a gradient descent solver that minimizes the following continuous relaxation of the cost (5):

$$J_{\text{relax}}(\mu) := c\psi_T,\ \psi_{t+1} = \Big(\sum_{u\in\mathcal{U}} \mu_u(t) A(u)\Big)\psi_t \tag{23}$$

where the optimization variables $\mu_u(t) \in [0, 1]$ are required to satisfy $\sum_{u\in\mathcal{U}} \mu_u(t) = 1$, $\forall t \in \mathcal{T}$. The global minimum J^*_{relax} of (23) would match that of (5), if we forced the $\mu_u(t)$ to take binary values in $\{0, 1\}$. Otherwise, J^*_{relax} provides a lower bound for (6). We minimized (23) using the toolbox [29]. The results shown were obtained using Nesterov-accelerated adaptive moment estimation [29], with $\eta = 0.5$, which resulted in the best performance among the algorithms supported by [29]. The constraints on $\mu_u(t)$ were enforced by minimum-distance projection into the constraint set. In general, gradient descent converges quickly, but to a local minima of (23) with cost higher then the minimum found by Algorithm 3 for (6); in spite of the fact that the global minima of (23) is potentially smaller than that of (6).

The bottom plot in Figure 3 compares Algorithm 3 with a genetic optimization algorithm that also resorts to stochastic exploration. This type of algorithm simulates a "population" of candidate solutions to the optimization (6), which evolves by mutation, crossover, and selection. We minimized (23) using the toolbox [30]. The results in Figure 3 used a population size $P = 50$, selection based on uniform ranking (which selects the best $\mu = 2$ individuals with probability $1/\mu$), binary crossover (which combines the solution of the two parent with a single crossover point), a mutation rate of 10%, and a probability of mutation for each "gene" of 5%. These parameters resulted in the lowest costs we could achieve among the options provided by [30], but still significantly higher than the costs obtained with Algorithm 3.

VI. Outlook

We showed that Koopman-based representations of optimal control problems can be efficiently solved for large lifted state-spaces, even when the inputs do not enter linearly. Among the important open issues in this area, we highlight the need to quantify the impact of errors arising from observables that do not span Koopman-invariant subspaces, with notable progress reported in [26], [31]. An additional issue relates to the use of continuous observables, as the ones used in (22) in our example, which are known to introduce fundamental limitations in what they can represent [32], [33].

Acknowledgment

The authors thanks the anonymous reviewers for multiple suggestions that greatly improved the quality of this letter.

References

[1] B. O. Koopman, "Hamiltonian systems and transformation in Hilbert space," *Proc. Nat. Acad. Sci.*, vol. 17, no. 5, pp. 315–318, 1931.
[2] S. Brunton, B. Brunton, J. Proctor, and J. N. Kutz, "Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control," *PloS ONE*, vol. 11, no. 2, 2016, Art. no. e0150171.
[3] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160, Jul. 2018.
[4] J. Proctor, S. Brunton, and J. N. Kutz, "Generalizing Koopman theory to allow for inputs and control," *SIAM J. Appl. Dyn. Syst.*, vol. 17, no. 1, pp. 909–930, 2018.
[5] A. Mauroy, Y. Susuki, and I. Mezić, *Koopman Operator in Systems and Control*. Cham, Switzerland: Springer, 2020.
[6] P. Bevanda, S. Sosnowski, and S. Hirche, "Koopman operator dynamical models: Learning, analysis and control," *Annu. Revi. Control*, vol. 52, pp. 197–212, Jan. 2021.
[7] S. E. Otto and C. W. Rowley, "Koopman operators for estimation and control of dynamical systems," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 4, pp. 59–87, May 2021.
[8] I. Mezić, "Spectral properties of dynamical systems, model reduction and decompositions," *Nonlin. Dyn.*, vol. 41, pp. 309–325, Aug. 2005.
[9] M. Blischke and J. P. Hespanha, "Learning switched Koopman models for control of entity-based systems," in *Proc. 62nd IEEE Conf. Decis. Control*, 2023, pp. 6006–6013.
[10] S. C. Bengea and R. A. DeCarlo, "Optimal control of switching systems," *Automatica*, vol. 41, no. 1, pp. 11–27, 2005.
[11] X. Xu and P. Antsaklis, "Optimal control of switched systems based on parameterization of the switching instants," *IEEE Trans. Autom. Control*, vol. 49, no. 1, pp. 2–16, Jan. 2004.
[12] T. R. Mehta and M. Egerstedt, "An optimal control approach to mode generation in hybrid systems," *Nonlin. Anal. Theory, Methods Appl.*, vol. 65, no. 5, pp. 963–983, 2006.
[13] F. Zhu and P. Antsaklis, "Optimal control of switched hybrid systems: A brief survey," Dept. Electr. Eng., Univ. Notre Dame, Notre Dame, IN, USA, Rep. ISIS-2013-007, 2011.
[14] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[15] V. Černỳ, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm," *J. Opt. Theory Appl.*, vol. 45, pp. 41–51, Jan. 1985.
[16] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-6, no. 6, pp. 721–741, Nov. 1984.
[17] R. H. Swendsen and J.-S. Wang, "Replica monte carlo simulation of spin-glasses," *Phys. Rev. Lett.*, vol. 57, no. 21, p. 2607, 1986.
[18] C. J. Geyer, "Markov chain monte carlo maximum likelihood," in *Proc. 23rd Symp. Interface Comput. Sci. Stat.*, 1991, pp. 156–163.
[19] K. Hukushima and K. Nemoto, "Exchange monte carlo method and application to spin glass simulations," *J. Phys. Soc. Jpn.*, vol. 65, no. 6, pp. 1604–1608, 1996.
[20] A. E. Gelfand and A. F. Smith, "Sampling-based approaches to calculating marginal densities," *J. Am. Stat. Assoc.*, vol. 85, no. 410, pp. 398–409, 1990.
[21] D. J. Earl and M. W. Deem, "Parallel tempering: Theory, applications, and new perspectives," *Phys. Chem. Chem. Phys.*, vol. 7, no. 23, pp. 3910–3916, 2005.
[22] N. Mohseni, P. L. McMahon, and T. Byrnes, "Ising machines as hardware solvers of combinatorial optimization problems," *Nat. Rev. Phys.*, vol. 4, no. 6, pp. 363–379, 2022.
[23] N. Aadit et al., "Massively parallel probabilistic computing with sparse Ising machines," *Nat. Electron.*, vol. 5, no. 7, pp. 460–468, 2022.
[24] J. P. Hespanha and K. Çamsarı, "Markov chain monte carlo for Koopman-based optimal control: Technical report," 2024, *arXiv:2405.01788*.
[25] C. Folkestad and J. W. Burdick, "Koopman NMPC: Koopman-based learning and nonlinear model predictive control of control-affine systems," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2021, pp. 7350–7356.
[26] M. Haseli and J. Cortés, "Modeling nonlinear control systems via Koopman control family: Universal forms and subspace invariance proximity," 2024, *arXiv:2307.15368*.
[27] J. G. Kemeny and J. L. Snell, *Finite Markov Chains*. Berlin, Germany: Springer, 1976.
[28] S. Brooks, "Markov chain monte carlo method and its application," *J. Royal Stat. Soc. Ser. D (Stat.)*, vol. 47, no. 1, pp. 69–100, 1998.
[29] "Jacobcvt12/GradDescent.jl." Accessed: Jun. 5, 2024. [Online]. Available: https://github.com/jacobcvt12/GradDescent.jl
[30] "Wildart/Evolutionary.jl." Accessed: Jun. 5, 2024. [Online]. Available: https://github.com/wildart/Evolutionary.jl
[31] F. Nüske, S. Peitz, F. Philipp, M. Schaller, and K. Worthmann, "Finite-data error bounds for Koopman-based prediction and control," *J. Nonlin. Sci.*, vol. 33, no. 1, p. 14, 2023.
[32] C. Bakker, K. E. Nowak, and W. S. Rosenthal, "Learning Koopman operators for systems with isolated critical points," in *Proc. 58th IEEE Conf. Decis. Control*, 2019, pp. 7733–7739.
[33] Z. Liu, N. Ozay, and E. D. Sontag, "Properties of immersions for systems with multiple limit sets with implications to learning Koopman embeddings," 2024, *arXiv:2312.17045*.