Policy-guided Monte Carlo: Reinforcement-learning Markov chain dynamics

Troels Arnfred Bojesen*

Department of Applied Physics, University of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo 113-8656, Japan

(Received 27 August 2018; published 4 December 2018)

We introduce policy-guided Monte Carlo (PGMC), a computational framework using reinforcement learning to improve Markov chain Monte Carlo (MCMC) sampling. The methodology is generally applicable, unbiased, and opens up a path to automated discovery of efficient MCMC samplers. After developing a general theory, we demonstrate some of PGMC's prospects on an Ising model on the kagome lattice, including when the model is in its computationally challenging kagome spin ice regime. Here we show that PGMC is able to automatically machine learn efficient MCMC updates without *a priori* knowledge of the physics at hand.

DOI: 10.1103/PhysRevE.98.063303

I. INTRODUCTION

Since the invention of Markov chain Monte Carlo (MCMC) in the mid 20th century [1,2], MCMC methods have grown to become the backbone not only of computational physics, but arguably of modern civilization as a whole [3]. The basic idea of MCMC is seemingly innocuous: Use a Markov chain to obtain a set of samples $\{s\}$ from a state space $\mathcal{S}$, each sample with a probability weight (or density) given by some function, or model, $w : \mathcal{S} \to \mathbb{R}_{\geqslant 0}$. Unfortunately, however, the autocorrelation of the samples means that long and possibly computationally expensive simulations may be needed in order to achieve reliable results. This reality poses a threat to the feasibility of using MCMC in practice, as real-world computational resources are limited (and comes with a monetary cost).

To combat this problem, a plethora of MCMC algorithms, methods, and paradigms has been developed, most of which attempt to improve the original Metropolis-Hastings framework [1,4] in one way or another [5]. Among those, an intriguing group of algorithms has emerged with the recent influx of machine-learning ideas and techniques to the domain of MCMC: the effective model Monte Carlo (EMMC) method [6,7].[1]

In EMMC, the task of sampling states following model w is divided into two stages: a learning or training stage and an earning or sampling stage. First, one obtains a set of training data, meaning a set of samples following the distribution of w, e.g., by means of a traditional MCMC simulation. Then an effective model $\tilde{w}_{\boldsymbol{\theta}} : \tilde{\mathcal{S}} \to \mathbb{R}_{\geqslant 0}$, $\tilde{\mathcal{S}} \supseteq \mathcal{S}$, parametrized by a set of parameters $\boldsymbol{\theta} = \{\theta_i\}$, is fitted to this data set by some form of regression, i.e., machine learned, such that $\tilde{w}_{\boldsymbol{\theta}}(s) \approx w(s)$ for states of statistical significance. The effective model should be constructed in a way that makes it computationally advantageous over the original model w. It could, for example, be computationally cheaper to evaluate and/or open up for the use of a previously inaccessible sampling method. In the second stage, $\tilde{w}_{\boldsymbol{\theta}}$ is used as a proposal generator: A number of updates $s \to s_1 \to s_2 \to \cdots \to s_{n-1} \to s_n = s'$ is performed following some MCMC sampling scheme applied to the effective model $\tilde{w}_{\boldsymbol{\theta}}$. Then the new state s' is treated as a proposal for the sampling of the original model, in which case it is accepted with the Metropolis acceptance rate (or any other acceptance rule fulfilling the detailed balance condition)

$$\alpha(s \to s') = \min\left[1, \frac{\tilde{w}_{\boldsymbol{\theta}}(s)w(s')}{\tilde{w}_{\boldsymbol{\theta}}(s')w(s)}\right]. \tag{1}$$

This ensures that the detailed balance condition

$$w(s)\pi(s \to s')\alpha(s \to s') = w(s')\pi(s' \to s)\alpha(s' \to s) \tag{2}$$

holds, in which case the Markov chain will asymptotically sample the distribution of w instead of the distribution of $\tilde{w}_{\boldsymbol{\theta}}$. Here $\pi(s \to s')$ denotes the probability of proposing the change $s \to s'$. By choosing a sensible effective model and update rule, an overall faster diffusion in state space (in CPU and/or real time) can be achieved, with obvious beneficial implications.

Although the EMMC scheme has demonstrated significant speedups in various instances [6,7,9–14], it is not without caveats and shortcomings. First, the effective model has to be flexible enough to be able to "imitate" the original model. If this is not the case, EMMC may perform worse than a traditional "naive" MCMC method. Second, the effective model has to be trained using sufficiently good training data, with complex effective models with more parameters requiring more data. Acquiring the training data can in itself be challenging, e.g., if the dynamics of the MCMC sampler used is slow. Should the training data not represent a sufficiently good sampling of the relevant parts of state space, the effective model, and hence the final outcome, will reflect this. In the best case, this only makes the EMMC simulation slower, in the worst, the proposed states will be drawn from a wrong or incomplete subset of state space.

*troels.bojesen@aion.t.u-tokyo.ac.jp

[1] We use the label effective model Monte Carlo instead of, e.g., self-learning Monte Carlo as done in [7], since the latter (i) is a too broad description and (ii) strongly resembles the name of the algorithmically different (and by more than a decade preceding) self-learning kinetic Monte Carlo method [8].

2470-0045/2018/98(6)/063303(16)

Issues concerning the first stage of the EMMC can be somewhat mitigated by iteratively (and gradually) improving the effective model, constructing a cascade of learning runs based on the previously obtained effective models, until convergence has been reached. Such a scheme, however, does not resolve situations where the MCMC algorithm used to generate proposal states based on the effective model is intrinsically inefficient or inappropriate for the problem at hand. For example, an effective-model-based proposal generator that is frozen, in the sense that it never or almost never proposes significant movements in state space, will lead to poor MCMC performance, (almost) regardless of each iteration's evaluation speed. EMMC is only as good as its weakest link: the proposal generation.

Naturally, the latter concerns can sometimes, when the details of the Markov chain dynamics are known *a priori*, be countered by hand crafted updates during the sampling run. Such "human intervention," however, does offset some of the usefulness and automatic discovery potential of having machine-learned algorithms in the first place.

In this work, we take a different approach in the quest for not only finding efficient MCMC algorithms, but also doing so in an as automatic as possible fashion. Instead of focusing on constructing effective models, we argue that it is more fruitful to target the proposal generation directly by tuning the policy for making updates, akin to what is done within the reinforcement-learning branch of machine learning [15]. After all, we are primarily interested in the (efficient) diffusion of Markov chains through state space, not the effective models *per se*.[2]

The idea of automatically tuning the proposal distributions of MCMC samplers is not new: It is not unreasonable to speculate that *ad hoc* solutions may have emerged already shortly after the invention of MCMC.[3] Later attempts at formalizing the process can be found, e.g., in Ref. [16] (and references therein) as well as in the form of the so-called adaptive MCMC algorithms.[4,5] Shared by these is the tuning of some defining parameters of select, typically Gaussian, proposal distributions. In joining forces with reinforcement learning, however, we will show that it is possible to take the idea even further.

In the following section, we will connect a few core concepts of reinforcement learning to MCMC as we prepare the foundation, before we develop in Sec. III a general reinforcement learning MCMC framework, dubbed policy-guided Monte Carlo (PGMC). In Sec. IV, PGMC is demonstrated through a series of simulations of the kagome lattice Ising model in a field. The examples also lead the way to a more detailed discussion of the specifics of the methodology. Finally, we attempt to put PGMC somewhat in perspective in Sec. V, before summarizing in Sec. VI.[6]

II. MCMC IN LIGHT OF REINFORCEMENT LEARNING

The Markov decision processes heavily employed within reinforcement learning may be regarded as a superset of the Markov chains [15]. Thus, much of the formalism developed in reinforcement learning can readily be adopted to MCMC: The update $s \to s'$ may be regarded as an action, one of possibly many taking the system, or agent, from state s to some other state s'. We denote the space of all actions by $\mathcal{A}_s$, where the subscript s, when included, implies that the possible actions may depend on the initial state s. The policy for taking action $a : s \mapsto s', a \in \mathcal{A}_s$, is quantified by the proposal probability $\pi : \mathcal{A}_s \times \mathcal{S} \to [0, 1]$. We will later refer to π itself as the policy. The inverse action is denoted by $a^{-1} : s' \mapsto s$.

In Metropolis-Hastings MCMC, the acceptance probability α can in principle be chosen in any way that satisfies Eq. (2). However, for simplicity, and in order to make the discussion more concrete, we will be using the Metropolis acceptance rate [1,4]

$$\alpha(s \to s') = \min\left[1, \frac{w(s')\pi(s' \to s)}{w(s)\pi(s \to s')}\right]. \tag{3}$$

Then the Markov chain dynamics will be determined solely by the choice of policy π (which implicitly also determines the action space), as the model w is given *a priori*.[7]

In the original Metropolis algorithm, the policy is symmetric, $\pi(s \to s') = \pi(s' \to s)$, rendering Eq. (3) particularly simple. However, although convenient, there is no reason to believe that this choice is optimal. For example, by associating $\pi(s \to s')$ with updates based on $\tilde{w}_{\boldsymbol{\theta}}$, the proposal generation of EMMC may be seen as a (typically) asymmetrical policy. A similar case can be made for many other existing MCMC algorithms.

A natural question then follows: What is the optimal policy for an MCMC simulation of a given model? To address this in a quantitative way, we look at the sampling of some observable O. (After all, this is ultimately the goal of the simulation.) The statistical efficiency of this process is typically gauged by the associated integrated autocorrelation time

[2] Even if the goal is to determine an effective model, we would claim that PGMC sampling is superior to EMMC in obtaining the necessary training data.

[3] After all, Metropolis *et al.* themselves note that, with respect to their tunable parameter α, "[it] may be mentioned in this connection that the maximum displacement α must be chosen with some care; if too large, most moves will be forbidden, and if too small, the configuration will not change enough" [1]. The motivation for finding a way to more or less automatically choose a "decent" α was present right from the start.

[4] See, for instance, Refs. [17,18] and references therein. Examples of algorithms showing some similarities to the algorithms presented in this work can be found in, e.g., Refs. [19,20].

[5] The naming is rather confusing, as the adaptive MCMC algorithms are not Markovian.

[6] Recently, the author became aware that Ref. [21] briefly floats a similar idea of applying reinforcement learning methods to directly search for an optimal policy. The idea however is not pursued in any way in Ref. [21].

[7] The derivations done here can straightforwardly be adopted for other acceptance rates as well. In the case in which these acceptance rates involve tunable parameters, like in Ref. [22], the parameters can be treated similarly to other parameters in the PGMC algorithm (see Sec. III).

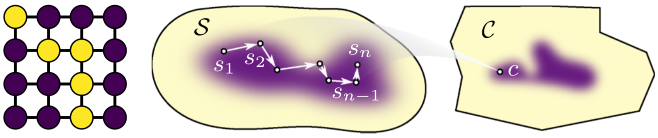

FIG. 1. Sketch of the hierarchy of abstractions in play. A state, e.g. a configuration of a lattice model, constitutes a single point in a high-dimensional state space $\mathcal{S}$. A trajectory of states $s_1 \to s_2 \to \cdots \to s_n$ of length n corresponds to a single point c in the even higher-dimensional trajectory space $\mathcal{C}$. The darker areas illustrate the regions of higher probability density that contributes the most to the final result.

$\tau_O : \mathcal{C} \to [\frac{1}{2}, \infty)$ [23].[8] Here $\mathcal{C}$ denotes the set of all possible trajectories of states the Markov chain can follow through state space, as illustrated in Fig. 1. In this work, τ_O is measured in units of Monte Carlo updates, so to reflect the real-world performance of generating a trajectory, the autocorrelation time should be multiplied by a cost factor $u : \mathcal{C} \to \mathbb{R}_{>0}$ that, ideally, incorporates all costs of importance associated with obtaining a trajectory. The costs of importance could, for example, be the time or computational resources needed.

Equivalently, and more conveniently, we introduce the performance factor

$$\epsilon_O(c) \equiv \frac{1}{2\tau_O(c)u(c)} > 0, \quad c \in \mathcal{C}, \tag{4}$$

as a measure of efficiency; the larger the performance factor, the better the MCMC sampling. If $\epsilon_O(c) = 0$, all samples are perfectly correlated and/or infinitely expensive to generate, while $\epsilon_O(c) = u(c)^{-1}$ means that they are statistically independent. Thus, all other factors being equal, the optimal policy π^* is the one that maximizes the expected performance factor

$$\langle \epsilon_O \rangle_{c\sim w} = \sum_{c\in\mathcal{C}} \epsilon_O(c)p(c). \tag{5}$$

The subscript $c \sim w$ (or sometimes just $\sim w$) signifies that the states of the trajectory c are sampled following the w distribution. This statistics is captured by $p(c)$, the probability of creating c. In the language of reinforcement learning, the performance factor plays the role of a reward function. Denoting by $\pi(c)$ and $\alpha(c)$ the probability of proposing and accepting a trajectory $c = (s_1 \to s_2 \to \cdots \to s_n)$ of some length n, we have

$$\begin{aligned} p(c) &= p(s_1)\pi(c)\alpha(c) \\ &= p(s_1)\prod_{t=1}^{n-1} \pi(s_t \to s_{t+1})\alpha(s_t \to s_{t+1}) \end{aligned} \tag{6}$$

due to the Markov property of the Markov chain. Here $p(s_1)$ is the probability of starting in state s_1, which must fulfill $p(s_1) \propto w(s_1)$ if $c \sim w$ is to hold in Eq. (5). (In practice, this will be the case if the trajectories are obtained from equilibrated MCMC simulations.)

III. POLICY-GUIDED MONTE CARLO

Similar to EMMC, we propose a two-stage PGMC scheme: (1) Given a model w, find a policy π^* that makes Eq. (5) as large as possible and (2) use π^* in an MCMC simulation to obtain samples distributed according to probability weight w. Thus, unlike conventional reinforcement learning, where the policy determines the complete behavior of an agent, in PGMC the policy only guides the behavior of the Markov chain. The final determinant is always the MCMC acceptance step, which enforces the sampling of w to be unbiased. Importantly, this holds regardless of the details of the policy, as long as it is ergodic.

Whereas stage 2 is conceptually straightforward, stage 1 demands a more thorough treatment. In the following sections, we will discuss each of the following central aspects of finding π^*: policy search, how to model the policy, and how to estimate the performance factor.

A. Policy search

Any practical attempt at finding π^* will run into at least two obstacles: (i) The set of all possible policies $\mathcal{P}$ is infinite and (ii) the set of all possible trajectories $\mathcal{C}$ is either infinite or extremely large. Clearly, approximations have to be made in order to proceed. We should therefore not expect to find the optimal policy, but rather just a good one.

To tackle (i), we restrict the policy search to $\{\pi_{\boldsymbol{\theta}}\} \subset \mathcal{P}$ for some policy model $\pi_{\boldsymbol{\theta}}$ parametrized by $\boldsymbol{\theta} \in \mathbb{R}^d$. Demanding $\pi_{\boldsymbol{\theta}}$ to be (almost everywhere) differentiable with respect to $\boldsymbol{\theta}$, we may proceed by policy gradient optimization [15]. At a conceptual level, the procedure is to perform gradient descent on $-\langle \epsilon_O \rangle_{c\sim w}$, now a function of $\boldsymbol{\theta}$, until a minimum is reached. In practice, we will use some variant of stochastic gradient descent to do so [24].

Using stochastic gradient descent also partially solves (ii), as the optimization is now based on a stochastic sampling of $\mathcal{C}$, rather than a practically impossible exact integration over $\mathcal{C}$. The samples, i.e., trajectories, can be generated and stored as a set of training data prior to the optimization (offline optimization), although in reinforcement learning it is typically more advantageous to generated them on the fly, as needed (online optimization). Here we will focus on the latter approach.

Generating even a single trajectory can be expensive if we demand it to be statistically independent of previous trajectories. Fortunately, this is not necessary for the stochastic gradient descent to converge; as long as all regions of $\mathcal{C}$ of importance are eventually visited, it is acceptable with correlations between subsequent samples. This, together with the statistical time translation invariance of the Markov chain in equilibrium, means that time-shifted segments of the same trajectory may be regarded as separate shorter (and strongly correlated) trajectories, or subtrajectories (see Fig. 2). In this sense, a new trajectory is created with every update step. Furthermore, if old subtrajectories are discarded from the training data as new ones are generated (as is the case with online optimization), it is reasonable to expect that a stochastic gradient descent optimization based on a small set of memorized subtrajectories will converge to a correct optimum, even if initial subtrajectories are out of equilibrium.

[8] In this work, we assume that the samples may not be anticorrelated. Nevertheless, even if (mild) anticorrelation emerges, the algorithms and procedures presented are expected to work qualitatively.

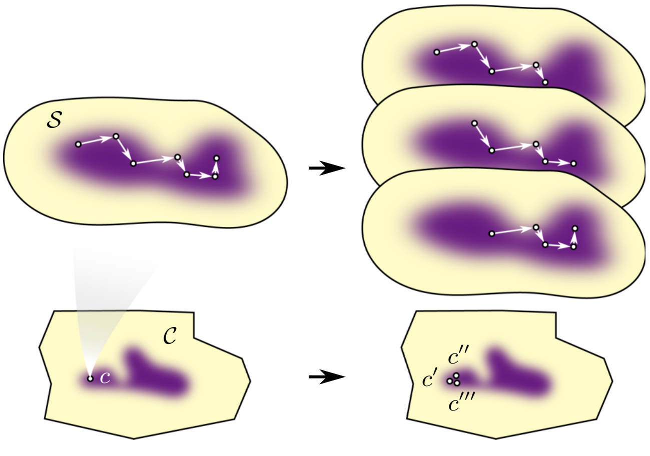

FIG. 2. Trajectory c may be regarded as a number of shorter, time-shifted, and correlated subtrajectories, e.g., c', c'', and c'''.

During the policy optimization, the trajectories are generated following a behavior policy $b \in \mathcal{P}$. The behavior policy may in general be different from the target policy $\pi_{\boldsymbol{\theta}}$ we are trying to optimize. In other words, b controls the update dynamics during the first stage of the PGMC simulation, while the target policy with optimized parameters $\boldsymbol{\theta} = \boldsymbol{\theta}^*$, $\pi_{\boldsymbol{\theta}^*} \approx \pi^*$, is to be used in the MCMC sampling in the second stage. Formally, Eq. (5) in conjunction with Eq. (6) may be written

$$\langle \epsilon_O \rangle_{c\sim w} = \sum_{\substack{s_1 \in \mathcal{S}\\ c\sim b}} \epsilon_O(c) p(s_1) \frac{\pi_{\boldsymbol{\theta}}(c)}{b(c)} \alpha_{\boldsymbol{\theta}}(c) \tag{7}$$

since we are generating trajectories based on b, as indicated by $c \sim b$, but ultimately desire the statistics generated by $\pi_{\boldsymbol{\theta}}$. Like before, each trajectory starts in some state s_1. A subscript $\boldsymbol{\theta}$ has been included in $\alpha_{\boldsymbol{\theta}}$ as a reminder that the acceptance rate also depends on $\boldsymbol{\theta}$ [through $\pi_{\boldsymbol{\theta}}$; see Eq. (3)]. We opt to not include a Metropolis acceptance step in the training stage of the simulation (i.e., all updates are accepted), as we desire as fast an exploration of the state space as possible. Then $\alpha_{\boldsymbol{\theta}}$ should be treated just as a numerical factor during the optimization.

Since the terms of Eq. (7) may span several orders of magnitude, especially in the initial iterations of the optimization, we will attempt to maximize $\ln\langle \epsilon_O \rangle_{c\sim w}$ rather than Eq. (7) directly. This does not change π^*, but reduces the variance of the policy gradient estimate, because

$$\nabla_{\boldsymbol{\theta}} \ln\langle \epsilon_O \rangle_{c\sim w} = \frac{\nabla_{\boldsymbol{\theta}} \langle \epsilon_O \rangle_{c\sim w}}{\langle \epsilon_O \rangle_{c\sim w}} \tag{8}$$

is numerically more stable when $\langle \epsilon_O \rangle_{c\sim w}$ is approximated by a finite number of stochastically sampled terms [15].

Moreover, exploiting that a reversible Markov chain in equilibrium is invariant with respect to time reversal, we may perform the replacement

$$p(c) \to \sqrt{p(\overrightarrow{c})p(\overleftarrow{c})}. \tag{9}$$

The arrows in $\overset{\rightleftarrows}{c} = (s_1 \rightleftarrows s_2 \rightleftarrows \cdots \rightleftarrows s_n)$ indicate whether the trajectory c is traced forward or backward in time. Although not imperative, this symmetrization improves the stability and convergence of the optimization. First, the amount of information taken into account is increased by imposing the time-reversal symmetry. Second, the multiplicative combination of $p(\overrightarrow{c})$ and $p(\overleftarrow{c})$ has a regularizing effect on the optimization, since before equilibrium has been reached, a large $p(\overrightarrow{c})$ will not necessarily be favored if it leads to a small $p(\overleftarrow{c})$. Thus, we expect the symmetrization to be of particular importance in dampening temporary learning of any transient behavior the system might experience before reaching equilibrium.

Using Eqs. (3), (7), and (9), the symmetry $\epsilon_O(\overrightarrow{c}) = \epsilon_O(\overleftarrow{c}) = \epsilon_O(c)$, and the identities $\nabla_{\boldsymbol{\theta}} g_{\boldsymbol{\theta}} = g_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \ln g_{\boldsymbol{\theta}}$ and $\min(0, x) + \min(0, -x) = -|x|$, we get

$$\begin{aligned}
&\nabla_{\boldsymbol{\theta}} \langle \epsilon_O \rangle_{c\sim w}\\
&\quad= \sum_{\substack{s_1 \in \mathcal{S}\\ c\sim b}} p(s_1)\epsilon_O(c) \frac{\pi_{\boldsymbol{\theta}}(c)}{b(c)} \alpha_{\boldsymbol{\theta}}(c) \nabla_{\boldsymbol{\theta}} [\ln \pi_{\boldsymbol{\theta}}(c) + \ln \alpha_{\boldsymbol{\theta}}(c)]\\
&\quad\to \frac{1}{2} \sum_{\substack{s_1 \in \mathcal{S}\\ c\sim b}} \sqrt{p(s_1)p(s_n)}\epsilon_O(c) \sqrt{\frac{\pi_{\boldsymbol{\theta}}(\overrightarrow{c})\pi_{\boldsymbol{\theta}}(\overleftarrow{c})}{b(\overrightarrow{c})b(\overleftarrow{c})}}\\
&\qquad\times \exp\left[-\frac{1}{2}|\Delta f_{\boldsymbol{\theta}}(c)|\right] \nabla_{\boldsymbol{\theta}} [\ln \pi_{\boldsymbol{\theta}}(\overrightarrow{c}) + \ln \pi_{\boldsymbol{\theta}}(\overleftarrow{c}) - |\Delta f_{\boldsymbol{\theta}}(c)|],
\end{aligned} \tag{10}$$

where

$$\begin{aligned}
\Delta f_{\boldsymbol{\theta}}(c) &\equiv \sum_{t=1}^{n-1} [\ln w(s_{t+1}) + \ln \pi_{\boldsymbol{\theta}}(s_{t+1} \to s_t)]\\
&\quad - [\ln w(s_t) + \ln \pi_{\boldsymbol{\theta}}(s_t \to s_{t+1})]\\
&= \ln w(s_n) - \ln w(s_1) + \sum_{t=1}^{n-1} [\ln \pi_{\boldsymbol{\theta}}(s_{t+1} \to s_t)\\
&\quad - \ln \pi_{\boldsymbol{\theta}}(s_t \to s_{t+1})]
\end{aligned} \tag{11}$$

and

$$\nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(\overset{\rightleftarrows}{c}) = \sum_{t=1}^{n-1} \nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(s_t \rightleftarrows s_{t+1}). \tag{12}$$

Equation (10) is the most general policy gradient for PGMC policy optimization. It can in principle be employed using any ergodic behavior policy b (off-policy learning), but unless a good behavior policy can be found, bad or very bad convergence is to be expected [15]. The particular choice of $b = \pi_{\boldsymbol{\theta}}$ (on-policy learning) circumvents this issue, as the behavior policy presumably will be the best available estimate for the optimal policy at any given time. Although such a choice makes b nonstationary, it is reasonable to expect that any undesired effects of this will only be transient as long as $\boldsymbol{\theta}$ converges.

Furthermore, like Eq. (5), Eq. (10) is based on the assumption that $p(s_i) \propto w(s_i)$, $i = 1, n$. If we were to restrict ourselves to the nonsymmetrized form of Eq. (10), $p(s_1) \propto w(s_1)$ could be achieved by drawing the initial states of the trajectories from an equilibrated MCMC simulation. If the symmetrized form is to be used, $p(s_i) \propto w(s_i)$, $i = 1, n$, can only be guaranteed if the entire trajectories are taken from equilibrated MCMC simulations. Both approaches are clearly not desirable, since they run against the aim for an efficient, online policy optimization procedure. Specifically, if a trajectory $c = (s_1 \to s_2 \to \cdots \to s_n)$ is to be treated as a collection of subtrajectories $\{(s_1 \to s_2 \to \cdots \to s_k), (s_2 \to s_3 \to \cdots \to s_{k+1}), \ldots\}$, then $p(s)$ for almost all $s \in c$ will depend on the particular dynamics of b, which is not guaranteed to sample states proportionally to w. Nevertheless, we will ignore this fact and continue to treat the states as if $p(s) \propto w(s)$. The bias introduced by doing so may at first seem severe, but note that a good behavior policy will only generate states close to the desired true distribution of w; as the policy of the on-policy learning becomes better (due to optimization and/or improved modeling), the bias will diminish.

With the above considerations taken into account, Eq. (10) is approximated by

$$\nabla_{\boldsymbol{\theta}} \langle \epsilon_O \rangle_{c\sim w} \approx \frac{1}{2N_c} \sum_{\substack{s_1\sim\pi_{\boldsymbol{\theta}} \\ c\sim\pi_{\boldsymbol{\theta}}}} \epsilon_O(c) \exp\left[-\frac{1}{2}|\Delta f_{\boldsymbol{\theta}}(c)|\right] \times \nabla_{\boldsymbol{\theta}}[\ln \pi_{\boldsymbol{\theta}}(\overrightarrow{c}) + \ln \pi_{\boldsymbol{\theta}}(\overleftarrow{c}) - |\Delta f_{\boldsymbol{\theta}}(c)|], \tag{13}$$

where N_c is the number of trajectories sampled. Equation (13) is the form of the policy gradient we will be using in this work.

B. Modeling the policy

There are very few restrictions on the functional form of a policy. As long as ergodicity and reversibility are preserved, the latter meaning that $\pi(s \to s') > 0 \Leftrightarrow \pi(s' \to s) > 0$, any function $\pi : \mathcal{A} \times S \to [0, 1]$ will do (albeit only a few of those will do well).

Without loss of generality,[9] we may write

$$\pi(a|s) = \frac{\exp[h(a|s)]}{\sum_{a'\in\mathcal{A}_s} \exp[h(a'|s)]} \in (0, 1], \tag{14}$$

where $h(a|s)$ is the preference for taking action a from state s [15]. It is often computationally more convenient to work with the preference than the policy directly. One reason is that $h : \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is not restricted to a finite interval. Another is that the preference, being logarithmic, easily handles a large range of policy values without leading to numerical underflow or overflow.

In practice, the preference will be parametrized by $\boldsymbol{\theta}$. It can be modeled by an arbitrary function approximator, although the PGMC performance will depend strongly on its ability to imitate an efficient Markov chain dynamics for the probability weight model in question (and doing so efficiently): Not only will Eq. (14) have to be evaluated given an action a, actions also have to selected based on it, all while the state s and possibly the action space $\mathcal{A}_s$ are dynamically changing during the simulation. Depending on the model to be simulated, a significant portion of the computer time may be spent on handling Eq. (14).

We will simply use linear feature maps as preferences in this work [15]. Despite these having limited expressive power, they are fast, automatically incorporate translational invariance, scale efficiently, and are easy to deal with in the relatively simple scenarios explored in Sec. IV. Utilizing more sophisticated constructs such as decision trees, neural networks, etc. [25], is left for future works to explore. Examples of more complex policy structures going beyond Eq. (14) are presented in Sec. IV.

C. Performance factor

The final element of the PGMC optimization to be discussed is the performance factor, as defined by Eq. (4). The first factor of the denominator, the integrated autocorrelation time, is formally given by

$$\tau_O = \lim_{\Delta\to\infty} \frac{1}{2} \sum_{\delta=-\Delta}^{\Delta} \rho_O(\delta), \tag{15}$$

$$\rho_O(\delta) \equiv \frac{\langle O_t O_{t+\delta}\rangle - \langle O\rangle^2}{\langle O^2\rangle - \langle O\rangle^2} \tag{16}$$

for a trajectory of samples $O_1 \to O_2 \to \cdots$, where $O_i \equiv O(s_i)$. In practice, the series should be truncated when Δ is a few times larger than τ_O to give a meaningful result, since $\mathrm{Var}(\tau_O) \to \infty$ as $\Delta \to \infty$ [23].

Estimating $\tau_O(c)$ based on Eq. (15) is a viable strategy during the training stage if the training trajectories fed to the stochastic gradient descent optimization are only weakly correlated. If this is not the case, as suggested in Sec. III A, the change in $\tau_O(c)$ between two consecutive subtrajectories will be minuscule and sensitive to noise and the optimization likely unstable. An alternative, heuristic approach is to simply use

$$(2\tilde{\tau}_O)^{-1} \equiv 1 - \rho_O(1), \tag{17}$$

possessing the desired limiting properties of $(2\tilde{\tau}_O)^{-1} = 0$ for perfectly correlated samples and $(2\tilde{\tau}_O)^{-1} = 1$ for perfectly uncorrelated samples.

The takeaway message of Eq. (17) is that a rough estimate for τ_O can be obtained based only on pairs of consecutive samples (O_t, O_{t+1}). Ultimately, this means that it should be viable to perform stochastic gradient descent with subtrajectories merely of length 2. Obviously, one has to be careful when doing so, as such a myopic perspective may disregard important longer-timescale dynamics; additional means may have to be employed to ensure that the policy converges to a reasonable target. When in doubt, the unbiased integrated autocorrelation time given by Eq. (15) will always be the correct measuring rod to compare with. Nevertheless, and crucially from a practical perspective, the challenging problem of maximizing Eq. (5) is now within reach of relatively simple program designs.

Among the many aspects that make up the cost factor u, only the cost associated with the policy is easily controllable and hence of interest here. The specifics are strongly case dependent, but two limiting regimes may be identified. (i) If the cost of evaluating the model dominates a PGMC sampling

[9] Strictly speaking, the functional form of Eq. (14) forces $\pi(a|s) > 0\, \forall a \in \mathcal{A}_s,\ s \in \mathcal{S}$, i.e., $\pi(a|s) = 0$ is now prohibited. In practice, however, there is no numerical difference between $\pi(a|s) = 0$ and $\pi(a|s) \to 0^+$ and the discrepancy can safely be neglected.

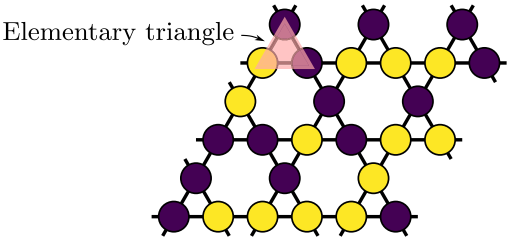

FIG. 3. Configuration of the Ising model on an $L = 3$ kagome lattice. The dark and light disks indicate down (-1) and up $(+1)$ spins, respectively.

iteration, the cost of evaluating the policy, i.e., selecting an action, is of minor importance and may be approximated by a (negligible) constant; in this case, the focus should be on getting the most out of each model evaluation, meaning that one should seek to minimize the integrated autocorrelation time in order to maximize Eq. (4). (ii) If, on the other hand, the cost of evaluating the model is small compared to that of the policy, the cost of the policy is obviously not negligible. In this case, a reasonable approximation would be to make $u(c)$ proportional to the number of operations needed to propose c.

Note that it should be sufficient to select a single, hopefully representative, slow mode (i.e., large autocorrelation time) observable for calculating the performance factor used in finding a good policy. Performing separate simulations for each observable of interest, as opposed to sample all of them during the same simulation, is most likely not an efficient strategy.

IV. PGMC IN PRACTICE

In this section, we will demonstrate how the PGMC framework can be employed in simulating a simple but challenging lattice model. The solutions presented are not necessarily meant to be the best ones, but are selected to gradually expose some of PGMC's potential. It should also be emphasized that PGMC is a general scheme and not restricted to the examples presented here. To underline this, we briefly share some general thoughts on the usage of PGMC in other cases in Sec. IV E. See the Appendix for technical details about the simulations.

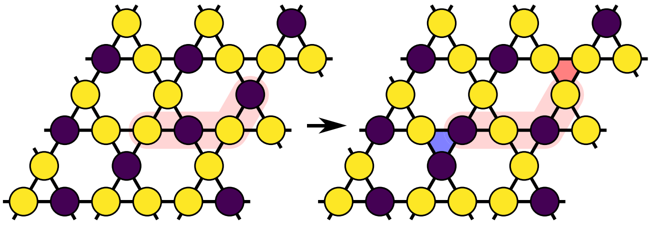

FIG. 4. Generation of two separated monopole excitations from a kagome spin ice ground state by flipping a string of alternating spins. The monopoles are associated with the colored elementary triangles, the only ones that violate the spin ice constraint by not containing two up spins (light) and one down spin (dark).

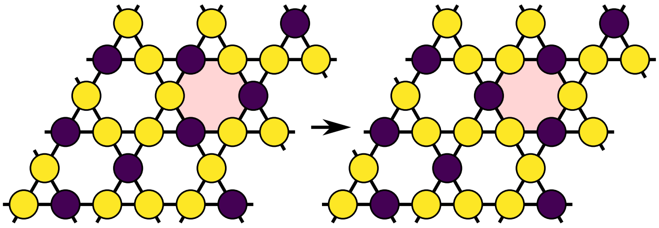

FIG. 5. Tunneling from one kagome spin ice ground state to another. A loop of spins of alternating signs has to be flipped in order to conserve the spin ice rule everywhere. The smallest loop for which this can be accomplished belongs to an elementary hexagon or plaquette.

A. Test model

As a test model, we choose the Ising model on a kagome lattice of size $N = 3 \times L^2$ with periodic boundary conditions (see Fig. 3).

Now the state space is $\mathcal{S} = \{\boldsymbol{\sigma} | \sigma_i \in \pm 1\}$ and the logarithm of the probability weight is given by

$$\ln w(\boldsymbol{\sigma}) = K \sum_{\langle i,j \rangle} \sigma_i \sigma_j - KB \sum_i \sigma_i, \qquad (18)$$

with K a coupling constant and B the strength of an external field. This simple model displays a surprisingly rich physics: In the thermodynamic limit, for $0 < K < K_\mathrm{c}$ and $B = 0$, it is a paramagnet, which at $K_\mathrm{c} = \frac{1}{4}\ln(3 + \sqrt{12}) \approx 0.467$ undergoes a continuous phase transition to a ferromagnetically ordered state [26]. Conversely, when $K < 0$, the model is frustrated. As $K \to -\infty$ and $B = 0$, it does not experience a phase transition, but rather a crossover to an extensively degenerate ground macrostate[10] with a residual entropy of approximately 0.502 per spin [27]. In the ground macrostate, the elementary triangles are constrained to contain either two up and one down spin, or one up and two down spins. The configurations belonging to this manifold of states are connected by a series of single spin flips, each of zero-energy cost. Applying a field of strength $0 < |B| < 4$ partially lifts the degeneracy, since now only one of the elementary triangle configuration categories will minimize the energy. This reduces the residual entropy to approximately 0.108 per spin, and the system enters the kagome spin ice submanifold, so named for its connection to (pyrochlore) spin ice [28]. This macrostate is topologically nontrivial, with particlelike gapped excitations dubbed monopoles [29,30] (see Fig. 4). In order to tunnel from one spin ice configuration to another, a minimum of six spins in a loop has to be flipped, as illustrated in Fig. 5.

The simple model of Eq. (18) provides a rich test bed for PGMC algorithms, whereby the intrinsic dynamics, and hence the nature and difficulty of the simulation, can be tuned by only two parameters. In the following, we will consider the representative set of parameters given in Table I for benchmarking.

[10] We use the term *macrostate*, instead of the more customary *state*, to explicitly distinguish this physical mode from the microstate meaning attached to the word state elsewhere.

TABLE I. Parameters used in the simulations of Eq. (18). The parameters have been chosen so as to realize the various physical (hence also simulation) regimes of the model.

K	B	Description
$K_c \approx 0.467$	0	critical point
0.5	1	ferromagnetic and nonzero field
-4	0	strong frustration and zero field
-4	0.5	strong frustration and weak field
-4	2.5	strong frustration and strong field

B. Single-flip policies

To demonstrate the basics of PGMC, we compare the performance of a few simple but increasingly complex policy models. The policies all have action spaces restricted to single-site spin flips

$$\mathcal{A} = \big\{a_{\sigma_i} | i \in \{1, \dots, N\}\big\}, \tag{19}$$

$$a_{\sigma_i} : \sigma_j \mapsto \begin{cases} -\sigma_j, & j = i \\ \sigma_j, & j \neq i, \end{cases} \qquad a_\sigma^{-1} = a_\sigma, \tag{20}$$

the minimum required for ergodicity to be possible. They are as follows.

(i) The trivial policy π_1, i.e., uniformly random spin selection as given by the preference

$$h_1(a_{\sigma_i}|\boldsymbol{\sigma}) = \frac{1}{N}. \tag{21}$$

This corresponds to the policy of a canonical Metropolis MCMC, with no independent parameters to be determined.

(ii) The local spin orientation policy π_2. This policy is given by

$$h_2(a_{\sigma_i}|\boldsymbol{\sigma}) = \begin{cases} \theta_1 & \text{if } \sigma_i = 1 \\ \theta_2 & \text{if } \sigma_i = -1. \end{cases} \tag{22}$$

With π_2, the probability of flipping a spin may depend on its sign. Although we expect $\theta_1 = \theta_2$ to be the right choice in zero external field, a distinction may be advantageous when the global $\mathbb{Z}_2$ symmetry of Eq. (18) is explicitly broken. The cost is that one independent parameter has to be tuned.

(iii) The local energy sign policy π_3, with preference

$$h_3(a_{\sigma_i}|\boldsymbol{\sigma}) = \begin{cases} \theta_1 & \text{if } E_{\text{local},i} > 0 \\ \theta_2 & \text{if } E_{\text{local},i} \leqslant 0, \end{cases} \tag{23}$$

where $E_{\text{local},i} \equiv \sigma_i(\sum_{\langle j,i\rangle} \sigma_j - B)$, the sum being over the nearest-neighbor sites of i. It is quite typical that the physics of a model is determined by only a few excited degrees of freedom, surrounded by a sea of less relevant, relaxed ones. This means that the trivial policy π_1 may waste most of its action proposals on unimportant (and mostly rejected) update attempts. The policy π_3 is designed to combat this by making a distinction in selecting (locally) excited and nonexcited spins. Compared to π_2, π_3 takes more information into account, rendering it more versatile and potent. This comes at the expense of more complexity. As with π_2, there is now one independent parameter to be determined.

(iv) The local mean field policy π_4, with

$$h_4(a_{\sigma_i}|\boldsymbol{\sigma}) = \theta_{q_i}, \tag{24}$$

where

$$q_i = \frac{1}{2}\left[(\sigma_i + 1) + 2\sum_{\langle j,i\rangle}(\sigma_j + 1)\right] + 1 \tag{25}$$

associates a unique integer $\in \{1, 2, \dots, 2(z+1)\}$ with each possible pair of $(\sigma_i, \sum_{\langle j,i\rangle} \sigma_j)$. For the kagome lattice, the coordination number is $z = 4$, so there are now nine independent parameters to be fixed. The policy π_4 contains all of the other policies, in the sense that π_1, π_2, and π_3 may be considered instances of π_4 with a restricted parameter space. It is therefore guaranteed to perform at least as well as them, given sufficient training.[11]

We have omitted the subscript $\boldsymbol{\theta}$ to avoid clutter in the above expressions. Like the model (18), the policies π_1–π_4 are translationally invariant. However, except trivially for π_1, they do not take advantage of the global $\mathbb{Z}_2$ symmetry when $B = 0$.

The parameters $\boldsymbol{\theta}$ are determined during the learning stage in an on-policy fashion, as outlined in Sec. III A. Setting the autocorrelation observable to

$$O(\boldsymbol{\sigma}) = \boldsymbol{\sigma}, \tag{26}$$

while treating the spin state as a vector with scalar product as multiplication, the performance factor may be approximated by just $\tilde{\epsilon}_O(c) = \text{const}$. This is because all available actions will change the state by exactly the same amount, namely, one spin flip, meaning that Eq. (17) will be constant during the policy search. We also assume that the costs of evaluating the single-flip policies are all similar. Then, if only a single two-state subtrajectory $c_t = (s_t \to s_{t+1})$ is kept in memory, Eq. (8) reduces to

$$\nabla_{\boldsymbol{\theta}}[\ln \pi_{\boldsymbol{\theta}}(\overrightarrow{c}_t) + \ln \pi_{\boldsymbol{\theta}}(\overleftarrow{c}_t) - |\Delta f_{\boldsymbol{\theta}}(c_t)|], \tag{27}$$

which is simply the gradient of the log-likelihood of creating this subtrajectory.

Examples of the policy optimization, as monitored by the mean acceptance rate $\langle\alpha\rangle$ and the normalized effective degrees of freedom[12]

$$d(\pi, s) \equiv \frac{1}{N} \exp\left(-\sum_{a \in \mathcal{A}_s} \pi(a|s) \ln \pi(a|s)\right) \tag{28}$$

are shown in Fig. 6. (Note that in this case, since $\tilde{\epsilon}_O = \text{const}$ and $b = \pi_{\boldsymbol{\theta}}$, $\langle\tilde{\epsilon}_O\rangle \propto \langle\alpha\rangle$. If this does not hold, it is more reasonable to track $\langle\tilde{\epsilon}_O\rangle$ directly.) After a short initial relaxation phase, the policies quickly converge to focus on a

[11] Likewise, no linear combination of the form $\tilde{\pi} = \sum_{x=1}^{4} \omega_x \pi_x$ for some $\omega_x > 0$ can surpass the performance of just π_4; $\tilde{\pi}$ will have the exact same ability to categorize local spin configurations as π_4.

[12] The effective degrees of freedom may also be seen as the exponential of the Shannon entropy [31] of the policy for a given state; it basically measures the amount of information encoded in the policy, with a lower entropy corresponding to less uncertainty of which action will be selected next.

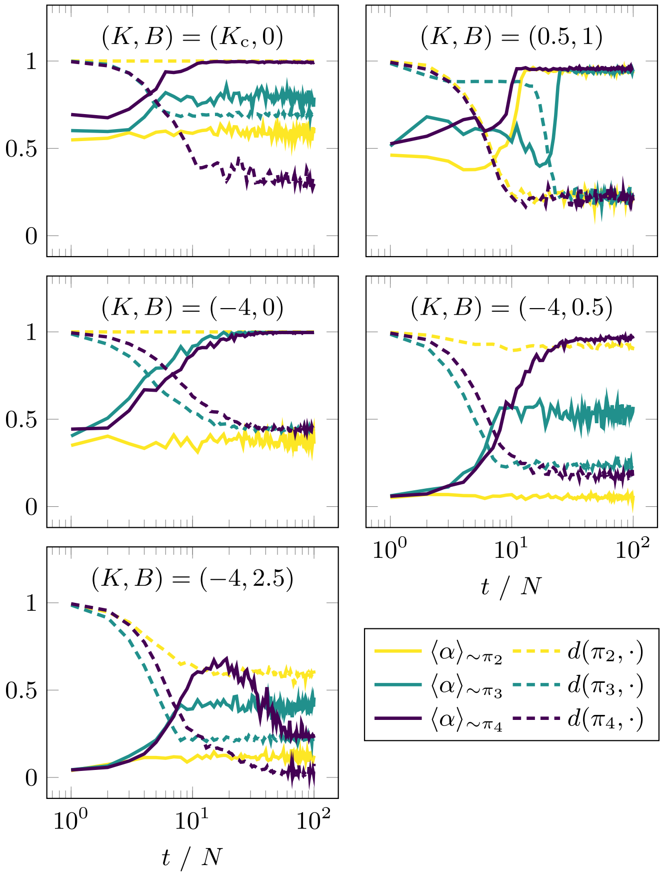

FIG. 6. Learning curves for the nontrivial single-flip policies π_2, π_3, and π_4 [with $\pi_x = \pi_x(t)$] showing the expected acceptance rate $\langle\alpha\rangle$ and normalized effective degrees of freedom d as a function of optimization iterations. The quantities displayed are averages over sweeps of N optimization iterations. Each case was initialized with a random configuration and $\boldsymbol{\theta} = \mathbf{0}$.

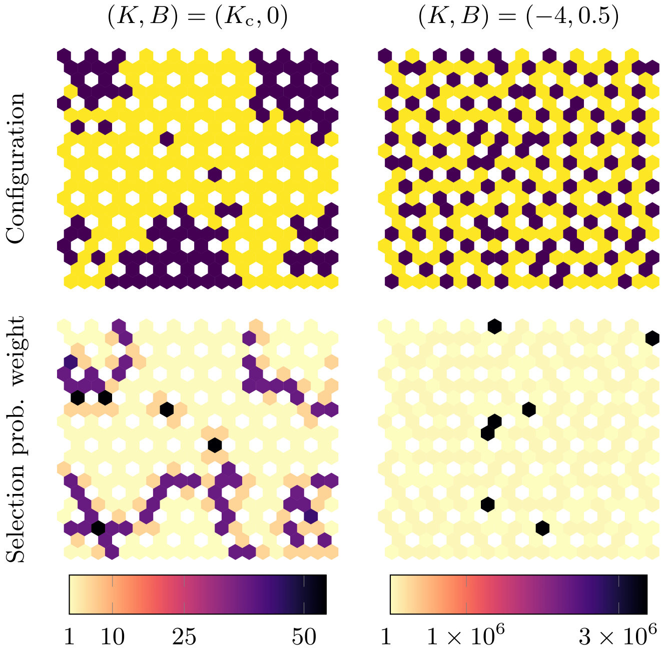

FIG. 7. Snapshots of configurations (top) and the corresponding relative probability weights for selecting the spin to flip in the next iteration (bottom), according to policy π_4. The state on the left is sampled at criticality, while the state on the right is frustrated. The policy, separately optimized to each parameter point, assigns a larger weight to energetic structures like domain walls and monopole excitations, and relatively more so in the case of the frustrated state close to the ground macrostate. The connectivity lines of Fig. 3 and 5 have been omitted and only the values associated with the sites (the small hexagons) are shown.

subset of more relevant degrees of freedom, with π_4 focusing more than π_3, and π_3 more than π_2. Concurrently, this leads to a bias towards sampling physically relevant states, which in turn makes a more refined tuning of the parameters possible. The overall effect is to increase the acceptance rate and decrease the effective number of degrees of freedom, with the additional benefit of improving the equilibration of the system. As expected, π_2 is not able to learn anything from the $B = 0$, nonsymmetry broken cases. The increase in number of parameters from π_2 and π_3 to π_4 means that the convergence typically (but not always) is slower for the latter: The number of iterations required increases because finding a convergent point in a higher-dimensional parameter space requires more information. On the other hand, an improved knowledge about the dynamics may also help with a quicker relaxation of the system, somewhat countering, and in the case of $(K, B) = (0.5, 1)$, exceeding, this effect. The peculiar nonmonotonic behavior of $\langle\alpha\rangle_{\sim\pi_4}$ for $(K, B) = (-4, 2.5)$ most likely relates to the challenging state space in this regime: At this point in (K, B) space, the model is close to the kagome spin ice ground macrostate. The remaining energy above the ground macrostate is associated with localized monopoles and can only be released by their mutual annihilation. Hence, while the monopoles diffuse around in search of an annihilation partner, π_4 is able to temporarily learn the "incorrect" dynamics of these excitations, before the ground macrostate is finally found and the parameters are adjusted accordingly. On the other hand, π_2 and π_3 do not have the capacity to model the physical macrostate properly, and therefore converge to a less restricted subset of state space. The policies ability to automatically learn to focus on important degrees of freedom may also be visualized directly, as is done in Fig. 7.

After convergence, the learned policies are used in the second stage of PGMC, the actual MCMC simulation. Their relative performances are evaluated by comparing estimates of $\langle\epsilon_O\rangle$, obtained from Eqs. (4), (15), and (26). The cost factors are set to be equal to the number of elementary actions, i.e., single spin flips, involved in a PGMC step, which in these cases equals one. The results are summarized in Fig. 8, where the measured acceptance rates are also plotted. The trends of the training results repeat here: The performance typically increases, sometimes significantly, by going from π_1 to π_4, with the notable and expected exception of π_2 when $B = 0$. Sampling at $(K, B) = (-4, -2.5)$ is by far the most challenging task, for which all the single-flip policies, except maybe π_4, fail. Tracking the performance at this parameter point will therefore be of great interest later on. The seemingly low yield at the critical point is a consequence of measuring the performance using the observable of Eq. (26), which does

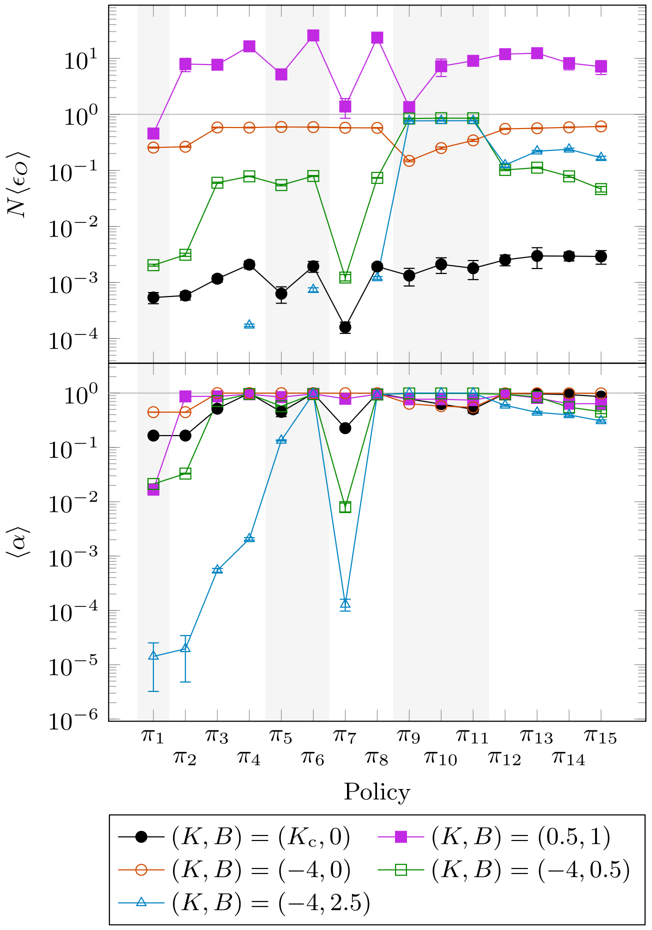

FIG. 8. PGMC sampling performance of Eq. (18) for the parameters listed in Table I and all policies investigated in this work, after being optimized. The top plot shows the estimated performance factor (4) calculated from Eq. (26) and (15) and the cost factor $u(c)$, which is equal to the number of elementary actions used in constructing c. The performance factor has been scaled by the number of sites N to make the timescale correspond to a Monte Carlo sweep. The bottom plot shows the acceptance rate. The horizontal lines indicate the optimal performance, i.e., what the results would have been had the configurations been drawn directly following Eq. (18) instead of being generated by Markov chains. The background shading indicate classes of policies sharing a similar structure. Results obtained at the same (K, B) point share a common color and symbol style, with lines as guides to the eye. Results in unfrustrated parameter regimes are shown with closed symbols, while the frustrated ones are shown with open symbols. In some cases, for π_1–π_3, π_5, and π_7, finding $\langle\epsilon_O\rangle$ for $(K, B) = (-4, 2.5)$ was too slow or unreliable, so these points have been omitted. Error bars indicate sample standard deviation. See the main text and the Appendix for discussion and further details.

not take the global $\mathbb{Z}_2$ symmetry into account; the autocorrelation length is therefore determined by the long timescale of reversing the global polarization. Another interesting manifestation of the focusing ability of the (nontrivial) policies appears at the ferromagnetic and polarized $(K, B) = (0.5, 1)$. As $N\langle\epsilon_O\rangle > 1$ in this case, the PGMC simulations are able to outperform a hypothetical perfect sampling of drawing directly from the desired probability distribution. The reason is straightforward: As the physically relevant states are close to being completely polarized, the effective number of degrees of freedom is significantly lower than the actual number of degrees of freedom N. Most spins are left untouched during an update, which corresponds to "updating" them "for free." On the contrary, constructing a configuration from scratch, however efficiently it is done, comes at a cost proportional to the actual number of degrees of freedom. At some point, the former starts outperforming the latter.

For several of the parameter choices, especially when using π_4, the PGMC simulations have close to 100% acceptance rates, i.e., they are almost rejection free. This means that the policies are able to capture almost the entire Markov chain dynamics, similar to what is achieved by algorithms hand crafted to do so (see, for example, Refs. [32–34]). This does not mean that these policies are optimal, in the sense of maximizing $\langle\epsilon_O\rangle$, but rather that the updates proposed by them, however correlated, closely follow the desired probability distribution of Eq. (18).

C. Chain policies

So far, we have encountered just a small fraction of the vast space of possible policies. The single-flip policies π_1–π_4 only (at most) take information about nearest-neighbor spins into account. This is not always sufficient for a satisfying PGMC sampling, as is exemplified by the challenging parameter point of $(K, B) = (-4, 2.5)$.

There are several possible directions to explore in a quest for improving this. One would be to increase the amount of information available for each action selection by taking further neighbors into account. It is easy to imagine that this could help a policy in guiding the Markov chain more precisely and hence improve the sampling performance. However, as a single-spin-flip policy can only change a configuration by one spin flip at a time, such an approach (alone) would still struggle with state space probability barriers of several spin flips.

A more radical way is to enlarge the action space by allowing for more spin flips between the Metropolis acceptance steps. In other words, let an action consist of several elementary actions instead of just one. Doing so increases the entropy associated with selecting an action, making longer jumps in state space possible and freezing of the Markov chain less likely.

We will pursue the latter approach here, by what we call chain policies. A chain policy consists of a number of elementary policies, each successively selecting and applying an elementary action to the most recent (transit) state, as illustrated in Fig. 9. For example, a chain policy π of the two elementary policies π_{I} and π_{II} is given by

$$\pi(s \to s') = \pi_{\mathrm{I}}(s \to s'')\pi_{\mathrm{II}}(s'' \to s'), \tag{29}$$

where the action $a : s \mapsto s'$ is composed of the two elementary actions $a_{\mathrm{I}} : s \mapsto s''$ and $a_{\mathrm{II}} : s'' \mapsto s'$, i.e., $a = a_{\mathrm{II}}a_{\mathrm{I}}$. Using π, the probability weight of the transit state s'' will not be subject to the Metropolis acceptance step (3), opening up the possibility of tunneling probability barriers. The state s'' is still needed in calculating the policy of the inverse process, which simply reads $\pi(s' \to s) = \pi_{\mathrm{I}}(s' \to s'')\pi_{\mathrm{II}}(s'' \to s)$. Note that the elementary policy of going from one (transit)

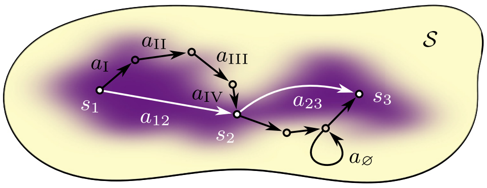

FIG. 9. Example of a three-state trajectory of a chain policy of length 4. Each action, $a_{12} : s_1 \mapsto s_2$ and $a_{23} : s_2 \mapsto s_3$, consists of four elementary actions. The Metropolis acceptance step is only concerned with the states before and after the actions have been completed (circles with white fill), making room for temporary visits of transit states (open circles) with lower probability weights.

state to the next does not need to be the same for the inverse action [compare $\pi_{\mathrm{I}}(s \to s'')$ with $\pi_{\mathrm{II}}(s'' \to s)$].

Now, if a chain policy is to be constructed based on the single-flip policies of the preceding section, the Markov chain will no longer be ergodic: In the above example of a chain policy of length 2, there is no way in which a configuration with an odd number of spins pointing up can be reached from a configuration with an even number of spins pointing up, a consequence of the binary nature of the spins.[13] Therefore, to ensure ergodicity, we are forced to extend the action space of the single-flip policies with a do-nothing action

$$a_\varnothing : s \mapsto s, \quad a_\varnothing^{-1} = a_\varnothing \tag{30}$$

and assign an additional preference weight of $\theta_\varnothing$ for selecting this.[14]

The increased complexity of the action space makes the policy training more complicated as well. The actions may now result in a net $n_{\text{flips}}(s \to s') \in \{0, \ldots, n\}$ spin flips when updating $s \to s'$ with a length n chain policy. (In this regard, a bounce process, a_σ followed by a_σ^{-1}, constitutes zero spin flips if it happens within an update, since $a_\sigma^{-1} a_\sigma = a_\varnothing$.) For simplicity, we model the performance factor for the chain policies as

$$\tilde{\epsilon}_O(s \to s') = n_{\text{flips}}(s \to s'). \tag{31}$$

This should, to leading order, be proportional to the true value for policy chains of moderate lengths, again assuming that the elementary policies all have similar cost factors. Since Eq. (31) is not constant, Eq. (27) can no longer be used. Instead, a minimum of three states $s_{t-2} \to s_{t-1} \to s_t$ has to be memorized such that two subtrajectories of length 2, $c_1 = (s_{t-2} \to s_{t-1})$ and $c_2 = (s_{t-1} \to s_t)$, can be used in approximating $\langle \epsilon_O \rangle$. This is necessary if Eq. (8) is to be calculated without the performance factors in the denominator and numerator canceling.[15]

We first test the following two chain policies of length 2: π_5, with an elementary preference like h_3 [Eq. (23)], with the addition of $a_\varnothing$ to the elementary action space, which increases the number of independent parameters to 2, and π_6, with an elementary preference like h_4 [Eq. (24)], with the addition of $a_\varnothing$ to the elementary action space, which increases the number of independent parameters to 10. For simplicity, we have kept the elementary policies of these chain policies equal (i.e., $\pi_{\mathrm{I}} = \pi_{\mathrm{II}}$).

The performance of PGMC sampling with π_5 and π_6, after training, is also shown in Fig. 8. Some trends are clear: The inclusion of $a_\varnothing$ in the action space means that the Markov chain dynamics may now follow the probability distributions of Eq. (18) much closer in all parameter regimes, as the Metropolis rejections are modeled as well. This leads to a very high acceptance rate, especially for π_6, which has the most flexible elementary policy. Were we to simulate a model that is expensive to evaluate, this feature may be of significance in itself: It is redundant to calculate the probability weight in the Metropolis acceptance step if it is known, in advance, that the state will not change. In the current case, however, such a saving is negligible. For the field-free parameter points $(K, B) = (K_{\mathrm{c}}, 0)$ and $(K, B) = (-4, 0)$, the performance factor estimates do not change much as compared to the results of the corresponding single-flip policies; from the perspective of the Markov chain dynamics, there is little difference in taking one or two elementary action steps at a time. If anything, there seems to be a disadvantage for π_5 at the critical point, as the imperfect tracking of the desired probability distribution by the elementary policy is enhanced when not corrected until after two iterations. The situation is more interesting with stronger fields. There is an additional energy cost of $2B$ associated with flipping a single spin, i.e., there is a probability barrier of exciting a spin against a nonzero field. The reverse operation cannot always counter this, as the gain of $-2B$ may be more than necessary for Metropolis acceptance. However, by combining two flips, one of a spin parallel and one of a spin antiparallel to the field, the overall energy change associated with the field cancels; the barrier has been tunneled, or at least lowered. This effect is readily seen for π_6, with an improvement over π_4 for both $(K, B) = (0.5, 1)$ and $(K, B) = (-4, 2.5)$. Similar gains are expected for π_5 over π_3 as well, but as mentioned above, the benefit seems to be outweighed by the cost of insufficient tracking of the probability distribution.

It is tempting to repeat the exercises of π_5 and π_6 with the chain policies π_7 and π_8 of lengths 6, composed from the same elementary policies, respectively. The motivation for the extended length is to make direct tunneling between kagome ice ground states possible (Sec. IV A). In order to promote such processes, we skew the training performance factor estimate from Eq. (31) to

$$\tilde{\epsilon}_O(s \to s') = \max[0, n_{\text{flips}}(s \to s') - 2], \tag{32}$$

[13] Flipping two spins leads to an overall change of $\pm 1 \pm 1 = 0, \pm 2$ in the number of spins pointing up. Adding an even number to an even (odd) number is an even (odd) number.

[14] Alternatively, we can extend the elementary action space to encompass both single spin flips and multiple spin flips. Such a solution is however significantly more complicated than extending it with a do-nothing action. That said, the main point is that the policy has to be made ergodic. Should the degrees of freedom of a model to be simulated be of a different character, e.g., continuous instead of binary, the elementary policies may not need to be amended when used in a chain policy.

[15] In the case $\tilde{\epsilon}_O(c_1) = \tilde{\epsilon}_O(c_2) = 0$, Eq. (8) can, for instance, be set to zero. This is what has been done in this work.

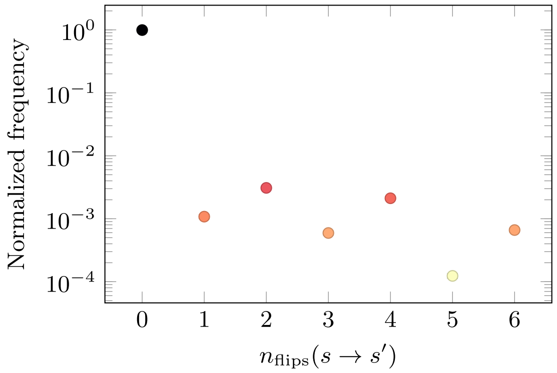

FIG. 10. Frequency of updates leading to $n_{\text{flips}}(s \to s') \in \{0, \dots, 6\}$ net spin flips in a PGMC sampling of Eq. (18) at $(K, B) = (-4, 2.5)$ with policy π_8. Most updates involve no change, either because the proposed action effectively is $a_\varnothing$ or because the proposed update was rejected. The error bars, indicating the sampled standard deviation, are smaller than the symbol size. See the Appendix for technical details.

hence ignoring training samples that involve two or fewer spin flips.

Figure 8 shows that π_7 and π_8 continue the trends of π_5 and π_6: When the elementary policy is able to follow the probability distribution given by the model (18), the performance is similar or improved, while if not, the consequence is more severe. Observe that π_7 and π_8 are the only policies presented so far that have the potential to properly sample the true kagome spin ice ground macrostate $K \to -\infty$ and $B \neq 0$, regardless of available computer power.

D. Stochastic chain policies

Figure 10 reveals what can also be deduced from Fig. 8: At $(K, B) = (-4, 2.5)$, most PGMC iterations guided by π_8 do not lead to a change in configuration. Furthermore, most of the spin flips that do take place are basically spent undoing previous flips. Overall, the Markov chain diffusion through state space is computationally inefficient and very slow.

Loop and worm algorithms [35–40], hand crafted for taking advantage of the stringy structure of the ground states of many frustrated models, like Eq. (18) with $K \ll 0$, will typically perform much better than the policies used so far. This raises the question of whether it is possible to construct a policy model that can mimic such an algorithm. The answer is affirmative, as can be demonstrated with the following worm policy scheme.[16]

(1) Initially, select and flip a random spin at site i_1 according to the single-flip policy π_{start}, $a_1 = a_{\sigma_{i_1}}$. (This time it is not necessary to include $a_\varnothing$ in the elementary action space.) Mark the site as the head of the worm.

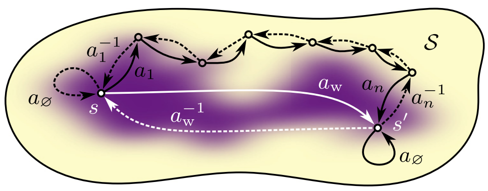

FIG. 11. Worm action $a_{\text{w}} = a_\varnothing a_n \cdots a_1 : s \mapsto s'$ (solid arrows) and its inverse $a_{\text{w}}^{-1} = a_\varnothing a_1^{-1} \cdots a_n^{-1} : s' \mapsto s$ (dashed arrows).

(2) For $t = 2, 3, \dots$, until the worm construction is terminated after n spin flips, according to elementary policy π_{move}, either pick a nearest-neighbor site j of the head i_{t-1}, flip σ_j ($a_t = a_{\sigma_j}$), move the head $i_t \leftarrow j$, and repeat the step with $t \leftarrow t + 1$, or terminate the worm construction ($a_t = a_{n+1} = a_\varnothing$).

(3) The complete worm construction $s = s_1 \to a_{\text{w}} s \equiv a_\varnothing a_n \cdots a_1 s_1 = s_{n+1} = s'$ can then be seen as the action chosen by the worm policy

$$\pi_{\text{worm}}(s \to s') = \pi_{\text{start}}(s_1 \to s_2) \prod_{t=2}^{n} \pi_{\text{move}}(s_t \to s_{t+1}) \times \pi_{\text{move}}(s_{n+1} \to s_{n+1}), \tag{33}$$

where $s_{t+1} = a_t s_t$.

Observe that the inverse action a_{w}^{-1} will not follow the exact same chain of elementary actions as a_{w} (in reverse), since, in order for the worm policy to be reversible, $a_{\text{w}}^{-1} = a_\varnothing a_1^{-1} \cdots a_n^{-1} \neq a_1^{-1} \cdots a_n^{-1} a_\varnothing$ (see Fig. 11).

The worm policy is an example of a chain policy of stochastic length, or stochastic chain policy. In a stochastic chain policy, a good distribution of lengths will be (indirectly) machine learned, instead of fixed *a priori* by a hyperparameter.

The training of the stochastic chain policy proceeds in very much the same fashion as before, but now with the training performance factor estimate (32) normalized by the length of the worm n. It is important to incorporate such a normalization to reflect the cost of creating a longer worm and prevent the average worm length from diverging.

We test the above-sketched worm policy in the form of π_9, for which $\pi_{\text{start}} = \pi_4$ and π_{move} is given by the same preference (24) (with a different set of parameters $\boldsymbol{\theta}$) but with the action space $\mathcal{A}_{\boldsymbol{\sigma},i} = a_\varnothing \cup \{a_{\sigma_j} \mid j \text{ nearest neighbor(s) of } i\}$, where i is the current position of the head. In total, there are $9 + 10 = 19$ independent parameters to be determined.

Figure 8 reveals π_9's success in simulating the kagome ice regime, with close to perfect sampling at $(K, B) = (-4, 0.5)$ and $(K, B) = (-4, 2.5)$, orders of magnitude better than what is achieved by π_1–π_8. In the other parameter regimes, where the important states are of another, less stringy character, the worm policy does not offer any advantages.

The universality and great flexibility of the PGMC framework makes it easy to explore more sophisticated policy models. For example, it is only a matter of imposing a few restrictions on the action space of the worm policy to essentially mimic (and extend) the highly specialized directed worm algorithms [41,42] within PGMC. The idea is to suppress

[16] Only one out of several possible worm policies is given; others may be constructed by permuting the order in which the movement, flip, and termination actions are selected. Note also that the worm policy operates in the ordinary spin space, as opposed to the dual space representation used in, e.g., Ref. [40].

or prevent the worm from backtracking itself, letting fewer elementary actions be wasted on reversing what has been done. In Refs. [41,42] this is achieved by carefully solving a set of coupled equations. Here we simply remove the actions associated with the m sites last visited by the worm head (including the current one) from the elementary action space. For instance, if $m = 2$, both the spin at the current position of the head i_t and the spin at the previously visited site (if it exists) i_{t-1} are excluded from possibly being flipped back in the next step: $\mathcal{A}_{\boldsymbol{\sigma},i_t} \leftarrow \mathcal{A}_{\boldsymbol{\sigma},i_t} \setminus \{a^{-1}_{\sigma_{i_t}}, a^{-1}_{\sigma_{i_{t-1}}}\}$.

It should be stressed that including a time dependence or memory restricted to within an action is perfectly valid in an MCMC simulation; the elementary policies do not need to be Markovian, as long as the entire chain policy is.[17]

We test two worm policies with memory:[18] π_{10}, which is like π_9 but with memory $m = 2$, hence with immediate backtracking blocked, and π_{11}, which is like π_9 but with memory $m = 3$. Now no more than three consecutive spin flips may happen within an elementary triangle of the kagome lattice. (By construction, π_9 has $m = 1$.) Here π_{10} and π_{11} do not result in any significant change from the already excellent sampling in the kagome ice regime by π_9, but the performances in the other parameter regimes are improved (see Fig. 8).

Now, in light of the criticism of hand-crafted effective models within EMMC (Sec. I), it is not unreasonable to be equally reluctant towards hand-crafted policies, like the worm policies just presented. The physics of Eq. (18) is well known, making targeted policy modeling possible, but what if it was not? The final policy model to be presented attempts to tackle this issue.

With the success of the worm policies in mind, the idea is rather straightforward: Remove the special worm constraint, but keep the much more general memory constraint. In other words, let the policy be a stochastic chain policy with an elementary action space

$$\mathcal{A}_{s,t} = \mathcal{A}_s \setminus \left\{a^{-1}_{t-\delta} \mid \delta \in \{1, \ldots, m\}\right\}, \quad a_{t\leqslant 0} = \varnothing \tag{34}$$

where $\mathcal{A}_s$ is the space of all possible elementary actions and $a_1, a_2, \ldots$ are the elementary actions taken so far. The constraint on immediately reversing elementary actions forces the policy to explore state space, instead of falling into the computationally wasteful local minima (attractive fix points) of bouncing. We call such a policy a short-term self-avoiding policy with memory m.

To demonstrate the short-term self-avoiding idea, we simply replace the π_{move} elementary policy of the worm algorithms π_9–π_{11} with a π_{SA} self-avoiding elementary policy. The spin-flip preference of π_{SA} is given by h_4 [Eq. (24)] and instead of using just a constant preference for terminating the update, we choose to use a slightly more involved (and flexible) stopping preference

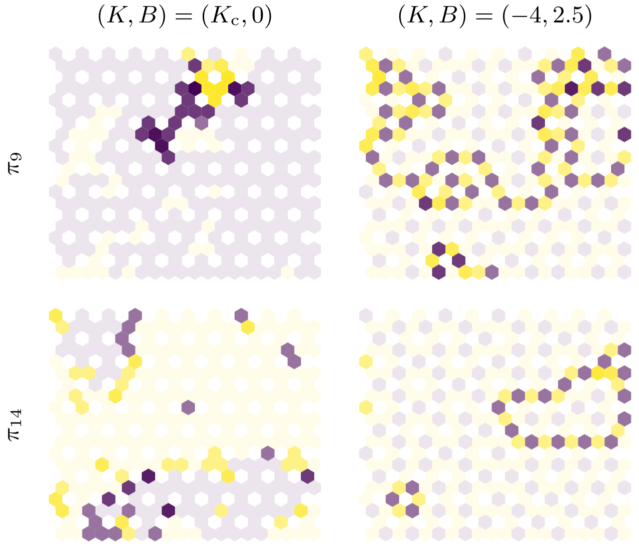

FIG. 12. Snapshots of configurations and most recent actions leading to them, at the critical point (left) and in the kagome spin ice regime (right). In the top panels, the actions have been selected by the worm policy π_9, while in the bottom panels, they have been selected by the short-term self-avoiding policy π_{14}. Stronger intensity indicates that the same spin has been flipped (and flipped back) multiple times within the last action, while the weakly colored background spins have not been flipped at all. Note that while the elementary actions of the worm policy, by construction, form a single connected set, the actions of the short-term self-avoiding policy have no such restrictions imposed. This makes the latter more flexible in adjusting to other parameter regimes, where string or loop updates may no longer be optimal.

$$h_{\text{stop}}(a_\varnothing|\boldsymbol{\sigma}) = \ln\left[\sum_{i=1}^{N} \exp\left(\theta^{\text{stop}}_{q_i}\right)\right] \tag{35}$$

parametrized by $\boldsymbol{\theta}^{\text{stop}}$. As for h_4, the local feature map q_i is also given by Eq. (25). In total, there are 29 independent parameters.

Four such short-term self-avoiding policies are tested: π_{12}, with memory $m = 1$; π_{13}, with memory $m = 3$; π_{14}, with memory $m = 6$; and π_{15}, with memory $m = 10$. As can be seen in Fig. 8, avoiding backtracking actions is indeed the crucial ingredient in improving the Markov chain dynamics, with π_{12}–π_{15} performing better than any of the other nonspecialized policies in the deep spin ice regime. The short-term self-avoiding policies may not be quite as efficient as the worm policies here, but crucially their aptness come from machine learning, rather than a hard coded design.

Figure 12 compares updates generated by the worm policy π_9 and the short-term self-avoiding policy π_{14} in the critical and frustrated regimes. In the latter, both construct efficient nonlocal loop updates. In the critical regime, however, the worm policy is no longer particularly advantageous; the stringy worm updates do not cope well with the fractal nature of the critical configurations, resulting in inefficient updates containing multiple bounce processes. The short-term

[17] This is not unique to the stochastic chain policies presented in this work. All traditional cluster algorithms are basically based on this fact. After all, they may themselves be cast as stochastic chain policies within the PGMC framework.

[18] These are the only two worm policies with memory, apart from $m = 1$, that make sense in the kagome lattice simulations. Longer memories either do not add extra benefits or run the risk of trapping the worm head in a position it cannot escape.

self-avoiding policy, on the other hand, does not suffer from the restricted action space of the worm policy and may therefore to a greater degree adjust to the physics experienced.

It is not unreasonable to speculate that a policy model even more flexible than π_{14} could rediscover a cluster algorithm [43,44] at criticality, as the cluster algorithms may themselves be regarded as stochastic chain policies with memory. If so, a single policy model would be sufficient to competitively simulate Eq. (18) in the entire (K, B) space.

The last to be examined in this work, the short-term self-avoiding policy model certainly does not represent the end of the story. The need for selecting the right memory, i.e., too short and the benefits of self-avoidance are not maximized, too long and the policy is no longer able to adequately track the desired probability distribution, again means tedious and inefficient hyperparameter tuning. It is not hard to envision better and more flexible policies that deal with such issues and more; after all, the stochastic chain policies with memory presented constitute only a few members of a broader class of policies with history-dependent elementary policies. In principle, such a history dependence can also be modeled by a differentiable structure and subsequently machine learned. The practicalities, however, are left for future work.

E. PGMC for other systems

While ideas like the chain and self-avoiding policies do not depend on the specifics of the probability weight model in question, other aspects of the preceding examples do. Specifically, the (elementary) policy models of the kagome Ising model all exploit the translational invariance of Eq. (18) and the binary nature of the Ising spins, both nonuniversal case-dependent traits. Even though the PGMC idea is generally applicable, specific policies may not be.

Clearly, if symmetries like translational invariance are not present in the system of interest, this should be reflected in the policy model. Not only is this likely to render the training stage more expensive (there is less information reuse when symmetries are not imposed), a feature map approach, such as the one employed in this work, may no longer be computationally efficient or feasible. Then other action selection strategies may be necessary, like those briefly mentioned in Sec. III B (see also the Appendix).

Furthermore, if the degrees of freedom are not binary, it is no longer sufficient to let the elementary actions be of the form of flips, i.e., to change the value of some degree of freedom i to its other value. In cases where a degree of freedom still constitutes a finite discrete set χ (e.g., $\chi = \mathbb{Z}_n$ with $n > 2$) it may be possible to construct (chain) actions of the following form: select a degree of freedom i and change it from σ_i to σ_i', $\sigma_i, \sigma_i' \in \chi_i$. Assigning individual weights to all elements of the (local) action space is however viable only up to some point. Beyond this, for cases where the number of elements in χ is too large or even infinite (i.e., for continuous degrees of freedom, like $\chi = \mathbb{R}$, $\chi = \mathrm{O}(2)$, etc.) some degree of probability weight or density interpolation has to take place.

As an example, in the case of a classical XY model [$\chi = \mathrm{O}(2)$], each degree of freedom can be parametrized by a continuous angle $\phi \in [0, 2\pi)$. A possible policy for updating a single degree of freedom may then read as follows.

(1) Select a site i with probability $\pi(i|s)$.

(2) Then select a sector $\chi_k \equiv [\frac{2\pi k}{n}, \frac{2\pi(k+1)}{n})$, $k \in \{0, 1, \dots, n-1\}$, with probability $\pi(\chi_k \mid \text{neighborhood of } i)$.

(3) Finally, select a new angle $\phi_i' \in \chi_k$ by sampling a uniform random number $R \in [0, 1)$ and then calculate $\phi_i' = \frac{2\pi(k+R)}{n}$.

The two first steps can then be modeled (parametrized by $\boldsymbol{\theta}$) and learned in a way similar to what was done for the kagome Ising model of the previous sections.[19] Since the infinitesimal probability weight of step 3 is constant and independent of state, its value is not of importance when calculating Eq. (8).

Interestingly, systems of continuous degrees of freedom may possess additional geometric information not available in the discrete systems. Thus, for systems of continuous degrees of freedom, one may contemplate policies where not only the state, but also properties like the state space gradient of the probability weight (or even a machine-learned effective model) is taken into account. Such a scheme could then be considered a PGMC equivalent to the Metropolis-adjusted Langevin algorithms (MALAs) [45–47].

V. DISCUSSION

After having seen some of PGMC's potential in the case studies of the preceding section, we now turn back to the more general perspective. The aim of the following brief discussion is to put PGMC somewhat in context with respect to MCMC in general and EMMC in particular. We also mention a few challenges and possible pitfalls concerning PGMC.

A. Machine-learning MCMC

Common to all traditional Monte Carlo algorithms is the trade-off between the inclusion of *a priori* knowledge and stochasticity, specialization, and generality: A very general, hence also "oblivious," Monte Carlo algorithm, which is applicable to many problems, will typically perform poorer than a specialized but limited algorithm that exploits many aspects of the problem at hand. Intriguingly, MCMC methods incorporating machine learning, like EMMC and PGMC, have the potential to bridge this gap: By letting the algorithms automatically learn from and adjust to the sampled distribution, these methods may, in a sense, be both general and specialized at the same time: general in usability and specialized in computation.

The value of such a property should not be understated. Even if algorithms hard coded by domain experts may surpass the performance of machine-learned algorithms in some cases, the usefulness of having general, flexible, and still reasonably well performing tools is significant in many real-world scenarios.

Looking further, there is nothing intrinsic that prevents the machine-learning methods from surpassing what is already known. In fact, for anything beyond the simplest and most

[19] There are obviously infinitely many alternative (and probably better) policy models in this particular case. Determining a good one is outside the scope of this work.

well understood models, machine-learning methods are likely to be a practical necessity if one wishes to reap the benefits of specialization.

B. EMMC and PGMC

The main difference between the EMMC and PGMC methods lies in their computational perspectives: While EMMC is static, exchanging one model (the original one) for another (the effective), PGMC is dynamic, attempting to model the Markov chain dynamics directly. Whereas the effective models of EMMC are constructed by imitating the training data, the policy search of PGMC aims at surpassing what is currently known. The autocorrelation, a central trait of the final MCMC sampling, is specifically taken into account in the PGMC policy optimization, while it is disregarded in EMMC's effective model training.

The PGMC scheme offers other advantages in the optimization stage, both conceptually and practically: The goal is clearly and quantitatively stated (maximize the expected performance factor), regardless of implementation details. This provides an explicit path forward, leading directly to efficient real-world algorithms, as seen in Sec. IV. Efficient bootstrapping techniques, like on-policy learning, is a natural consequence of the dynamical perspective.

The global view that readily comes with the policies is another asset of PGMC: Computational resources can easily be directed towards the most important or relevant degrees of freedom of a state. Exemplified in Fig. 7, even a relatively simple policy may target structures like domain walls or particlelike excitations, while spending less time on fluctuations of minor importance. Thus, a PGMC simulation can easily approach a performance that would otherwise require more specialized and involved rejection-free MCMC schemes.

The situation is less clear for EMMC. Even with an effective model in place, there is still a need to find or design good updates, with no definite guidelines to follow. The effective model updates used are therefore prone to be either generic and relatively poor or specialized and of limited applicability.

C. Pitfalls and challenges

In machine-learned MCMC, as in reinforcement learning, there is danger in trading stochastically sampled training data for increased simulation speed: If the available information is inadequate or wrong, the effective models or policies, and hence the final sampling, may be bad. In PGMC, extra care has to be taken when the policy training is conducted online, on policy. The reason is that the optimization may, in some situations, enter a feedback loop where the policy, with an increasingly strong bias, selects update proposals among only a subset of relevant states. The exploration of the state space stops and, for all practical purposes, ergodicity is broken. Taking the worm PGMC of Sec. IV D as an example, if the policy training was not properly controlled, it could happen that the learned policy would end up staying entirely within the kagome spin ice manifold. The correct sampling at $(K, B) = (-4, 2.5)$ should contain a low density of excited states. Nevertheless, these excitations may not always have been encountered sufficiently often during a crucial period of the training, leading the policy into a spiral of suboptimization where excited states were not proposed because excited states were not proposed.

Such problems, although mostly absent from the simple test cases of Sec. IV, are likely to happen more if policy model complexity, hence also (overfitting) capacity, is increased. A possible solution, similar to what is done within reinforcement learning [15], could be to move away from strict on-policy-based learning by including some stochastic noise in the behavior policy. For example, a uniformly random action could be selected with some low frequency, sacrificing a little bootstrapping performance for a persistent exploration of the state space.

Another set of issues may arise with the training of intricate policies, e.g., chain policies with long actions. First, a larger policy model requires more information and is typically more demanding to train. Second, the longer actions take longer to construct, hence the frequency of generating training samples is reduced. Furthermore, the longer delay between selecting elementary actions and evaluating the total action means that the propagation of information back through the chain will be less efficient and more noisy, as the effect of later actions may strongly depend on earlier ones. Overall, the positive aspects of having more complex, globally updating policies are also the ones that make it harder to train them. As such, the issues cannot easily be circumvented. Fortunately, however, many of these sides of policy training are well known and tackled within the reinforcement learning community, with solutions ripe for being incorporated into PGMC: It could, for instance, be fruitful to use information acquired during the updates to build value or action-value function models, either to support the policy model training or as an alternative to the policy models discussed in this work [15].

Finally, the stability and convergence speed of the training stage can likely be improved by storing and reusing more of previously obtained data in an offline, off-policy approach (experience replay), as demonstrated in Ref. [48]. With a slight modification, such a method could also benefit from simulation data obtained with slightly different models (e.g., the model of interest, but at a different temperature), analogous to what is achieved with reweighting techniques [49,50].

VI. CONCLUSION

In this paper, we have seen how reinforcement learning and Markov chain Monte Carlo simulations may be combined in a general, unbiased, and powerful framework: policy-guided Monte Carlo. The basic theory of PGMC has been developed and subsequently scrutinized, with the aim of advancing PGMC as a practical tool for both improving and expanding the scope of MCMC sampling, as well as automatizing the process of developing alternative, efficient MCMC algorithms. Examples of increasing sophistication, centered around a simple but challenging Ising model on the kagome lattice, underlined the effort and showed how PGMC may be done in practice. We also discussed PGMC's relation to other MCMC algorithms and where care has to be taken.

The gains of PGMC may span the entire spectrum, from modest speedups in handling known problems to novel samplers tackling new ones. We therefore expect PGMC to be widely beneficial in the realm of MCMC simulations.

ACKNOWLEDGMENTS

The author would like to thank Shigeki Onoda, Yukitoshi Motome, and Yasuyuki Kato for fruitful discussions and feedback related to this work. He would also like to thank an anonymous referee for bringing the adaptive MCMC algorithms and the MALAs to his attention. This work was supported by Grants-in-Aid for Scientific Research (KAKENHI) (Grant No. JP16H02206).

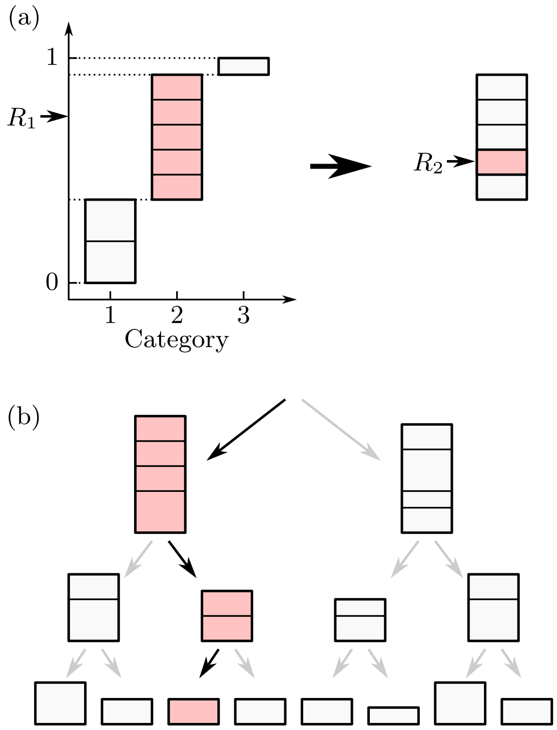

FIG. 13. (a) Inverse transform and (b) weighted binary tree selection of an action based on some policy π. Each box represents some action a, with the height being given by $\pi(a)$. In (a), the action space is sorted into feature categories. A category is chosen (colored) according to the total marginal probability of the category by drawing a uniform random number R_1. Next, a specific action is selected uniformly at random among the members of the category by random number R_2. The total number of operations is constant with respect to the size of the action space $N_{\mathcal{A}}$. In (b), the probability (weights) are arranged in a binary tree structure, with the weight of each parent node being the sum of its children nodes. An action is selected by traversing the tree from root to leaf, each time selecting the left or the right child at random, proportionally to their relative weights. This scheme comes at the (mild) expense of an $\mathcal{O}(\log N_{\mathcal{A}})$ scaling in the number of operations needed. The advantage is that it can be efficiently used even when the number of possible categories is large or even infinite. Naturally, when an action has been applied to a state, both the category groupings in (a) and the weights of the tree in (b) will have to be updated. In cases with short-range feature maps (i.e., when the features only depend on a finite neighborhood of the degree of freedom being updated by the action) this can also be done with $\mathcal{O}(1)$ and $\mathcal{O}(\log N_{\mathcal{A}})$ numbers of operations, respectively.

APPENDIX: SIMULATION DETAILS

1. Implementation

All implementations were done in the JULIA [51] programming language with the help of the FLUX machine learning library [52] and its implementation of the ADAM stochastic gradient descent algorithm [53].

Actions were efficiently chosen by first picking a feature category using inverse transform sampling and then selecting among the equally weighted members of the category. An alternative approach of using a weighted binary tree was also successfully tested, although not used in the test simulations presented here. Both methods are illustrated in Fig. 13. Naturally, the complete information about the actions had to be temporarily stored, both for calculating the policy value of the inverse action and for being able reconstruct the previous state in case of rejection.

2. Simulation parameters used

In all the displayed simulation results, the system size was set to $L = 10$ and the spins were initialized at random. The policy parameters were set to $\boldsymbol{\theta} = \mathbf{0}$ before training. The

TABLE II. Simulation parameters used in each of the 16 independent simulation series behind each data point presented in Fig. 8. A thinning factor of x means that only one out of every x MCMC iteration steps is sampled. (The timescale is still set by a single iteration.)

		Training	Equilibration	Thinning	
(K, B)	Policies	iterations	iterations	factor	Samples
$(K_c, 0)$	π_1		$5000N$	$50N$	5000
	π_2–π_4	$100N$	$100N$	$50N$	5000
	π_5–π_7	$100N$	$100N$	$10N$	10000
	π_8	$200N$	$100N$	$10N$	10000
	π_9–π_{11}	$500N$	1000	100	5000
	π_{12}–π_{15}	$100N$	1000	100	5000
(0.5,1)	π_1		$500N$	$1N$	5000
	π_2–π_4	$100N$	$100N$	15	5000
	π_5–π_7	$100N$	$100N$	1	50000
	π_8	$200N$	$100N$	1	50000
	π_9–π_{11}	$100N$	1000	1	10000
	π_{12}–π_{15}	$100N$	1000	1	10000
$(-4, 0)$	π_1		$100N$	$1N$	2000
	π_2–π_4	$100N$	$100N$	100	5000
	π_5–π_7	$100N$	$100N$	20	5000
	π_8	$200N$	$100N$	20	5000
	π_9–π_{11}	$200N$	1000	5	10000
	π_{12}–π_{15}	$200N$	1000	1	10000
$(-4, 0.5)$	π_1		$2000N$	$50N$	2000
	π_2–π_4	$100N$	$1000N$	$2N$	10000
	π_5–π_7	$100N$	$1000N$	N	5000
	π_8	$200N$	$1000N$	N	5000
	π_9–π_{11}	$300N$	1000	5	10000
	π_{12}–π_{15}	$200N$	1000	5	10000
$(-4, 2.5)$	π_1		$5000N$	$50N$	2000
	π_2–π_4	$100N$	$2000N$	$500N$	1000
	π_5–π_7	$100N$	$2000N$	$10N$	5000
	π_8	$200N$	$2000N$	$10N$	5000
	π_9–π_{11}	$300N$	1000	1	10000
	π_{12}–π_{15}	$200N$	1000	1	10000

other simulation parameters were chosen as to obtain reliable results, e.g., clear convergence and more-than-sufficient equilibration. Figure 8 and 10 show averages and standard deviations based on 16 completely independent simulation runs. The simulation parameters of Fig. 8 are listed in Table II. In Fig. 10, a simulation run consisted of $200N$ training iterations, $1000N$ equilibration iterations, and 10^7 samples (without thinning).

[1] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, J. Chem. Phys. **21**, 1087 (1953).

[2] C. Robert and G. Casella, Stat. Sci. **26**, 102 (2011).

[3] F. Sullivan and J. Dongarra, Comput. Sci. Eng. **2**, 22 (2000).

[4] W. K. Hastings, Biometrika **57**, 97 (1970).

[5] D. P. Landau and K. Binder, *A Guide to Monte-Carlo Simulations in Statistical Physics*, 3rd ed. (Cambridge University Press, Cambridge, 2009).

[6] L. Huang and L. Wang, Phys. Rev. B **95**, 035105 (2017).

[7] J. Liu, Y. Qi, Z. Y. Meng, and L. Fu, Phys. Rev. B **95**, 041101 (2017).

[8] O. Trushin, A. Karim, A. Kara, and T. S. Rahman, Phys. Rev. B **72**, 115401 (2005).

[9] L. Huang, Y.-f. Yang, and L. Wang, Phys. Rev. E **95**, 031301 (2017).

[10] J. Liu, H. Shen, Y. Qi, Z. Y. Meng, and L. Fu, Phys. Rev. B **95**, 241104(R) (2017).

[11] X. Y. Xu, Y. Qi, J. Liu, L. Fu, and Z. Y. Meng, Phys. Rev. B **96**, 041119 (2017).

[12] Y. Nagai, H. Shen, Y. Qi, J. Liu, and L. Fu, Phys. Rev. B **96**, 161102 (2017).

[13] C. Chen, X. Y. Xu, J. Liu, G. Batrouni, R. Scalettar, and Z. Y. Meng, Phys. Rev. B **98**, 041102 (2018).

[14] Z. H. Liu, X. Y. Xu, Y. Qi, K. Sun, and Z. Y. Meng, Phys. Rev. B **98**, 045116 (2018).

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. (MIT Press, Cambridge, MA, 2018).

[16] G. O. Roberts, A. Gelman, and W. R. Gilks, Ann. Appl. Probab. **7**, 110 (1997).

[17] G. O. Roberts and J. S. Rosenthal, J. Comput. Graph. Stat. **18**, 349 (2009).

[18] C. Andrieu and J. Thoms, Stat. Comput. **18**, 343 (2008).

[19] K. Łatuszyński, G. O. Roberts, and J. S. Rosenthal, Ann. Appl. Probab. **23**, 66 (2013).

[20] J. Yang and J. S. Rosenthal, Comput. Stat. **32**, 315 (2017).

[21] L. Wang, Phys. Rev. E **96**, 051301 (2017).

[22] H. Müller-Krumbhaar and K. Binder, J. Stat. Phys. **8**, 1 (1973).

[23] A. Sokal, in *Functional Integration: Basics and Applications*, edited by C. DeWitt-Morette, P. Cartier, and A. Folacci (Springer US, Boston, 1997), pp. 131–192.

[24] L. Bottou, F. E. Curtis, and J. Nocedal, SIAM Rev. **60**, 223 (2018).

[25] C. M. Bishop, *Pattern Recognition and Machine Learning* (Springer, New York, 2006).

[26] I. Syôzi, Prog. Theor. Phys. **6**, 306 (1951).

[27] K. Kanô and S. Naya, Prog. Theor. Phys. **10**, 158 (1953).

[28] M. Udagawa, M. Ogata, and Z. Hiroi, J. Phys. Soc. Jpn. **71**, 2365 (2002).

[29] A. J. Macdonald, P. C. W. Holdsworth, and R. G. Melko, J. Phys.: Condens. Matter **23**, 164208 (2011).

[30] C. Castelnovo, R. Moessner, and S. L. Sondhi, Nature (London) **451**, 42 (2008).

[31] C. E. Shannon, Bell Syst. Tech. J. **27**, 379 (1948).

[32] A. Bortz, M. Kalos, and J. Lebowitz, J. Comput. Phys. **17**, 10 (1975).

[33] K. A. Fichthorn and W. H. Weinberg, J. Chem. Phys. **95**, 1090 (1991).

[34] E. A. J. F. Peters and G. de With, Phys. Rev. E **85**, 026703 (2012).

[35] D. Kandel, R. Ben-Av, and E. Domany, Phys. Rev. Lett. **65**, 941 (1990).

[36] G. T. Barkema and M. E. J. Newman, Phys. Rev. E **57**, 1155 (1998).

[37] H. Shinaoka and Y. Motome, Phys. Rev. B **82**, 134420 (2010).

[38] Y. Wang, H. De Sterck, and R. G. Melko, Phys. Rev. E **85**, 036704 (2012).

[39] H. Otsuka, Phys. Rev. B **90**, 220406 (2014).

[40] G. Rakala and K. Damle, Phys. Rev. E **96**, 023304 (2017).

[41] O. F. Syljuåsen and A. W. Sandvik, Phys. Rev. E **66**, 046701 (2002).

[42] F. Alet and E. S. Sørensen, Phys. Rev. E **68**, 026702 (2003).

[43] R. H. Swendsen and J.-S. Wang, Phys. Rev. Lett. **58**, 86 (1987).

[44] U. Wolff, Phys. Rev. Lett. **62**, 361 (1989).

[45] G. O. Roberts and R. L. Tweedie, Bernoulli **2**, 341 (1996).

[46] M. Girolami and B. Calderhead, J. R. Stat. Soc. B **73**, 123 (2011).

[47] A. Durmus, G. O. Roberts, G. Vilmart, and K. C. Zygalakis, Ann. Appl. Probab. **27**, 2195 (2017).

[48] 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, Nature (London) **518**, 529 (2015).

[49] A. M. Ferrenberg and R. H. Swendsen, Phys. Rev. Lett. **61**, 2635 (1988).

[50] A. M. Ferrenberg and R. H. Swendsen, Phys. Rev. Lett. **63**, 1195 (1989).

[51] J. Bezanson, A. Edelman, S. Karpinski, and V. Shah, SIAM Rev. **59**, 65 (2017).

[52] M. Innes, J. Open Source Software **3**, 602 (2018).

[53] D. P. Kingma and J. Ba, arXiv:1412.6980.