Approximate Inference in Discrete Distributions with Monte Carlo Tree Search and Value Functions

Lars Buesing, Theophane Weber, Nicolas Hees
DeepMind

Abstract

Exact probabilistic inference in discrete models is often prohibitively expensive, as it may require evaluating the (unnormalized) target density on its entire domain. Here we consider the setting where only a limited budget of calls to the unnormalized target density oracle is available, raising the challenge of where in the domain to allocate these function calls in order to construct a good approximate solution. We formulate this problem as an instance of *sequential decision-making under uncertainty* and leverage methods from reinforcement learning for probabilistic inference with budget constraints. In particular, we propose the TREESAMPLE algorithm, an adaptation of Monte Carlo Tree Search to approximate inference. This algorithm caches all previous queries to the density oracle in an explicit search tree, and dynamically allocates new queries based on a "best-first" heuristic for exploration, using existing upper confidence bound methods. Our non-parametric inference method can be effectively combined with neural networks that compile approximate conditionals of the target, which are then used to guide the inference search and enable generalization across multiple target distributions. We show empirically that TREESAMPLE outperforms standard approximate inference methods on synthetic factor graphs.

1 Introduction

Probabilistic models are often easy to specify, e.g. by multiplying non-negative functions that each reflect an independent piece of information, yielding an *unnormalized target density* (UTD). However, extracting knowledge from the model, such as marginal distributions of variables, is notoriously difficult. For discrete distributions, inference is #P-complete (Roth, 1996), and thus at least as hard as (and suspected to be much harder than) NP-complete problems (Stockmeyer, 1985).

The hardness of exact inference, which often prevents its application in practice, has led to the development of numerous approximate methods such as Markov Chain Monte Carlo (MCMC) and Sequential Monte Carlo (SMC) (Hastings, 1970; Del Moral et al., 2006). Whereas exact inference methods need to evaluate and sum the UTD over its entire domain in the worst case, approximate methods attempt to reduce computation by concentrating evaluations of the UTD on regions of the domain that contribute most to the probability mass. The exact locations of high-probability regions are, however, often unknown a-priori, and different approaches use a variety of means to identify them efficiently. In continuous domains, Hamiltonian Monte Carlo, for instance, guides a set of particles towards high density regions by using gradients of the target density (Neal et al., 2011). Instead of exclusively relying on a-priori knowledge (such as a gradient oracle), adaptive approximation methods use the outcome of previous evaluations of the UTD to dynamically allocate subsequent evaluations on promising parts of the domain (Mansinghka et al., 2009; Andrieu and Thoms, 2008). This can be formalized as an instance of *decision-making under uncertainty*, where acting corresponds to evaluating the UTD in order to discover probability mass in the domain (Lu et al., 2018). Form this viewpoint, approximate inference methods attempt to *explore* the target domain based on a-priori information about the target density as well as on *partial feedback* from previous evaluations of the UTD.

In this work, we propose a new approximate inference method for discrete distributions, termed TREESAMPLE, that is motivated by the correspondence between probabilistic inference and sequential decision-making highlighted previously in the literature, e.g. (Dayan and Hinton, 1997; Rawlik et al., 2013; Weber et al., 2015). TREESAMPLE approximates a joint distribution over multiple discrete variables by a *sequential decision-making* approach: Variables are inferred / sampled one variable at a time based on all

Proceedings of the 23rdInternational Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

previous ones in an arbitrary, pre-specified ordering. An explicit tree-structured cache of all previous UTD evaluations in maintained, and a heuristic inspired by Upper Confidence Bounds on Trees (UTC) (Kocsis and Szepesvári, 2006) for trading off exploration around configurations that were previously found to yield high values of UTD and configurations in regions that have not yet been explored, is applied. Algorithmically, TREESAMPLE amounts to a variant of Monte Carlo Tree Search (MCTS) (Browne et al., 2012), modified so that it performs integration rather than optimization. In contrast to other approximate methods, it leverages systematic, backtracking tree search with a "best-first" exploration heuristic. Inspired by prior work on combining MCTS with function approximation (Silver et al., 2016), we further augment TREESAMPLE with neural networks that parametrically cache previously computed approximate solutions of inference sub-problems. This enables *generalization* across branches of the search tree for a given target density as well as across inference problems for different target densities. In particular, we experimentally show that suitably structured neural networks such as Graph Neural Networks (Battaglia et al., 2018) can efficiently guide the search even on new problem instances by reducing the effective search space.

2 Inference with Budget Constraint

Notation Let $X = (X_1, \ldots, X_N) \sim P^*_X$ be a discrete random vector taking values $x = (x_1, \ldots, x_N)$ in $\mathbb{X} := \{1, \ldots, K\}^N$, and let $x_{\leq n} := (x_1, \ldots, x_n) \in \mathbb{X}_{\leq n}$ be its n-prefix and define $x_{<n} \in \mathbb{X}_{<n}$ analogously. We assume the distribution P^*_X is given by a factor graph. Denote with γ^* its density:

$$\log \gamma^*(x) = \sum_{m=1}^{M} \psi_m(x) - \log \sum_{x \in \mathbb{X}} \exp \sum_{m=1}^{M} \psi_m(x). \quad (1)$$

We denote with Z the normalization constant and with $\hat{\gamma}$ the unnormalized density. We assume that all factors ψ_m, defined in the log-domain, take values in $\mathbb{R} \cup \{-\infty\}$. Furthermore, $\mathrm{scope}(\psi_m)$ is assumed to be known for all factors, where $\mathrm{scope}(\psi_m) \subseteq \{1, \ldots, N\}$ is the index set of the variables that ψ_m takes as input. We denote the densities of the conditionals $P^*_{X_n|x_{<n}}$ as $\gamma^*_n(x_n|x_{<n})$.

Problem Setting and Motivation Consider the problem of constructing a tractable approximation P_X to P^*_X. In this context, we define tractable as being able to sample from P_X (say in polynomial time in N). Such a P_X then allows Monte Carlo estimates of $\mathbb{E}_{P^*_X}[f] \approx \mathbb{E}_{P_X}[f]$ for any function f of interest in downstream tasks without having to touch the original P^*_X again. This setup is an example of model compilation (Darwiche, 2002). We assume that the computational cost of inference in P^*_X is dominated by evaluating any of the factors ψ_m. Therefore, we are interested in compiling a good approximation P_X using a fixed computational budget:

Input:	Factor oracles $\psi_1, \ldots, \psi_M$ with known $\mathrm{scope}(\psi_m)$; budget $B \in \mathbb{N}$ of pointwise evaluations of any ψ_m
Output:	Approximation $P_X \approx P^*_X$ that allows tractable sampling

A brute force approach would exhaustively compute all conditionals $P^*_{X_1|\varnothing}$, $P^*_{X_2|x_1}$ up to $P^*_{X_N|x_{<N}}$ and resort to ancestral sampling. This entails explicitly evaluating the factors ψ_m everywhere, likely including "wasteful" evaluations in regions of $\mathbb{X}$ with low density γ^*, i.e. parts of $\mathbb{X}$ that do not significantly contribute to P^*_X. Instead, it may be more efficient to construct an approximation P_X that concentrates computational budget on those parts of the domain $\mathbb{X}$ where the density γ^*, or equivalently $\hat{\gamma}$, is suspected to be high. For small budgets B, determining the points where to probe $\hat{\gamma}$ should ideally be done sequentially: Having evaluated $\hat{\gamma}$ on values $x^1, \ldots, x^b$ with $b < B$, the choice of x^{b+1} should be informed by the previous results $\hat{\gamma}(x^1), \ldots, \hat{\gamma}(x^b)$. If e.g. the target density is assumed to be "smooth", a point x "close" to points x^i with large $\hat{\gamma}(x^i)$ might also have a high value $\hat{\gamma}(x)$ under the target, making it a good candidate for future exploration (under appropriate definitions of "smooth" and "close"). In this view, inference presents itself as a structured *exploration* problem of the form studied in the literature on sequential decision-making under uncertainty and reinforcement learning (RL), in which we decide where to evaluate $\hat{\gamma}$ next in order to reduce uncertainty about its exact values. As presented in detail in the following, borrowing from the RL literature, we will use a form of tree search that preferentially explores points x^j that share a common prefix with previously found points x^i with high $\hat{\gamma}$.

3 Inference with Monte Carlo Tree Search

In the following, we cast sampling from P^*_X as a sequential decision-making problem in a suitable maximum-entropy Markov Decision Process (MDP). We show that the target distribution P^*_X is equal to the solution, i.e. the optimal policy, of this MDP. This representation of P^*_X as optimal policy allows us to leverage standard methods from RL for approximating P^*_X. Our definition of the MDP will capture the following intuitive procedure: At each step $n = 1, \ldots, N$ we decide how to sample X_n based on the realization $x_{<n}$ of $X_{<n}$ that has already been sampled. The reward function of the MDP will be defined such that the return (sum of rewards) of an episode will equal the unnormalized target density $\log \hat{\gamma}$, therefore "rewarding" samples that have high probability under the target.

3.1 Sequential Decision-Making Representation

We first fix an arbitrary ordering of the variables $X_1, \dots, X_N$; for now any ordering will do, but see the discussion in sec. 5. We then construct an episodic, maximum-entropy MDP $\mathcal{M} = ((\mathbb{X}_1, \dots, \mathbb{X}_{\leq N}), \mathcal{A}, \circ, R^\pi)$ consisting of episodes of length N. The state space at time step n is $\mathbb{X}_{<n}$ and the action space is $\mathcal{A} = \{1, \dots, K\}$ for all n. State transitions from $x_{<n}$ to $x_{\leq n}$ are deterministic: Executing action $a \in \mathcal{A}$ in state $x_{<n} \in \mathbb{X}_{<n}$ at step n results in setting x_n to a, or equivalently the action a is appended to the current state, i.e. $x_{\leq n} = x_{<n} \circ a = (x_1, \dots, x_{n-1}, a)$. A stochastic policy π in this MDP is defined by probability densities $\pi_n(a|x_{<n})$ over actions conditioned on $x_{<n}$ for $n = 1, \dots, N$. It induces a joint distribution P_X^π over $\mathbb{X}$ with the density $\pi(x) = \prod_{n=1}^N \pi_n(x_n|x_{<n})$. Therefore, the space of stochastic policies is equivalent to the space of distributions over $\mathbb{X}$.

We now define the reward function of $\mathcal{M}$:

Definition 1 (Reward)**.** *For $n = 1 \dots, N$, we define the reward function $R_n : \mathbb{X}_{\leq n} \to \mathbb{R} \cup \{-\infty\}$, as the sum over factors ψ_m that can be computed from $x_{\leq n}$, but not already from $x_{<n}$, i.e. :*

$$R_n(x_{\leq n}) := \sum_{\psi \in M_n} \psi(x_{\leq n}), \tag{2}$$

where $M_n := \{ \psi_m \mid \max(\text{scope}(\psi_m)) = n \}$. We further define the maximum-entropy reward as $R_n^\pi(x_{\leq n}) = R_n(x_{\leq n}) - \log \pi_n(x_n|x_{<n})$.

To illustrate this definition, assume ψ_1 is only a function of x_n; then it will contribute to R_n. If, however, it is has full support $\text{scope}(\psi_1) = \{1, \dots, N\}$, then it will contribute to R_N. Evaluating R_n at any input incurs a cost of $|M_n|$ towards the budget B. This completes the definition of $\mathcal{M}$. From the reward definition follows that we can write the logarithm of the unnormalized target density as the *return*, i.e. sum of rewards (without entropy terms) $\log \hat{\gamma}(x) = \sum_{n=1}^N R_n(x_{\leq n})$. We now establish that the MDP $\mathcal{M}$ is equivalent to the initial inference problem by using the standard definition of the value $V_n^\pi(x_{<n})$ of a policy as expected return conditioned on $x_{<n}$, i.e. $V_n^\pi(x_{<n}) := \mathbb{E}_\pi[\sum_{n'=n}^N R_{n'}^\pi]$ where the expectation $\mathbb{E}_\pi$ is taken over $P^\pi_{X_{\geq n}|x_{<n}}$ The following straight-forward observation holds:

Observation 1. *The value $V^\pi := V^\pi(\varnothing)$ of the initial state under π in the maximum-entropy MDP $\mathcal{M}$ is given by the negative KL-divergence between P_X^π and the target P_X^* up to the normalization constant Z:*

$$V^\pi = -D_{\text{KL}}[P_X^\pi \| P_X^*] + \log Z. \tag{3}$$

The optimal policy $\pi^* = \arg\max_\pi V^\pi$ is equal to the target conditionals $\gamma_n^*(x_n|x_{<n}) = \pi_n^*(x_n|x_{<n})$.

Algorithm 1 TREESAMPLE sampling procedure

```
1: procedure SAMPLE(tree $\mathcal{T}$, default $Q^\phi$)
2:     $x \leftarrow \varnothing$
3:     for $n = 1, \dots, N$ do
4:         if $x \in \mathcal{T}$ then
5:             $a \sim \text{softmax } Q_n(\cdot|x)$
6:         else
7:             $a \sim \text{softmax } Q_n^\phi(\cdot|x)$
8:         end if
9:         $x \leftarrow x \circ a$
10:    end for
11:    return $x$
12: end procedure
```

Therefore, solving the maximum-entropy MDP $\mathcal{M}$ is equal to finding all target conditionals $P^*_{X_n|x_{<n}}$, and running the optimal policy π^* yields samples from P_X^*. In order to convert the above MDP into a representation that facilitates finding a solution, we use the standard definition of the state-action values as $Q_n^\pi(x_n|x_{<n}) := R_n(x_{\leq n}) + V_{n+1}^\pi(x_{\leq n})$. This definition together with observation 1 directly results in (see supplementary material for proof):

Observation 2 (Conditionals as state-action values)**.** *The target conditional is given by the optimal state-action value function, i.e. $\gamma_n^*(x_n|x_{<n}) = \exp(Q_n^*(x_n|x_{<n}) - V_n^*(x_{<n}))$ where the normalizer is given by $V_n^*(x_{<n}) = \log \sum_{x_n} \exp Q_n^*(x_n|x_{<n})$. Furthermore, the optimal state-action values obey the* soft Bellman equation*:*

$$Q_n^*(x_n|x_{<n}) = R_n(x_{\leq n}) + \log \sum_{x_{n+1}=1}^K \exp\, Q_{n+1}^*(x_{n+1}|x_{\leq n}). \tag{4}$$

3.2 TREESAMPLE Algorithm

In principle, the soft-Bellman equation 4 can be solved by backwards dynamic programming in the following way. We can represent the problem as a K-ary tree $\mathcal{T}^*$ over nodes corresponding to all partial configurations $\bigcup_{n=0}^N \mathbb{X}_{\leq n}$, root $\varnothing$ and each node $x_{<n}$ being the parent of K children $x_{<n} \circ 1$ to $x_{<n} \circ K$. One can compute all Q-values by starting from all K^N leafs $x_{\leq N} \in \mathbb{X}_N$ for which we can compute the state-action values $Q_N^*(x_N|x_{<N}) = R_N(x)$ and solve eqn. 4 in reverse order. Furthermore, a simple softmax operation on each Q_n^* yields the target conditional $\gamma_n^*(x_n|x_{<n})$. Unfortunately, this requires exhaustive evaluation of all factors.

As an alternative to exhaustive evaluation, we propose the TREESAMPLE algorithm for approximate inference. The main idea is to construct an approximation P_X consisting of a *partial tree* $\mathcal{T} \subseteq \mathcal{T}^*$ and approximate state-actions values Q with support on $\mathcal{T}$. A node in $\mathcal{T}$ at depth n corresponds to a prefix $x_{<n}$, with the attached vector of state-action values $Q_n(\cdot|x_{<n}) = (Q_n(x_n = 1|x_{<n}), \dots, Q_n(x_n = K|x_{<n})) \approx Q^*(\cdot|x_{<n})$ for its K children $x_{<n} \circ 1$ to $x_{<n} \circ K$ (which might not be in tree themselves). Sampling

from P_X is defined in algorithm 1: The tree is traversed from the root $\varnothing$ and at each node, a child is sampled from the softmax distribution defined by Q. If at any point, a node $x_{\leq n}$ is reached that is not in $\mathcal{T}$, the algorithm falls back to a distribution defined by a user-specified, default state-action value function Q^ϕ; we will also refer to Q^ϕ as prior state-action value function as it assigns values before any evaluation of the reward. Later, we will discuss using learned, parametric functions for Q^ϕ. In the following we describe how the partial tree $\mathcal{T}$ is constructed using a given budget of B of evaluations of the factors ψ_m.

3.2.1 Tree Construction with Soft-Bellman MCTS

TREESAMPLE leverages the correspondence of approximate inference and decision-making that we have discussed above. It consists of an MCTS-like algorithm to iteratively construct the tree $\mathcal{T}$ underlying the approximation P_X. Given a partially-built tree $\mathcal{T}$, the tree is expanded (if budget is still available) using a heuristic inspired by Upper Confidence Bound (UCB) methods (Auer et al., 2002). It aims to expand the tree at branches expected to have large contributions to the probability mass by taking into account how important a branch currently seems, given by its current Q-value estimates, as well as a measure of uncertainty of this estimate. The latter is approximated by a computationally cheap heuristic based on the visit counts of the branch, i.e. how many reward evaluations have been made in this branch. The procedure prefers to explore branches with high Q-values and high uncertainty (low visit counts). The pseudo code of TREESAMPLE is given in full in the supplementary material, but is briefly summarized in the following.

Each node $x_{<n}$ in $\mathcal{T}$, in addition to $Q_n(\cdot|x_{<n})$, also keeps track of its visit count $\eta(x_{<n}) \in \mathbb{N}$ and the cached reward evaluation $R_{n-1}(x_{<n})$. For a single tree expansion, $\mathcal{T}$ is traversed from the root by choosing at each intermediate node $x_{<n}$ the next action $a \in \{1\ldots, K\}$ in the following way:

$$\begin{aligned}\arg\max_{a\in\{1,\ldots,K\}} \quad & Q_n(a|x_{<n}) + \\ & c\cdot\max(Q^\phi_n(a|x_{<n}),\epsilon)\frac{\eta(x_{<n})^{1/2}}{1+\eta(x_{<n}\circ a)}. \qquad (5)\end{aligned}$$

Here, the hyperparameters $c > 0$ and $\epsilon > 0$ determine the influence of the second term, which can be seen as a form of exploration bonus and which is computed from the inverse visit count of the action a relative to the visit counts of the parent. This rule is inspired by the PUCT variant employed in (Silver et al., 2016), but using the default value Q^ϕ for the exploration bonus. When a new node $x^{new} \notin \mathcal{T}$ at depth n is reached, the reward function $R_n(x^{new})$ is evaluated, decreasing our budget B. The result is cached and the node is added $\mathcal{T} \leftarrow \mathcal{T} \cup x^{new}$ using Q^ϕ to initialize $Q_{n+1}(\cdot|x^{new})$. Then the Q-values are updated: On the path of the tree-traversal that led to x^{new}, the values are back-upped in reverse order using the soft-Bellman equation. This constitutes the main difference to standard MCTS methods, which employ max- or averaging backups. This reflects the difference of sampling / integration to the usual application of MCTS to maximization / optimization problems. Once the entire budget is spent, $\mathcal{T}$ with its tree-structured Q is returned.

3.2.2 Consistency

As argued above, the exact conditionals $\gamma^*_{n+1}(\cdot|x_{\leq n})$ can be computed by exhaustive search in exponential time. Therefore, a reasonable desideratum for any inference algorithm is that given a large enough budget $B \geq MK^N$ the exact distribution is inferred. In the following we show that TREESAMPLE passes this basic sanity check. The first important property of TREESAMPLE is that a tree $\mathcal{T}$ has the exact conditional $\gamma^*_{n+1}(\cdot|x_{\leq n})$ if the unnormalized target density has been evaluated on all states with prefix $x_{\leq n}$ during tree construction. To make this statement precise, we define $\mathcal{T}_{x_{\leq n}} \subseteq \mathcal{T}$ as the sub-tree of $\mathcal{T}$ consisting of node $x_{\leq n}$ and all its descendants in $\mathcal{T}$. We call a sub-tree $\mathcal{T}_{x_{\leq n}}$ fully expanded, or complete, if all partial states with prefix $x_{\leq n}$ are in $\mathcal{T}_{x_{\leq n}}$. With this definition, we have the following lemma (proof in the supplementary material):

Lemma 1. *Let $\mathcal{T}_{x_{\leq n}}$ be a fully expanded sub-tree of $\mathcal{T}$. Then, for all nodes $x'_{\leq m}$ in $\mathcal{T}_{x_{\leq n}}$, i.e. $m \geq n$ and $x'_{\leq n} = x_{\leq n}$, the state-action values are exact:*

$$Q_{m+1}(\cdot|x'_{\leq m}) \quad = \quad Q^*_{m+1}(\cdot|x'_{\leq m}).$$

Furthermore, constructing the full tree $\mathcal{T}^*$ with TREESAMPLE incurs a cost of at most MK^N evaluations of any of the factors ψ_m, as there are K^N leaf node in $\mathcal{T}^*$ and constructing the path from the root $\varnothing$ to each leaf requires at most M oracle evaluations. This leads to the following result:

Corollary 1 (Exhaustive budget consistency)**.** TREESAMPLE *outputs the correct target distribution P^*_X for budgets $B \geq MK^N$.*

3.3 TREESAMPLE with Learned Parametric Priors

TREESAMPLE explicitly allows for a "prior" Q^ϕ over state-action values with parameters ϕ. It functions as a parametric approximation to $Q^*_n \propto \log\gamma^*_n$. In principle, an appropriate Q^ϕ can guide the search towards regions in $\mathbb{X}$ where probability mass is likely to be found a-priori by the following two mechanisms. It scales the exploration bonus in the PUCT-like decision rule eqn. 5, and it is used to initialize the state-action values Q for a newly expanded node in the search tree.

If Q^ϕ comes from an appropriate function class, it can *transfer* knowledge within the inference problem at hand. Assume we spent some of the available search budget on

TREESAMPLE to build an approximation $\mathcal{T}$. Due to the tree-structure, search budget spent in one branch of the tree does not benefit any other sibling branch. For many problems, there is however structure that would allow for *generalizing* knowledge across branches. This can be achieved via Q^ϕ, e.g. one could train Q^ϕ to approximate the Q-values of the current $\mathcal{T}$, and (under the right inductive bias) knowledge would transfer to newly expanded branches. A similar argument can be made for parametric generalization across problem instances. Assume a given a family of distributions $\{P_X^i\}_{i\in I}$ for some index-set I. If the different distributions P_X^i share structure, it is possible to leverage search computations performed on P_X^i for inference in P_X^j to some degree. A natural example for this is posterior inference in the same underlying model conditioned on different evidence / observations, similar e.g. to amortized inference in variational auto-encoders (Kingma and Welling, 2013). Besides transfer, there is a purely computational reason for learning a parametric Q^ϕ. The memory footprint of TREESAMPLE grows linearly with the search budget B. For large problems with large budgets $B \gg 0$, storing the entire search tree in memory might not be feasible. In this case, compiling the current tree periodically into Q^ϕ and rebuilding it from scratch under prior Q^ϕ and subsequent refinement using TREESAMPLE may be preferable.

Concretely, we propose to train Q^ϕ by regression on state-action values Q generated by TREESAMPLE. For generalization across branches, Q approximates directly the distribution of interest; for transfer across distributions, Q approximates the source distribution, and we apply the trained Q^ϕ for inference search in a different target distribution. We match Q^ϕ to Q by minimizing the expected difference of the values $\mathbb{E}_{P_X}[\sum_n \|Q_n^\phi(\cdot|X_{<n}) - Q_n(\cdot|X_{<n})\|_2^2]$ wrt. ϕ.

4 Experiments

In the following, we empirically compare TREESAMPLE to other baseline inference methods on different families of distributions. We quantify approximation error by the Kullback-Leibler divergence:

$$D_{\mathrm{KL}}[P_X \| P_X^*] = \log Z - \mathbb{E}_{P_X}\left[\sum_{m=1}^{M} \psi_m(X)\right] - \mathbb{H}\left[P_X\right],$$

where we refer to the second term as negative expected energy, and the last term is the entropy of the approximation. We can get unbiased estimates of these using samples from P_X. For intractable target distributions, we compare different inference methods using $\Delta D_{\mathrm{KL}} := D_{\mathrm{KL}} - \log Z$, which is tractable to compute and preserves ranking of different approximation methods.

As baselines we consider the following: Sequential Importance Sampling (SIS), Sequential Monte Carlo (SMC) and for a subset of the environments also Gibbs sampling (GIBBS) and sampling with loopy belief propagation (BP); details are given in the supplementary material. We use the baseline methods in the following way: We generate a set of particles $\{x^i\}_{i\leq I}$ of size I such that we exhaust the budget B, and then return the (potentially weighted) sum of atoms $\sum_{i\leq I} p^i \delta(x, x^i)$ as the approximation density; here δ is the Kronecker delta, and p^i are either set to $1/I$ for GIBBS, BP and to the self-normalized importance weights for SIS and SMC. Hyperparameters for all methods where tuned individually for different families of distributions on an initial set of experiments and then kept constant across all reported experiments. For further details, see the supplementary material. For SIS and SMC, the proposal distribution plays a comparable role to the state-action prior in TREESAMPLE. Therefore, for all experiments we used the same parametric family for Q^ϕ for TREESAMPLE, SIS and SMC.

For the sake of simplicity, in the experiments we measured and constrained the inference budget B in terms of reward evaluations, i.e. each pointwise evaluate of a R_n incurs a cost of one, instead of factor evaluations.

4.1 TREESAMPLE w/o Parametric Value Function

We first investigated inference without learned parametric Q^ϕ. Instead, we used the simple heuristic of setting $\forall a \forall n\ Q_n^\phi(a|x_{<n}) := (N-n)\log K$, which corresponds to the state-action values when all factors $\psi_m \equiv 0$ vanish everywhere.

Chain Distributions We initially tested the algorithms on inference in chain-structured factor graphs (CHAINS, CH). These allow for exact inference in linear time, and therefore we can get unbiased estimates of the true Kullback-Leibler divergences. We report results averaged over 10^3 different chains of length $N = 10$ with randomly generated unary and binary potential functions; for details, see supplementary material. The number of states per variable was set to $K = 5$, yielding $K^N \approx 10^7$ states in total. The results shown in fig. 1 as a function of the inference budget B, show that TREESAMPLE outperforms the SMC baseline (see also tab. 1). In particular, TREESAMPLE generates approximations of similar quality compared to SMC with a roughly 30 times smaller budget. We further investigated the energy and entropy contributions to D_{KL} separately. We define Δenergy$= -\mathbb{E}_{P_X^*}[\sum\psi] + \mathbb{E}_{P_X}[\sum\psi]$ (lower is better), and Δentropy$= \mathbb{H}[P_X^*] - \mathbb{H}[P_X]$ (higher is better). Fig. 1 shows that TREESAMPLE finds approximations that have lower energy as well as higher entropy compared to SMC.

A known limitation of tree search methods is that they tend to under-perform for shallow (here small N) decision-making problems with large action spaces (here large K). We performed experiments on chain distributions with varying K and N while keeping the state-space size approximately constant, i.e. $N \log K \approx$ const. We confirmed that

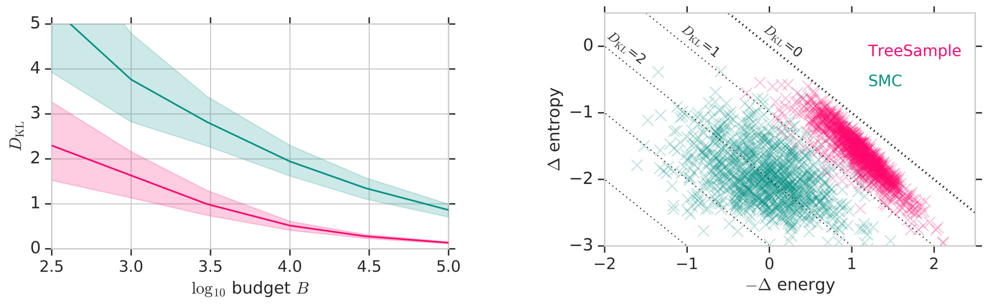

Figure 1: Comparison of TREESAMPLE to SMC on inference in 1000 randomly generated Markov chains. Left: Approximation error as a function of inference budget (log-scale), showing that SCM needs more than 30 times the budget of TREESAMPLE to generate comparable approximations. Right: TREESAMPLE finds approximations with both higher entropy and lower energy.

Table 1: Approximation error (lower is better) for different inference methods on four distribution classes.

	CH	PM-CH	FG1	FG2
SIS	11.61	9.23	-21.97	-31.70
SMC	1.94	7.08	-24.09	-35.90
GIBBS	–	–	-18.67	-25.12
BP	exact	exact	-21.50	-31.48
TS	**0.53**	**3.41**	**-28.89**	**-38.70 9**

for very shallow, bushy problems with $\log K \gg N$, SMC outperforms TREESAMPLE, whereas TREESAMPLE dominates SMC in all other problem configurations, see figure in supplementary material.

Next, we considered chain-structured distributions where the indices of the variables X_n do not correspond to the ordering in the chain; we call these PERMUTEDCHAINS (PM-CH). These are in general more difficult to solve as they exhibit "delayed" rewards, i.e. binary chain potentials $\psi_m(X_{\sigma(n)}, X_{\sigma(n+1)})$ depend on non-consecutive variables. This can create "dead-end" like situations, that SMC, not having the ability to backtrack, can get easily stuck in. Indeed, we find that SCM performs only somewhat better on this class of distributions than SIS, whereas TREESAMPLE achieves better results by a wide margin. Results on both families of distributions are shown in tab. 1.

Factor Graphs We also tested the inference algorithms on two classes of non-chain factor graphs, denoted as FACTORGRAPHS1 (FG1) and FACTORGRAPHS2 (FG2). Distributions in FACTORGRAPHS1 are over $N = 10$ variables with $K = 5$ states each. Factors were randomly generated with maximum degree d of 4 and their d^K values where iid drawn from $\mathcal{N}(0, 1)$. Distributions in FACTORGRAPHS2 are over $N = 20$ binary variables, i.e. $K = 2$. These distributions are generated by two types of of factors: NOT (degree 2) and MAJORITY (max degree 4), both taking values in $\{0, 1\}$. Results are shown in tab. 1. For both families of distributions, TREESAMPLE outperforms all considered baselines by a wide margin. We found that GIBBS generally failed to find configurations with high energy due to slow mixing. BP-based sampling was observed to generate samples with high energy but small entropy, yielding results comparable to SIS.

4.2 TREESAMPLE with Parametric Value Functions

Next, we investigated the performance of TREESAMPLE, as well as SMC, with additional parametric state-action value functions Q^ϕ (used as proposal for SMC). We focused on inference problems from FACTORGRAPHS2. We implemented the inference algorithm as a distributed architecture consisting of a *worker* and a *learner* process, both running simultaneously. The worker requests an inference problem instance, and performs inference either with TREESAMPLE or SMC with a small budget of $B = 2500$ using the current parametric Q^ϕ. After building the approximation P_X, 128 independent samples $x^i \sim P_X$ are drawn from it and the inferred Q-values $Q_n(\cdot|x^i_{\leq n})$ for $i = 1, \ldots, 128$ and $n = 1, \ldots, N$ are written into a replay buffer as training data; following this, the inference episode is terminated, the tree is flushed and a new episode starts. The learner process samples data from the replay as training data for updating the parametric Q^ϕ with an SGD step on a minibatch of size 128; then the updated model parameters ϕ are sent to the worker. We tracked the error of the inference performed on the worker using the unnormalized ΔD_{KL} as a function of the number of completed inference episodes. We expect ΔD_{KL} to decrease, as Q^ϕ adapts to the inference problem, and therefore becomes better at guiding the search. Separately, we also track the inference performance of only using the value function Q^ϕ without additional search around it, denoted as $\Delta D^\phi_{\mathrm{KL}}$. This is a purely parametric approximation to the inference problem, trained by samples from TREESAMPLE and SMC respectively. We observed that ΔD_{KL} as well as $\Delta D^\phi_{\mathrm{KL}}$ stabilized after roughly 1500 inference episodes for all experiments. Results were then averaged over episodes 2000-4000 and are shown in tab. 2.

Table 2: Approximation error for inference in factor graphs for different types of value functions and training regimes.

value func.	$\varnothing$	MLP	GNN				
single graph	N/A	Yes	Yes	No	No	No	No
N_{train}	N/A	20	20	20	12	16	24
Q^ϕ trained by SMC: $\Delta D^\phi_{\text{KL}}$	–	+1.63	-0.19	-0.97	-1.00	-1.17	-0.64
Q^ϕ + SMC: ΔD_{KL}	+2.72	+2.93	+1.64	+2.56	+2.00	+1.64	+2.10
Q^ϕ trained by TREESAMPLE: $\Delta D^\phi_{\text{KL}}$	–	-3.61	-3.86	-2.05	-2.12	-2.52	-1.83
Q^ϕ + TREESAMPLE: ΔD_{KL}	**0.00**	**-3.63**	**-3.87**	**-2.23**	**-2.22**	**-2.64**	**-2.35**

Results in tab. 2 are medians over 20 runs and, to facilitate comparison, are reported relative to ΔD_{KL} for TREESAMPLE without value functions.

We first performed a simple set of "sanity-check" experiments on TREESAMPLE with parametric value functions in a non-transfer setting, where the worker repeatedly solves the same inference problem arising from a single factor graph. As value function, we used a simple MLP with 4-layers and 256 hidden units each. As shown in the second column of tab. 2, approximation error ΔD_{KL} decreases significantly compared to plain TREESAMPLE without value functions. This corroborates that the value function can indeed cache part of the previous search trees and facilitate inference if training and testing factor graphs coincide. Furthermore, we observed that once Q^ϕ is fully trained, the inference error $\Delta D^\phi_{\text{KL}}$ obtaind using only Q^ϕ is only marginally worse than ΔD_{KL} using Q^ϕ plus TREESAMPLE-search on top of it; see row four and five in tab. 2 respectively. This indicates that the value function was powerful enough in this experiment to almost cache the entire search computation of TREESAMPLE.

Next, we investigated graph neural networks (GNNs) (Battaglia et al., 2018) as value functions Q^ϕ. This function class can make explicit use of the structure of the factor graph instances. Details about the architecture can be found in (Battaglia et al., 2018) and the supplementary material, but are briefly described in the following. GNNs consist of two types of networks, node blocks and edge blocks (we did not use global networks), that are connected according to the factor graph at hand, and executed multiple times mimicking a message-passing like procedure. We used three node block networks, one for each type of graph node, i.e. variable node (corresponding to a variable X_n), NOT-factors and MAJORITY-factors. We used four edge block networks, namely one for each combination of {incoming,outgoing}$\times${NOT, MAJORITY}. Empirically, we found that GNNs slightly outperform MLPs in the non-transfer setting, see third column of tab. 2. The real advantage of GNNs comes into play in a transfer setting, when the worker performs inference in a new factor graph for each episode. We keep the number of variables fixed ($N_{\text{train}} = N_{\text{test}} = 20$) but vary the number and configuration of factors across problems. GNNs successfully generalize across graphs, see fourth column of tab. 2. This is due to their ability to make use of the graph topology of a new factor graph instance, by connecting its constituent node and edge networks accordingly. Furthermore, the node and edge networks evidently learned generic message passing computations for variable nodes as well as NOT/MAJORITY factor nodes. The results show that a suitable Q^ϕ generalizes knowledge across inference problems, leading to less approximation error on new distributions. Furthermore, we investigated a transfer setting where the worker solves inference problems on factor graphs of sizes $N_{\text{train}} = 12, 16$ or 24, but performance is tested on graphs of size $N_{\text{test}} = 20$; see columns five to seven in tab. 2. Strikingly, we find that the value functions generalize as well across problems of different sizes as they generalize across problems of the same size. This demonstrates that prior knowledge can successfully guide the search and greatly facilitate inference.

Finally, we investigated the performance of SMC with trained value functions Q^ϕ; see rows one and two in tab. 2. Overall, we found that performance was worse compared to TREESAMPLE: Value functions Q^ϕ trained by SMC were found to give worse results $\Delta D^\phi_{\text{KL}}$ compared to those trained by TREESAMPLE, and overall inference error was worse compared to TREESAMPLE. Interestingly, we found that once Q^ϕ is fully trained, performing additional SMC on top of it made results worse. Although initially counterintuitive, these results are sensible in our problem setup. The entropy of SMC approximations $\approx \log I$ is essentially given by the number of particles I that SMC produces; this number is limited by the budget B that can be used to compute importance weights. Once a parametric Q^ϕ is trained, it does not need to make any further calls to the ψ_m factors, and can therefore exhibit much higher entropy, therefore making $\Delta D^\phi_{\text{KL}}$ smaller than ΔD_{KL}.

5 Related Work

TREESAMPLE is based on the connection between probabilistic inference and maximum-entropy decision-making problems established by previous work. This connection has mostly been used to solve RL problems with inference methods e.g. (Attias, 2003; Hoffman et al., 2007). Closely related to our approach, this relationship has also been used in the reverse direction, i.e. to solve inference problems using tools from RL (Mnih and Gregor, 2014; Weber et al., 2015; Wingate and Weber, 2013; Schulman et al., 2015; Weber et al., 2019), however without utilizing tree search and emphasizing the importance of exploration for inference. The latter has been recognized in (Lu et al., 2018), and applied to hierarchical partitioning for inference in continuous spaces, see also (Rainforth et al., 2018). In contrast to this, we focus on discrete domains with sequential decision-making utilizing MCTS and value functions.

For approximating general probabilistic inference problems, Markov Chain Monte Carlo (MCMC) has proven very successful in practice. MCMC operates on a fully specified, approximate sample which is perturbed iteratively by a transition operator. The latter is usually designed specifically for a family of distributions to leverage problem structure for achieving fast mixing. However, mixing times are difficult to analyze theoretically and hard to monitor in practice (Cowles and Carlin, 1996). TREESAMPLE circumvents the mixing problem by generating a new sample "from scratch" when returning to the root node and then iteratively stepping through the dimensions of the random vector. Furthermore, TREESAMPLE can make use of powerful neural networks for approximating conditionals of the target, thus caching computations for related inference problems. Although, adaptive MCMC methods exist, they usually only consider small sets of adaptive parameters (Andrieu and Thoms, 2008). Recently, MCMC methods have been extended to transition operators generated by neural networks, which are trained either by adversarial training, meta learning or mixing time criteria (Song et al., 2017; Levy et al., 2017; Neklyudov et al., 2018; Wang et al., 2018). However, these were formulated for continuous domains and rely on differentiability and thus do not carry over directly to discrete domains.

Our proposed algorithm is closely related to Sequential Monte Carlo (SMC) methods (Del Moral et al., 2006), another class of broadly applicable inference algorithms. Often, these methods are applied to generate approximate samples by sequentially sampling the dimensions of a random vector, e.g. in particle filtering for temporal inference problems (Doucet and Johansen, 2009). Usually, these methods do not allow for backtracking, i.e. re-visiting previously discarded partial configurations, although few variants with some back-tracking heuristics do exist (Klepal et al., 2008; Grassberger, 2004). In contrast, the TREESAMPLE algorithm decides at every iteration where to expand the current tree based on a full tree-traversal from the root and therefore allows for backtracking an arbitrary number of steps. Furthermore, we propose to train value functions which approximately marginalize over the "future" (i.e. variables following the one in question in the ordering), thus taking into account relevant downstream effects. Gu et al. (2015); Kempinska and Shawe-Taylor (2017) introduce adaptive NN proposals, i.e. value functions in our formulation, but these are trained to match the "filtering" distribution, thus they do not marginalize over the future. In the decision-making formulation, this corresponds to learning proposals based on immediate rewards instead of total returns. However, recent work in continuous domains has begun to address this (Guarniero et al., 2017; Heng et al., 2017; Lawson et al., 2018; Piché et al., 2018), however, they do not make use of guided systematic search.

Furthermore, Weighted Model Counting (WMC) is an exact inference method related to TREESAMPLE. The target probability distribution is represented as Boolean formulas with associated weights, and inference is carried out by summing weights over satisfying assignments of the associated SAT problem (Chavira and Darwiche, 2008). In particular, it has been shown that DPLL-style SAT solvers (Davis et al., 1962) can be extended to exactly solve general discrete inference problems (Sang et al., 2005; Bacchus et al., 2009), often outperforming other standard methods such as the junction tree algorithm (Lauritzen and Spiegelhalter, 1988). Similar to TREESAMPLE, this DPLL-based approach performs inference by search, i.e. it recursively instantiates variables of the SAT problem.

6 Discussion

For sake of simplicity, we assumed in this paper that the computational cost of inference is dominated by evaluations of the factor oracles. This assumption is well justified e.g. in applications, where some factors represent large scale scientific simulators (Baydin et al., 2019), or in modern deep latent variable models, where some factors are given by deep neural networks that take potentially high-dimensional observations as inputs. If this assumption is violated, i.e. all factors can be evaluated cheaply, the comparison of TREESAMPLE to SMC and other inference methods will become less favourable for the former. TREESAMPLE incurs an overhead for traversing a search tree before expanding it, attempting to use the information of all previous oracle evaluations. If these are cheap, a less sequential and more parallel approach, such as SMC, might become more competitive.

We expect that TREESAMPLE can be improved and extended in many ways. Currently, the topology of the factor graph is only partially used for the reward definition and potentially for graph net value functions. One obvious way to better leverage it would be to check if after conditioning on a prefix $x_{<n}$, corresponding to a search depth n, the factor graph decomposes into independent components that can be solved independently. Furthermore, TREESAMPLE uses a fixed ordering of the variables. However, a good variable ordering can potentially make the inference problem much easier. Leveraging existing or developing new heuristics for a problem-dependent and dynamic variable ordering could potentially increase the inference efficiency of TREESAMPLE.

References

Christophe Andrieu and Johannes Thoms. A tutorial on adaptive MCMC. *Statistics and computing*, 18(4):343–373, 2008.

Hagai Attias. Planning by probabilistic inference. In *AISTATS*, 2003.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Solving # SAT and Bayesian inference with backtracking search. *Journal of Artificial Intelligence Research*, 34: 391–442, 2009.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Atılım Güneş Baydin, Lei Shao, Wahid Bhimji, Lukas Heinrich, Lawrence Meadows, Jialin Liu, Andreas Munk, Saeid Naderiparizi, Bradley Gram-Hansen, Gilles Louppe, et al. Etalumis: Bringing Probabilistic Programming to Scientific Simulators at Scale. *arXiv preprint arXiv:1907.03382*, 2019.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

Mary Kathryn Cowles and Bradley P Carlin. Markov chain Monte Carlo convergence diagnostics: a comparative review. *Journal of the American Statistical Association*, 91(434):883–904, 1996.

Adnan Darwiche. A logical approach to factoring belief networks. *KR*, 2:409–420, 2002.

Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

Peter Dayan and Geoffrey E Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68 (3):411–436, 2006.

Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of nonlinear filtering*, 12(656-704):3, 2009.

Peter Grassberger. Sequential Monte Carlo methods for protein folding. *arXiv preprint cond-mat/0408571*, 2004.

Shixiang Shane Gu, Zoubin Ghahramani, and Richard E Turner. Neural adaptive sequential Monte Carlo. In *Advances in Neural Information Processing Systems*, pages 2629–2637, 2015.

Pieralberto Guarniero, Adam M Johansen, and Anthony Lee. The iterated auxiliary particle filter. *Journal of the American Statistical Association*, 112(520):1636–1647, 2017.

W Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 1970.

Jeremy Heng, Adrian N Bishop, George Deligiannidis, and Arnaud Doucet. Controlled sequential Monte Carlo. *arXiv preprint arXiv:1708.08396*, 2017.

Matt Hoffman, Arnaud Doucet, Nando de Freitas, and Ajay Jasra. Trans-dimensional MCMC for Bayesian policy learning. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, pages 665–672. Curran Associates Inc., 2007.

Kira Kempinska and John Shawe-Taylor. Adversarial Sequential Monte Carlo. In *Bayesian Deep Learning (NIPS Workshop)*, 2017.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Martin Klepal, Stephane Beauregard, et al. A backtracking particle filter for fusing building plans with PDR displacement estimates. In *2008 5th Workshop on Positioning, Navigation and Communication*, pages 207–212. IEEE, 2008.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988.

Dieterich Lawson, George Tucker, Christian A Naesseth, Chris J Maddison, Ryan P Adams, and Yee Whye Teh. Twisted variational sequential Monte Carlo. In *Bayesian Deep Learning Workshop, NIPS*, 2018.

Daniel Levy, Matthew D Hoffman, and Jascha Sohl-Dickstein. Generalizing Hamiltonian Monte Carlo with neural networks. *arXiv preprint arXiv:1711.09268*, 2017.

Xiaoyu Lu, Tom Rainforth, Yuan Zhou, Jan-Willem van de Meent, and Yee Whye Teh. On Exploration, Exploitation and Learning in Adaptive Importance Sampling. *arXiv preprint arXiv:1810.13296*, 2018.

Vikash Mansinghka, Daniel Roy, Eric Jonas, and Joshua Tenenbaum. Exact and approximate sampling by systematic stochastic search. In *Artificial Intelligence and Statistics*, pages 400–407, 2009.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014.

Radford M Neal et al. MCMC using Hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

Kirill Neklyudov, Pavel Shvechikov, and Dmitry Vetrov. Metropolis-Hastings view on variational inference and adversarial training. *arXiv preprint arXiv:1810.07151*, 2018.

Alexandre Piché, Valentin Thomas, Cyril Ibrahim, Yoshua Bengio, and Chris Pal. Probabilistic planning with sequential Monte Carlo methods. *ICLR 2019*, 2018.

Tom Rainforth, Yuan Zhou, Xiaoyu Lu, Yee Whye Teh, Frank Wood, Hongseok Yang, and Jan-Willem van de Meent. Inference trees: Adaptive inference with exploration. *arXiv preprint arXiv:1806.09550*, 2018.

Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.

Tian Sang, Paul Beame, and Henry Kautz. Solving Bayesian networks by weighted model counting. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, volume 1, pages 475–482. AAAI Press, 2005.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient Estimation Using Stochastic Computation Graphs. *CoRR*, abs/1506.05254, 2015.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484, 2016.

Jiaming Song, Shengjia Zhao, and Stefano Ermon. A-nice-MC: Adversarial training for MCMC. In *Advances in Neural Information Processing Systems*, pages 5140–5150, 2017.

Larry Stockmeyer. On approximation algorithms for # P. *SIAM Journal on Computing*, 14(4):849–861, 1985.

Tongzhou Wang, Yi Wu, Dave Moore, and Stuart J Russell. Meta-learning MCMC proposals. In *Advances in Neural Information Processing Systems*, pages 4146–4156, 2018.

Theophane Weber, Nicolas Heess, S. M. Ali Eslami, John Schulman, David Wingate, and David Silver. Reinforced variational inference. In *In Advances in Neural Information Processing Systems (NIPS) Workshop on Advances in Approximate Bayesian Inference. 2015*, 2015.

Théophane Weber, Nicolas Heess, Lars Buesing, and David Silver. Credit Assignment Techniques in Stochastic Computation Graphs. *CoRR*, abs/1901.01761, 2019.

David Wingate and Theophane Weber. Automated variational inference in probabilistic programming. *arXiv preprint arXiv:1301.1299*, 2013.