Supplementary Material to: Approximate Inference in Discrete Distributions with Monte Carlo Tree Search and Value Functions

Lars Buesing, Theophane Weber, Nicolas Heess

A Details for TREESAMPLE algorithm

The complete pseudo code for TREESAMPLE is given in algorithm 1. We define a search tree $\mathcal{T}$ in the following way. Nodes in $\mathcal{T}$ at depth $n \in \{0, 1, \ldots, N\}$ are indexed by the (partial) state $x \in \{1, \ldots, K\}^n$, and the root is denoted by $\varnothing$. Each node x at depth $n = len(x)$ keeps track of the corresponding reward evaluation $R_n(x)$ and the following quantities for all its children:

1. visit counts $\eta_{n+1}(\cdot|x) = (\eta_{n+1}(1|x), \ldots, \eta_{n+1}(K|x)) \in \mathbb{N}^K$ over the children,
2. state-action values $Q_{n+1}(\cdot|x) \in \mathbb{R}^K$,
3. prior state-action values $Q^{\phi}_{n+1}(\cdot|x) \in \mathbb{R}^K$, and
4. a boolean vector $C_{n+1}(\cdot|x) \in \{0, 1\}^K$ if its children are complete (i.e. fully expanded, see below).

Standard MCTS with (P)UCT-style tree traversals applied to the inference problem can in general visit any state-action pair multiple times; this is desirable behavior in general MDPs with stochastic rewards, where reliable reward estimates require multiple samples. However, the reward R in our MDP is deterministic as defined in eqn. 2, and therefore there is no benefit in re-visiting fully-expanded sub-trees. To prevent the TREESAMPLE algorithm from doing so, we explicitly keep track at each node if the sub-tree rooted in it is fully-expanded; such a node is called complete. Initially no internal node is complete, only leaf nodes at depth N are tagged as complete. In the backup stage of the tree-traversal, we tag a visited node as complete if it is a node of depth N (corresponding to a completed sample) or if all its children are complete. We modify the action selection eqn. 5 such that the $\arg\max$ is only taken over actions not leading to completed sub-trees.

B Proofs

B.1 Observation 2

Proof. This observation has been proven previously in the literature, but we will give a short proof here for completeness. We show the statement by determining the optimal policy and value function by backwards dynamic programming (DP). We anchor the DP induction by defining the optimal value function at step $N + 1$ as zero, i.e. $V^*_{N+1}(x_{\leq N}) = 0$. Using the law of iterated expectations, we can decompose the optimal value function in the following way for any $n = N, \ldots, 1$:

$$\begin{aligned}
V_n^*(x_{<n}) &= \max_{\pi_n,\ldots,\pi_N} \mathbb{E}_{P^\pi_{X_{\geq n}|x_{<n}}} \left[\sum_{n'=n}^{N} R_{n'} - \log \pi_{n'} \right] \\
&= \max_{\pi_n} \mathbb{E}_{P^\pi_{X_n|x_{<n}}} \max_{\pi_{n+1},\ldots,\pi_N} \mathbb{E}_{P^\pi_{X_{>n}|x_n \circ X_n}} \left[\sum_{n'=n}^{N} R_{n'} - \log \pi_{n'} \right] \\
&= \max_{\pi_n} \mathbb{E}_{P^\pi_{X_n|x_{<n}}} \left[R_n - \log \pi_n + \max_{\pi_{n+1},\ldots,\pi_N} \mathbb{E}_{P^\pi_{X_{>n}|x_n \circ X_n}} \left[\sum_{n'=n+1}^{N} R_{n'} - \log \pi_{n'} \right] \right] \\
&= \max_{\pi_n} \mathbb{E}_{P^\pi_{X_n|x_{<n}}} \left[R_n - \log \pi_n + V_{n+1}^* \right].
\end{aligned}$$

Therefore, assuming by induction that V_{n+1}^* has been computed, we can find the optimal policy π_n^* and value V_n^* at step n by solving:

$$\begin{aligned}
&\underset{f:\{1\ldots,K\}\to[0,1]}{\arg\max} && \sum_{a=1}^{K} f(a) \cdot \left(R_n(x_{<n} \circ a) - \log f(a) + V_{n+1}^*(x_{<n} \circ a) \right) && \text{(1a)} \\
&\text{subject to} && \sum_{a=1}^{K} f(a) = 1, \\
& && f(a) \geq 0, \ \ \forall a \in \{1, \ldots, K\}.
\end{aligned}$$

The solution to this optimization problem can be found by the calculus of variations (omitted here) and is given by:

$$\log \pi^*(x_n|x_{<n}) \quad \propto \quad R_n(x_{\leq n}) + V_{n+1}^*(x_{\leq n}) = Q_n^*(x_n|x_{<n}),$$

where we used the definition of the optimal state-action value function. Furthermore, at the optimum, the objective eqn. 1a assumes the value:

$$V_n^*(x_{<n}) \quad = \quad \log \sum_{x_n=1}^{K} \exp Q_n^*(x_n|x_{<n}).$$

This expression, together with the definition of Q^* establishes the soft-Bellman equation. The optimal value V_{n+1}^* is also exactly the log-normalizer for π_n^*. Therefore, we can write:

$$\log \pi_n^*(x_n|x_{<n}) = Q_n^*(x_n|x_{<n}) - V_{n+1}^*(x_{\leq n}).$$

□

B.2 Proof of Lemma 1

Proof. We will show this statement by induction over the depth $d := N - n$ of the sub-tree $\mathcal{T}_{x_{\leq n}}$ with root $x_{\leq n}$. For $d = 0$, i.e. $n = N$, the state-action values $Q_{N+1}(\cdot|x_{\leq N}) := -\log K$ are defined such that $V_{N+1}(x_{\leq N}) = 0$, which is the correct value. Consider now the general case $1 \leq d \leq N$. Let $\mathcal{T}'_{x_{\leq n}}$ be the sub-tree *before* the last tree traversal that expanded the last missing node $x_{\leq N}$, ie $\mathcal{T}_{x_{\leq n}} = \mathcal{T}'_{x_{\leq n}} \cup x_{\leq N}$; for an illustration see fig. 1. The soft-Bellman backups of the last completing tree-traversal on the path leading to $x_{\leq N}$ are by construction all of the following form: For any node $x_{\leq m}$ on the path, all children except for one correspond to already completed sub-trees (before the last traversal). The sub-tree of the one remaining child is completed by the last traversal. All complete sub-trees on the backup path are of depth smaller than d and therefore by induction their roots have the correct values $V_{m+1}^*(x_{\leq m})$. Hence evaluating the soft-Bellman backup eqn. 4 (with the true noiseless reward R) yields the correct value for $x_{\leq n}$. □

C Details for Experiments

C.1 Baseline Inference Methods

C.1.1 SIS and SMC

For each experiment we determined the number of SIS and SMC particles I such that the entire budget B was used. We implemented SMC with an resampling threshold $t \in [0, 1]$, i.e. a resampling step was executed when the effective sample size (ESS) was smaller than tI. The threshold t was treated as a hyperparameter; SMC with $t = 0$ was used as SIS results.

C.1.2 BP

We used the algorithm outline on p. 301 from Mezard et al. [2009]. For generating a single approximate sample from the target, the following procedure was executed. Messages from variable to factor nodes were initialized as uniform; then $N_{message}$ message-passing steps, each consisting of updating factor-variable and variable-factor messages were performed. X_1 was then sampled form the resulting approximate marginal, and the messages from X_1 to its neighboring factors were set to the corresponding atom. This was repeated until all variables X_n were sampled, generating one approximate sample from the joint P^*_X.

In total, we generated multiple samples with the above algorithm such that the budget B was exhausted. The number $N_{message}$ of message-passing steps before sampling each variable $X_n|X_{<n}$ was treated as a hyperparameter.

C.1.3 GIBBS

We implemented standard Gibbs sampling. All variables were initially drawn uniformly from $\{1, \ldots, K\}$, and N_{Gibbs} iterations, each consisting of updating all variables in the fixed order X_1 to X_N, were executed. This generated a single approximate sample. We repeated this procedure to generate multiple samples such that the budget B was exhausted. We treated N_{Gibbs} as a hyperparameter.

C.2 Hyperparameters optimization

For each inference method (except for SIS) we optimized one hyperparameter on a initial set of experiments. For TREESAMPLE, we fixed $\epsilon = 0.1$ and optimized c from eqn. 4. Different hyperparameter values were used for different families of distributions. Hyperparameters were chosen such as to yield lowest ΔD_{KL}.

C.3 Details for Synthetic Distributions

C.3.1 Chains

The unary potentials $\psi_n(x_n)$ for $n = 1, \ldots, N$ for the chain factor graphs where randomly generated in the following way. The values of $\psi_n(x_n = k)$ for $n = 1, \ldots, N$ and $k = 1, \ldots, K$ where jointly drawn from a GP over the two dimensional domain $\{1, \ldots, N\} \times \{1, \ldots, K\}$ with an RBF kernel with bandwidth 1 and scale 0.5. Binary potentials $\psi_{n,n+1}(x_n, x_{n+1})$ were set to $2.5 \cdot d(x_n, x_{n+1})$, where $d(x_n, x_{n+1})$ is the distance between x_n and x_{n+1} on the 1-d torus generated by constraining 1 and K to be neighbors.

C.3.2 PermutedChains

We first uniformly drew random permutations $\sigma : \{1, \ldots, N\} \to \{1, \ldots, N\}$. We then randomly generated conditional probability tables for $P^*_{X_{\sigma(n)}|x_{\sigma(n-1)}}$ by draws from a symmetric Dirichlet with concentration parameter $\alpha = 1$. These were then used as binary factors.

C.3.3 FactorGraphs1

We generated factor graphs for this family in the following way. First, we constructed Erdős-Rényi random graphs with N nodes with edge probability $p = 2\log(N)/N$; graphs with more than one connected component were rejected. For each clique in this graph we inserted a random factor and connected it to all nodes in the clique; graphs with cliques of size > 4 where rejected.

For applying the sequential inference algorithms TREESAMPLE, SIS and SMC, variables in the graph were ordered by a simple heuristic. While iterating over factors in order of descending degree, all variables in the current factor were were added to the ordering until all were accounted for.

C.3.4 FactorGraphs2

We generated factor graphs for this family over binary random variables $K = 2$ in the following way. Variables X_{2n+1} and $X_{2(n+1)}$ for $n = 0, \dots, N/2 - 1$ were connected with a NOT factor, which carries out the computation $XOR(X_{2n+1}, X_{2(n+1)})$. We then constructed Erdős-Rényi random graphs of size $N/2$ over all pairs of nodes $(X_{2n+1}, X_{2(n+1)})$ with edge probability $p = 3\log(N/2)/N$; graphs with more than one connected component were rejected. For each clique in this intermediate graph we inserted a MAJORITY factor and connected it to either to X_{2n+1} or $X_{2(n+1)}$; graphs with cliques of size > 4 where rejected. MAJORITY factors return a value of 1.0 if half or more nodes in its neighborhood are 2 and return 0 otherwise. The output values of all factors were also scaled by 2.0; otherwise the resulting distributions were found to be very close to uniform.

C.4 Details For Experiments w/ Value Functions

All neural networks were trained with the ADAM optimizer Kingma and Ba [2014] with a learning rate of $3 \cdot 10^{-4}$ and mini-batches of size 128. The replay buffer size was set to 10^4.

The MLP value function used for the experiment consisted of 4 hidden layers with 256 units each with RELU activation functions. Increasing the number of units or layers did not improve results.

The GNN value function $Q^\phi(\cdot|x_{\leq n})$ was designed as follows. Each variable node in the factor graph was given a $(16 + K)$-dimensional feature vector, and each edge node a 16-dimensional feature vector. The input prefix $x_{\leq n}$ was encoded in a one-hot manner in the first K components of the variable feature vectors for variables up to n; for variables $> n$ the first K components were set to 0. All edge features were initialized to 0. All node and edge block networks where chosen to be MLPs with 3 hidden layers and 16 units each with RELU activations. Results did not improve with deeper or wider networks. The resulting GNN was iterated 4 times; interestingly more iterations actually reduced final performance somewhat. The output feature vector of the GNN at variable node x_n was then passed to a linear layer with K outputs yielding the vector $(Q^\phi(x_{n+1} = 1|x_{\leq n}), \dots, Q^\phi(x_{n+1} = K|x_{\leq n}))$.

References

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Marc Mezard, Marc Mezard, and Andrea Montanari. *Information, physics, and computation*. Oxford University Press, 2009.

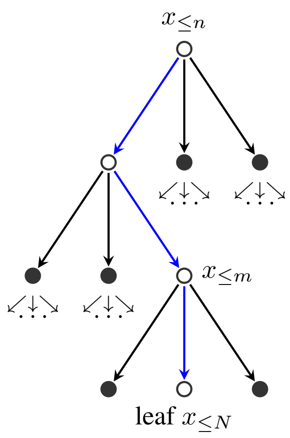

Figure 1: **Completing a sub-tree yields the exact Q-value.** Assume the tree traversal shown in blue completes the sub-tree $\mathcal{T}_{x_{\leq n}}$ rooted in $x_{\leq n}$. Then, by construction, the soft-Bellmann backups along this path, at every intermediate node $x_{\leq m}$ for $m > n$, take as input the values of all children. By construction, all but one children correspond to complete sub-trees of a smaller depth; these have the correct values by induction. The other remaining child corresponds to a sub-tree that was completed by the last traversal and therefore has also the correct value.

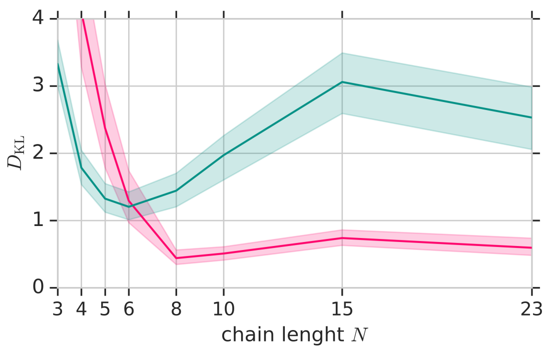

Figure 2: Approximation error for inference in chain graphs as a function of varying chain length N; the number K of states per variable was abjusted such that the total domain size $N \log K$ stayed roughly constant. TREESAMPLE (red) performed worse than SMC (turquoise) for short and wide chains, but performs better everywhere else.

Algorithm 1 TREESAMPLE procedures

```
 1: globals reward function $R$, prior state-action value function $Q^{\phi}$

 2: procedure TREESAMPLE(budget B)
 3:     initialize empty tree $\mathcal{T} \leftarrow \varnothing$
 4:     available budget $b \leftarrow B$
 5:     while $b > M$ do
 6:         $\mathcal{T}, \Delta b \leftarrow$ TREETRAVSERSAL ($\mathcal{T}$)
 7:         $b \leftarrow b - \Delta b$
 8:     end while
 9:     return tree $\mathcal{T}$
10: end procedure

11: procedure TREETRAVERSAL(tree $\mathcal{T}$)
        // traversal
12:     $x \leftarrow \varnothing$
13:     while $x \in \mathcal{T}$ do
14:         $n \leftarrow len(x)$
15:         $a \leftarrow$ Q-UCT$(\eta_{n+1}(\cdot|x), Q_{n+1}(\cdot|x), Q^{\phi}_{n+1}(\cdot|x), C_{n+1}(\cdot|x))$
16:         $x \leftarrow x \circ a$
17:     end while
        // expansion
18:     if $x \notin \mathcal{T}$ then
19:         $\mathcal{T} \leftarrow \mathcal{T} \cup$ EXPAND$(x)$
20:         used budget $\Delta b \leftarrow |M_n|$     // see eqn. 2
21:     end if
        // backup
22:     for $n = len(x), \ldots, 1, 0$ do
23:         $V_{n+1}(x_{\leq n}) = \log \sum_{a'=1}^{K} \exp Q_{n+1}(a'|x_{\leq n})$
24:         $Q_n(x_n|x_{<n}) \leftarrow R_n(x_n|x_{<n}) + V_{n+1}(x_{\leq n})$
25:         $C_n(x_n|x_{<n}) \leftarrow \min_{a'} C_{n+1}(a'|x_{\leq n})$
26:         $\eta_n(x_n|x_{<n}) \leftarrow \eta_n(x_n|x_{<n}) + 1$
27:     end for
28:     return $\mathcal{T}$, $\Delta b$
29: end procedure

30: procedure Q-UCT($\eta_{n+1}(\cdot|x)$, $Q_{n+1}(\cdot|x)$, $Q^{\phi}_{n+1}(\cdot|x)$, $C_{n+1}(\cdot|x)$)
31:     return arg max of eqn. 5 over in-complete children $\{a|C_{n+1}(a|x) = 0\}$
32: end procedure

33: procedure EXPAND(state $x$)
34:     $n \leftarrow len(x)$
35:     evaluate reward function $R_n(x_n|x_{<n})$
36:     initialize $\eta_{n+1}(\cdot|x) \leftarrow (0, \ldots, 0)$
37:     if $n = N$ then    // $x$ is leaf
38:         initialize $Q_{n+1}(a \cdot |x) \leftarrow -\log K$ for all $a \in \{1, \ldots, K\}$
39:         initialize $C_{n+1}(\cdot|x) \leftarrow (1, \ldots, 1)$
40:     else
41:         evaluate prior $Q^{\phi}_{n+1}(\cdot|x)$
42:         initialize $Q_{n+1}(\cdot|x) \leftarrow Q^{\phi}_{n+1}(\cdot|x)$
43:         initialize $C_{n+1}(\cdot|x) \leftarrow (0, \ldots, 0)$
44:     end if
45:     return node $x$ with $\eta_{n+1}(\cdot|x)$, $Q_{n+1}(\cdot|x)$, $Q^{\phi}_{n+1}(\cdot|x)$, $C_{n+1}(\cdot|x)$, $R_n(x_n|x_{<n})$
46: end procedure
```