Journal of Machine Learning Research 21 (2020) 1-36 Submitted 11/19; Published 1/20

Ancestral Gumbel-Top-k Sampling for Sampling Without Replacement

Wouter Kool W.W.M.KOOL@UVA.NL
University of Amsterdam
P.O. Box 19268, 1000GG, Amsterdam, The Netherlands
ORTEC
Houtsingel 5, 2719EA, Zoetermeer, The Netherlands

Herke van Hoof H.C.VANHOOF@UVA.NL
University of Amsterdam

Max Welling M.WELLING@UVA.NL
University of Amsterdam, CIFAR

Editor: Kilian Weinberger

Abstract

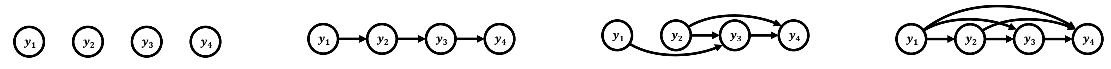
We develop ancestral Gumbel-Top-k sampling: a generic and efficient method for sampling without replacement from discrete-valued Bayesian networks, which includes multivariate discrete distributions, Markov chains and sequence models. The method uses an extension of the Gumbel-Max trick to sample without replacement by finding the top k of perturbed log-probabilities among all possible configurations of a Bayesian network.

Despite the exponentially large domain, the algorithm has a complexity *linear* in the number of variables and sample size k. Our algorithm allows to set the number of parallel processors m, to trade off the number of iterations versus the total cost (iterations times m) of running the algorithm. For $m = 1$ the algorithm has minimum total cost, whereas for $m = k$ the number of iterations is minimized, and the resulting algorithm is known as *Stochastic Beam Search*.[1] We provide extensions of the algorithm and discuss a number of related algorithms.

We analyze the properties of Gumbel-Top-k sampling and compare against alternatives on randomly generated Bayesian networks with different levels of connectivity. In the context of (deep) sequence models, we show its use as a method to generate diverse but high-quality translations and statistical estimates of translation quality and entropy.

Keywords: sampling without replacement, ancestral sampling, bayesian networks, stochastic beam search, gumbel-max trick

1. This paper is an extended version of Kool et al. (2019b) which introduced Stochastic Beam Search.



(a) Independent $p(\boldsymbol{y}) = \prod_v p(y_v)$

(b) Markov chain $p(\boldsymbol{y}) = p(y_1) \prod_{t>1} p(y_t | y_{t-1})$

(c) General network $p(\boldsymbol{y}) = \prod_v p\left(y_v | \boldsymbol{y}_{\text{pa}(v)}\right)$

(d) Sequence model $p(\boldsymbol{y}) = \prod_t p\left(y_t | \boldsymbol{y}_{1:t-1}\right)$

Figure 1: Examples of Bayesian networks.

1. Introduction

Sampling from graphical models is a widely studied problem in machine learning. In many applications, such as neural machine translation, one may desire to obtain *multiple* samples, but wish to avoid duplicates, i.e. *sample without replacement.* In general, this is non-trivial: for example rejection sampling may take long if entropy is low.

In this paper, we extend the idea of sampling through optimization (Papandreou and Yuille, 2011; Hazan and Jaakkola, 2012; Tarlow et al., 2012; Hazan et al., 2013; Ermon et al., 2013; Maddison et al., 2014; Chen and Ghahramani, 2016; Balog et al., 2017) to generate multiple samples *without replacement.* The most well-known example of sampling by optimization is the Gumbel-Max trick (Gumbel, 1954; Maddison et al., 2014), which samples from the categorical distribution by optimizing (i.e. taking the argmax of) log-probabilities perturbed with independent Gumbel noise. Whereas the Gumbel-Max trick only considers the argmax (top 1), taking the *top k* of perturbed log-probabilities actually results in a sample *without replacement* from the categorical distribution (Yellott, 1977; Vieira, 2014).

We consider sampling from discrete-valued Bayesian networks (since sampling without replacement from continuous domains is trivial), which means that we sample from a discrete multivariate distribution which is represented by a probabilistic directed acyclic graphical model (for examples, see Figure 1). By treating each possible configuration of the variables in the network as a category in a single (flat) categorical distribution, we can use 'Gumbel-Top-k sampling' to sample k configurations without replacement. To efficiently sample from the exponentially large domain, we use a top-down sampling procedure (Maddison et al., 2014) combined with an efficient branch-and-bound algorithm, which runs in time *linear* in the number of variables and number of samples.

The algorithm presented in this paper, which we refer to as *ancestral Gumbel-Top-k sampling*, is a generalization of *Stochastic Beam Search* (originally presented in Kool et al. 2019b[2]), which allows to trade off 'parallelizability' against total required computation and is applicable to general Bayesian networks. As such, it serves the same purposes and can be used to generate representative and unique sequences from sequence models, for example for task such as neural machine translation (Sutskever et al., 2014; Bahdanau et al., 2015) and image captioning (Vinyals et al., 2015), where the diversity can be controlled by the sampling temperature. Additionally, being a sampling method, it can be used to construct statistical estimators as we show in Section 5. The ability to sample without replacement enables the construction of lower variance gradient estimators (Kool et al., 2019a, 2020).

2. This paper is an extended version of Kool et al. (2019b). The applicability is extended to general Bayesian Networks. The algorithm is extended to allow trading off parallelism and computational cost and an experiment has been added to evaluate this trade-off. Related algorithms and alternatives are discussed, as well as several possible extensions. The presentation has been improved and extended with graphics.

2. Preliminaries

This section introduces Bayesian Networks, Deep Learning and the Gumbel-Max trick.

2.1. Bayesian Networks

A Bayesian network, also known as *belief network*, is a probabilistic directed acyclic graphical model that represents a joint probability distribution over a set of variables, which are nodes in a directed acyclic graph. We index the variables by $v \in \mathcal{V}$, where a node v can take values $y_v \in D_v$. For an arbitrary subset $\mathcal{S} \subseteq \mathcal{V}$, we write the corresponding set of values as $\boldsymbol{y}_{\mathcal{S}} = (y_v : v \in \mathcal{S})$, with domain $D_{\mathcal{S}} = \prod_{v \in \mathcal{S}} D_v$. For the complete network we write $\boldsymbol{y} = \boldsymbol{y}_{\mathcal{V}}$ with domain $D = D_{\mathcal{V}}$. The probability distribution for y_v is defined conditionally on the parents $\text{pa}(v) \subseteq V \setminus \{v\}$, with values $\boldsymbol{y}_{\text{pa}(v)}$. This way, the distribution $p(\boldsymbol{y})$ is given by

$$p(\boldsymbol{y}) = \prod_{v \in \mathcal{V}} p\left(y_v | \boldsymbol{y}_{\text{pa}(v)}\right). \tag{1}$$

Any directed acyclic graph has at least one *topological* order (see e.g. Kahn 1962), which is an ordering in which each node is preceded by its parent nodes. For an example of a Bayesian network in topological order, see Figure 1c. Examples of Bayesian networks include *multivariate categorical distributions* where $y_v, v \in \mathcal{V}$ are independent (Figure 1a), *Markov chains* where y_t depends only on y_{t-1} (assuming $\mathcal{V} = \{1, ..., T\}$), and (finite length) *sequence models* where y_t depends on the complete 'prefix' $\boldsymbol{y}_{1:t-1}$ and the topological order is natural.

To keep notation compact, in general, we will not make the distinction between variables and their realizations. As each variable y_v has a finite domain D_v, there is a finite number of possible realizations, or *configurations*, for the complete Bayesian network, specified by the domain $D = \prod_{v \in \mathcal{V}} D_v$. A configuration $\boldsymbol{y} \in D$ has probability given by (1) and therefore, ignoring the graphical structure, we can treat $\boldsymbol{y}$ as a 'flat' categorical variable over the domain D, where each category corresponds to a possible configuration $\boldsymbol{y}$.

2.2. Deep Learning

Our focus is on modern Deep Learning (LeCun et al., 2015) applications, especially sampling from models represented as (discrete-valued) Stochastic Computation Graphs (Schulman et al., 2015), which can be considered Bayesian networks. Such models specify conditional distributions for variables using neural networks, dependent on parameters $\boldsymbol{\theta}$ and an input or *context* $\boldsymbol{x}$. In a discrete setting, such models usually output probabilities for all $y_v \in D_V$ in a single *model evaluation*, by computing a *softmax* (with temperature $\tau \geq 0$) over unnormalized log-probabilities $\phi_{\boldsymbol{\theta}}(y_v | \boldsymbol{y}_{\text{pa}(v)}, \boldsymbol{x})$:

$$p_{\boldsymbol{\theta}}(y_v | \boldsymbol{y}_{\text{pa}(v)}, \boldsymbol{x}) = \frac{\exp\left(\phi_{\boldsymbol{\theta}}(y_v | \boldsymbol{y}_{\text{pa}(v)})/\tau\right)}{\sum_{y'_v \in D_v} \exp\left(\phi_{\boldsymbol{\theta}}(y'_v | \boldsymbol{y}_{\text{pa}(v)})/\tau\right)} \quad \forall y_v \in D_v. \tag{2}$$

In Deep Learning, model evaluations involve millions of computations, which is why we seek to minimize them. Additionally, to make efficient use of modern hardware, algorithms should be efficiently *parallelizable*. In the remainder of this paper, we are concerned with sampling (without replacement) given fixed values of $\boldsymbol{\theta}$ and $\boldsymbol{x}$ so we will omit these in the notation.

2.3. The Gumbel-Max Trick

Treating the Bayesian network as specifying a categorical distribution, we can use the Gumbel-Max trick (Gumbel, 1954; Maddison et al., 2014) to sample from it by finding the configuration $\boldsymbol{y}$ with the largest *perturbed log-probability*. We define $\phi_{\boldsymbol{y}}$ as the *log-probability* of $\boldsymbol{y} \in D$:

$$\phi_{\boldsymbol{y}} = \log p(\boldsymbol{y}) = \sum_{v \in \mathcal{V}} \log p\left(y_v | \boldsymbol{y}_{\text{pa}(v)}\right). \tag{3}$$

Next we define $G_{\phi_{\boldsymbol{y}}}$ as the *perturbed log-probability* of $\boldsymbol{y}$, which is obtained by adding (independent!) Gumbel noise $G_{\boldsymbol{y}} \sim \text{Gumbel}(0)$ to $\phi_{\boldsymbol{y}}$, where $\text{Gumbel}(\phi)$ is the Gumbel distribution with CDF

$$F_\phi(g) = \exp(-\exp(\phi - g)). \tag{4}$$

Using inverse transform sampling, this noise is generated as $G_{\boldsymbol{y}} = F_0^{-1}(U_{\boldsymbol{y}}) = -\log(-\log U_{\boldsymbol{y}})$, where $U_{\boldsymbol{y}} \sim \text{Uniform}(0,1)$. By the shifting property of the Gumbel distribution, we have

$$G_{\phi_{\boldsymbol{y}}} := G_{\boldsymbol{y}} + \phi_{\boldsymbol{y}} = \phi_{\boldsymbol{y}} - \log(-\log U_{\boldsymbol{y}}) \sim \text{Gumbel}(\phi_{\boldsymbol{y}}). \tag{5}$$

For any subset $B \subseteq D$ it holds that (Maddison et al., 2014)

$$\max_{\boldsymbol{y} \in B} G_{\phi_{\boldsymbol{y}}} \sim \text{Gumbel}\left(\log \sum_{\boldsymbol{y} \in B} \exp \phi_{\boldsymbol{y}}\right), \tag{6}$$

$$\arg\max_{\boldsymbol{y} \in B} G_{\phi_{\boldsymbol{y}}} \sim \text{Categorical}\left(\frac{\exp \phi_{\boldsymbol{y}}}{\sum_{\boldsymbol{y}' \in B} \exp \phi_{\boldsymbol{y}'}}, \boldsymbol{y} \in B\right). \tag{7}$$

Equation (6) is a useful property that states that the maximum of a set of Gumbel variables is a Gumbel variable with as location the LOGSUMEXP of the individual Gumbel locations. Equation (7) states the most important result: the configuration $\boldsymbol{y}$ corresponding to the largest perturbed log-probability is a sample from the desired categorical distribution (since $\exp \phi_{\boldsymbol{y}} = p(\boldsymbol{y})$). Additionally, the max (6) and argmax (7) are independent, which is an important property that is used in this paper. For details, see Maddison et al. (2014). Figure 2 gives a visual illustration of the Gumbel-Max trick.

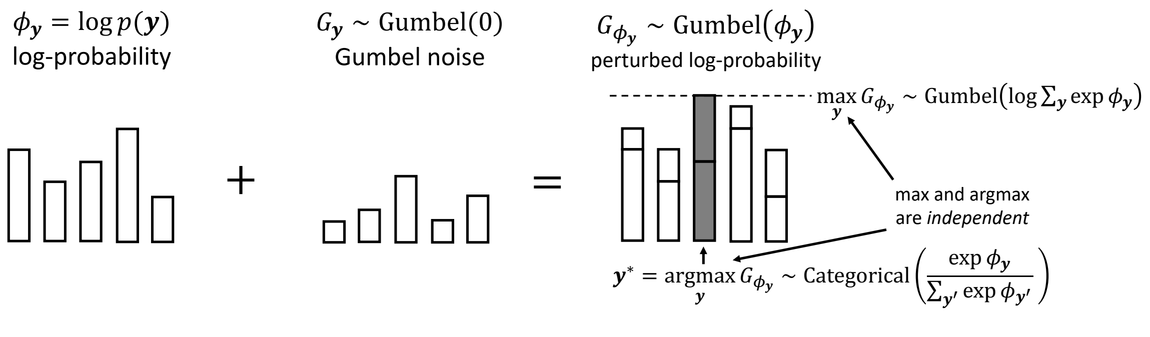

$\phi_y = \log p(\boldsymbol{y})$
log-probability
+
$G_y \sim \text{Gumbel}(0)$
Gumbel noise
=
$G_{\phi_y} \sim \text{Gumbel}(\phi_y)$
perturbed log-probability
$\max_y G_{\phi_y} \sim \text{Gumbel}(\log \sum_y \exp \phi_y)$
max and argmax are *independent*
$y^* = \operatorname{argmax}_y G_{\phi_y} \sim \text{Categorical}\left(\frac{\exp \phi_y}{\sum_{y'} \exp \phi_{y'}}\right)$

Figure 2: The Gumbel-Max trick: the argmax of perturbed log-probabilities has a categorical distribution. The maximum has an independent Gumbel distribution.

2.4. Gumbel-Top-k Sampling

An extension of the Gumbel-Max trick can be used to sample from the categorical distribution without replacement (Yellott, 1977; Vieira, 2014). To this end, let $\boldsymbol{y}_1^*, ..., \boldsymbol{y}_k^* = \arg\operatorname{top} k\, G_{\phi_{\boldsymbol{y}}}$, i.e. $\boldsymbol{y}_1^*, ..., \boldsymbol{y}_k^*$ are the configurations with the k largest perturbed log-probabilities in decreasing order (see Figure 3). Denoting with $D_j^* = D \backslash \{\boldsymbol{y}_1^*, ..., \boldsymbol{y}_{j-1}^*\}$ the domain (without replacement) for the j-th sampled element, the probability for this *ordered sample without replacement* is given by

$$
\begin{aligned}
p\left(\boldsymbol{y}_1^*, ..., \boldsymbol{y}_k^*\right) &= \prod_{j=1}^{k} p\left(\boldsymbol{y}_j^* \middle| \boldsymbol{y}_1^*, ..., \boldsymbol{y}_{j-1}^*\right) && (8)\\
&= \prod_{j=1}^{k} P\left(\boldsymbol{y}_j^* = \arg\max_{\boldsymbol{y} \in D_j^*} G_{\phi_{\boldsymbol{y}}} \middle| \max_{\boldsymbol{y} \in D_j^*} G_{\phi_{\boldsymbol{y}}} < G_{\phi_{\boldsymbol{y}_{j-1}^*}}\right) && (9)\\
&= \prod_{j=1}^{k} P\left(\boldsymbol{y}_j^* = \arg\max_{\boldsymbol{y} \in D_j^*} G_{\phi_{\boldsymbol{y}}}\right) && (10)\\
&= \prod_{j=1}^{k} \frac{\exp \phi_{\boldsymbol{y}_j^*}}{\sum_{\boldsymbol{y} \in D_j^*} \exp \phi_{\boldsymbol{y}}} && (11)\\
&= \prod_{j=1}^{k} \frac{p(\boldsymbol{y}_j^*)}{1 - \sum_{\ell=1}^{j-1} p(\boldsymbol{y}_\ell^*)}. && (12)
\end{aligned}
$$

To understand the derivation, note that conditioning on $\boldsymbol{y}_1^*, ..., \boldsymbol{y}_{j-1}^*$ means that $\boldsymbol{y}_1^*, ..., \boldsymbol{y}_{j-1}^*$ are the configurations with the $j-1$ largest perturbed log-probabilities, so $\boldsymbol{y}_j^*$, the configuration with the j-th largest perturbed log-probability, must be the $\arg\max$ of the remaining log-probabilities. Additionally, we know that $\max_{\boldsymbol{y} \in D_j^*} G_{\phi_{\boldsymbol{y}}}$, the maximum of the remaining log-probabilities, must be smaller than $G_{\phi_{\boldsymbol{y}_{j-1}^*}}$, which is the smallest of the $j-1$ largest perturbed log-probabilities. This allows us to rewrite (8) as (9). The step from (9) to (10) follows from the independence of the max and $\arg\max$ (Section 2.3) and the step from (10) to (11) uses the Gumbel-Max trick.

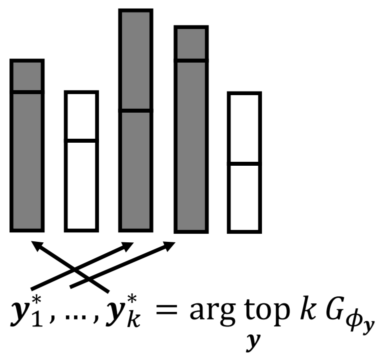

$\boldsymbol{y}_1^*, ..., \boldsymbol{y}_k^* = \arg\operatorname{top}_{\boldsymbol{y}} k\, G_{\phi_{\boldsymbol{y}}}$

Figure 3: Gumbel-Top-k sampling: the top k perturbed log-probabilities are a sample without replacement.

The form (11) is also known as the Plackett-Luce model (Plackett, 1975; Luce, 1959) and the form (12) highlights its interpretation as sampling without replacement, where the probabilities get renormalized after each sampled configuration. We refer to sampling without replacement by taking the top k of Gumbel perturbed log-probabilities as *Gumbel-Top-k sampling*. This is mathematically equivalent to weighted reservoir sampling (Efraimidis and Spirakis, 2006) which can be seen as a streaming implementation of Gumbel-Top-k sampling.

3. Ancestral Gumbel-Top-k Sampling

Ancestral Gumbel-Top-k sampling is an efficient implementation of Gumbel-Top-k sampling for sampling from a probability distribution specified as a Bayesian network. It exploits the graphical structure to sample variables one at a time, conditionally on their parent variables, similar to ancestral sampling. The algorithm finds the top k configurations with largest perturbed log-probabilities (which determine the sample without replacement) *implicitly*, i.e. without sampling perturbed log-probabilities for all possible configurations of the Bayesian network. It does so by bounding the perturbed log-probabilities for parts of the domain, such that they can be pruned from the search if they are guaranteed to not contain a top k perturbed log-probability. Therefore, it can be considered a *branch-and-bound* algorithm (see e.g. Lawler and Wood 1966).

To derive ancestral Gumbel-Top-k sampling, we start with *explicit* Gumbel-Top-k sampling to sample k configurations $\boldsymbol{y}$ without replacement from the Bayesian network. This requires instantiating *all* configurations $\boldsymbol{y} \in D$, sampling their perturbed log-probabilities and finding the k largest to obtain the sample. See for an example Figure 4, where the leaf nodes represent all 8 possible configurations for a Bayesian network with 3 binary variables. As the size of the domain D is exponential in the number of variables $|\mathcal{V}|$, this is not practically feasible in general, and we will derive ancestral Gumbel-Top-k sampling as an alternative, equivalent, sampling procedure that does *not* require to instantiate the complete domain. Instead, it uses a *top-down* sampling procedure to sample perturbed log-probabilities for *partial configurations* $\boldsymbol{y}_{\mathcal{S}}$ (internal nodes in Figure 4) first, which is equivalent but allows to obtain a bound on the perturbed log-probabilities of completions of $\boldsymbol{y}_{\mathcal{S}}$ (descendants of $\boldsymbol{y}_{\mathcal{S}}$ in the tree). This allows the dashed parts of the tree in Figure 4 to be pruned from the search, while obtaining the same result.

3.1. The Probability Tree Diagram

To help develop our theory, we will first assume a *fixed* topological order of the nodes $\mathcal{V} = \{1, ..., T\}$. For $t \leq T$, we will use the notation $\boldsymbol{y}_{1:t} = \boldsymbol{y}_{\{1,...,t\}} = (y_1, ..., y_t)$ to indicate a *prefix* of $\boldsymbol{y}$, which is a *partial configuration* of the Bayesian network with domain $D_{1:t} = \prod_{v \in \{1,...,t\}} D_v$. This topological order allows us to represent possible configurations of the Bayesian network as the *tree diagram* in Figure 4, where level t has $|D_{1:t}|$ nodes: one node for each possible partial configuration $\boldsymbol{y}_{1:t} \in D_{1:t}$. Given a partial configuration $\boldsymbol{y}_{1:t}$ of the nodes $\{1, ..., t\}$, let $\mathcal{S}$ be a superset of those nodes (i.e. $\{1, ..., t\} \subseteq \mathcal{S}$) and let $D_{\mathcal{S}|\boldsymbol{y}_{1:t}}$ be the domain of configurations $\boldsymbol{y}_{\mathcal{S}}$ given the partial assignment $\boldsymbol{y}_{1:t}$:

$$D_{\mathcal{S}|\boldsymbol{y}_{1:t}} = \{\boldsymbol{y}'_{\mathcal{S}} \in D_{\mathcal{S}} : \boldsymbol{y}'_{1:t} = \boldsymbol{y}_{1:t}\} = \prod_{v \in \{1,...,t\}} \{y_v\} \times D_{\mathcal{S} \setminus \{1,...,t\}}.$$

In particular, for $\mathcal{S} = \{1, ..., t+1\}$, $D_{1:t+1|\boldsymbol{y}_{1:t}} = \prod_{v \in \{1,...,t\}} \{y_v\} \times D_{t+1}$ is the set of possible extensions of $\boldsymbol{y}_{1:t}$ by the variable y_{t+1}, which defines the set of *direct child nodes* of $\boldsymbol{y}_{1:t}$ in the tree diagram. $D_{|\boldsymbol{y}_{1:t}} = D_{\mathcal{V}|\boldsymbol{y}_{1:t}}$ is the set of possible *complete configurations* compatible with (or given) $\boldsymbol{y}_{1:t}$, which corresponds to the set of leaf nodes in the subtree rooted at $\boldsymbol{y}_{1:t}$ in the tree diagram.

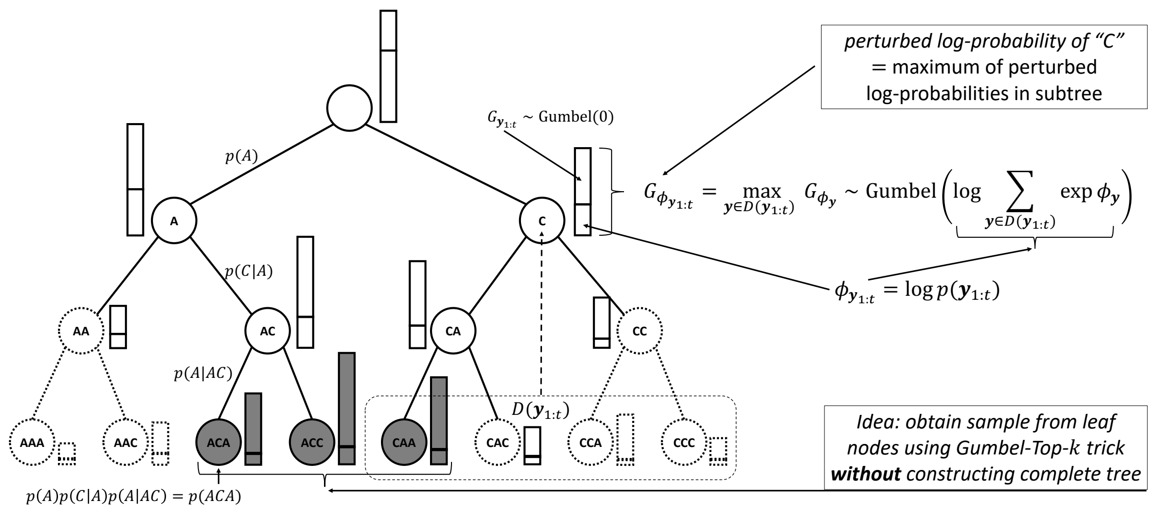

perturbed log-probability of "C" = maximum of perturbed log-probabilities in subtree
$G_{y_{1:t}} \sim \text{Gumbel}(0)$
$G_{\phi_{y_{1:t}}} = \max_{y \in D(y_{1:t})} G_{\phi_y} \sim \text{Gumbel}\left(\underbrace{\log \sum_{y \in D(y_{1:t})} \exp \phi_y}\right)$
$\phi_{y_{1:t}} = \log p(y_{1:t})$
$p(A)$
$p(C|A)$
$p(A|AC)$
$D(y_{1:t})$
A
C
AA
AC
CA
CC
AAA
AAC
ACA
ACC
CAA
CAC
CCA
CCC
$p(A)p(C|A)p(A|AC) = p(ACA)$
Idea: obtain sample from leaf nodes using Gumbel-Top-k trick **without** constructing complete tree

Figure 4: Example of Gumbel-Top-k sampling ($k = 3$) for a Bayesian network $p(\boldsymbol{y}) = p(y_1)p(y_2|y_1)p(y_3|y_1, y_2)$, represented by *probability tree diagram.* The shaded leaf nodes correspond to the configurations $\boldsymbol{y} \in D$ with the largest perturbed log-probabilities, which is the resulting sample without replacement. We also indicate the perturbed log-probabilities for partial configurations $\boldsymbol{y}_{1:t}$ (internal nodes in the tree diagram), which are the maximum of the perturbed log-probabilities in the subtree. Using top-down sampling, the dashed parts of the tree do not need to be instantiated.

We define the *marginal probability* for the partial configuration $\boldsymbol{y}_{1:t}$ by marginalizing over all possible configurations $\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}$ that match the partial configuration:

$$\begin{aligned} p(\boldsymbol{y}_{1:t}) &= \sum_{\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}} p(\boldsymbol{y}) && (13) \\ &= \sum_{\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}} \prod_{v \in \{1,\dots,t\}} p\left(y_v|\boldsymbol{y}_{\text{pa}(v)}\right) \prod_{v \in \{t+1,\dots,T\}} p\left(y_v|\boldsymbol{y}_{\text{pa}(v)}\right) \\ &= \prod_{v \in \{1,\dots,t\}} p\left(y_v|\boldsymbol{y}_{\text{pa}(v)}\right) \sum_{\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}} \prod_{v \in \{t+1,\dots,T\}} p\left(y_v|\boldsymbol{y}_{\text{pa}(v)}\right) \\ &= \prod_{v \in \{1,\dots,t\}} p\left(y_v|\boldsymbol{y}_{\text{pa}(v)}\right) \\ &= p(\boldsymbol{y}_{1:t-1}) p\left(y_t|\boldsymbol{y}_{\text{pa}(t)}\right). && (14) \end{aligned}$$

Note that $\boldsymbol{y}_{1:t-1}$ is the direct parent of the node $\boldsymbol{y}_{1:t}$ so (14) allows efficient computation of the probability $p(\boldsymbol{y}_{1:t-1})$ by multiplying the conditional probability $p\left(y_t|\boldsymbol{y}_{\text{pa}(t)}\right)$ (note that $\text{pa}(t) \subseteq \{1, ..., t-1\}$) with the marginal probability $p(\boldsymbol{y}_{1:t-1})$ of the parent in the tree diagram.

3.2. Explicit Gumbel-Top-k Sampling

By using Gumbel-Top-k sampling (see Section 2.4) *explicitly*, we can (inefficiently!) obtain a sample without replacement of size k from the Bayesian network as follows:

- Compute $\phi_{\boldsymbol{y}} = \log p_{\boldsymbol{\theta}}(\boldsymbol{y})$ for *all configurations* $\boldsymbol{y} \in D$. This means that the complete tree diagram is instantiated, as in Figure 4.
- For $\boldsymbol{y} \in D$, sample $G_{\phi_{\boldsymbol{y}}} \sim \text{Gumbel}(\phi_{\boldsymbol{y}})$, so $G_{\phi_{\boldsymbol{y}}}$ is the perturbed log-probability of $\boldsymbol{y}$.
- Let $\boldsymbol{y}_1^*, ..., \boldsymbol{y}_k^* = \arg\text{top}\, k\, G_{\phi_{\boldsymbol{y}}}$, then $\boldsymbol{y}_1^*, ..., \boldsymbol{y}_k^*$ is a sample of configurations from (1) without replacement.

Note that the result corresponds to a subset of leaf nodes in the tree diagram. As explicit Gumbel-Top-k sampling requires to instantiate the complete tree diagram to compute perturbed log-probabilities for all leaf nodes, this is computationally prohibitive. Therefore, we construct an equivalent process that we call ancestral Gumbel-Top-k sampling, which *implicitly* finds the configurations $\boldsymbol{y}_1^*, ..., \boldsymbol{y}_k^*$ with computation that is only *linear* in the number of samples k and the number of variables T.

3.3. Perturbed Log-Probabilities of Partial Configurations

So far, we have only defined the perturbed log-probabilities $G_{\phi_{\boldsymbol{y}}}$ for *complete configurations* $\boldsymbol{y} \in D$, corresponding to leaf nodes in the tree diagram. To derive ancestral Gumbel-Top-k sampling, we find it convenient to also define the perturbed log-probabilities for *partial configurations* $\boldsymbol{y}_{1:t} \in D_{1:t}$, which correspond to internal nodes in the tree diagram. From Equation (13), it follows that a partial configuration $\boldsymbol{y}_{1:t}$ has log-probability

$$\phi_{\boldsymbol{y}_{1:t}} = \log p(\boldsymbol{y}_{1:t}) = \log \sum_{\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}} p(\boldsymbol{y}) = \log \sum_{\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}} \exp \phi_{\boldsymbol{y}}.$$

For a partial configuration $\boldsymbol{y}_{1:t}$, we now consider the *maximum of the perturbed log-probabilities* $G_{\phi_{\boldsymbol{y}}}$ of compatible complete configurations $\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}$. This corresponds to the maximum of perturbed log-probabilities in the subtree below $\boldsymbol{y}_{1:t}$ in Figure 4. By the Gumbel-Max trick (Equation 6), it has a Gumbel distribution with location $\phi_{\boldsymbol{y}_{1:t}}$. Therefore, we use the notation $G_{\phi_{\boldsymbol{y}_{1:t}}}$:

$$G_{\phi_{\boldsymbol{y}_{1:t}}} = \max_{\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}} G_{\phi_{\boldsymbol{y}}} \sim \text{Gumbel}(\phi_{\boldsymbol{y}_{1:t}}). \tag{15}$$

Since $G_{\phi_{\boldsymbol{y}_{1:t}}} \sim \text{Gumbel}(\phi_{\boldsymbol{y}_{1:t}})$, we can rewrite it as

$$G_{\phi_{\boldsymbol{y}_{1:t}}} = \phi_{\boldsymbol{y}_{1:t}} + G_{\boldsymbol{y}_{1:t}}, \tag{16}$$

where we have defined $G_{\boldsymbol{y}_{1:t}} \sim \text{Gumbel}(0)$. Equation (16) reveals that we can interpret $G_{\phi_{\boldsymbol{y}_{1:t}}}$ as the *perturbed log-probability* of partial configuration $\boldsymbol{y}_{1:t}$. Thus, the perturbed log-probability of a partial configuration $\boldsymbol{y}_{1:t}$ is the maximum of perturbed log-probabilities of its possible completions $D_{|\boldsymbol{y}_{1:t}}$. This relation is key to the top-down sampling algorithm that we will derive.

Looking at the tree diagram in Figure 4, the maximum below a node $\boldsymbol{y}_{1:t}$ must be attained in one of the subtrees, such that the perturbed log-probability of $\boldsymbol{y}_{1:t}$ must be the maximum of perturbed log-probabilities of its possible one-variable extensions (children)

$\boldsymbol{y}_{1:t+1} \in D_{1:t+1|\boldsymbol{y}_{1:t}}$. Formally, we can partition the domain $D_{|\boldsymbol{y}_{1:t}}$ based on the value of $y_{t+1} \in D_{t+1}$ as $D_{|\boldsymbol{y}_{1:t}} = \bigcup_{y_{t+1} \in D_{t+1}} D_{|\boldsymbol{y}_{1:t+1}}$, such that we can write (15) as

$$G_{\phi_{\boldsymbol{y}_{1:t}}} = \max_{y_{t+1} \in D_{t+1}} \max_{\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t+1}}} G_{\phi_{\boldsymbol{y}}} = \max_{y_{t+1} \in D_{t+1}} G_{\phi_{\boldsymbol{y}_{1:t+1}}}. \tag{17}$$

This means that we can compute $G_{\phi_{\boldsymbol{y}_{1:t}}}$ for all nodes in the tree recursively, sampling $G_{\phi_{\boldsymbol{y}}}$ for the leaf nodes $\boldsymbol{y} \in D$ and computing Equation (17) recursively up the tree. We refer to this procedure as *bottom-up* sampling of the perturbed log-probabilities. Here we consider computing $G_{\phi_{\boldsymbol{y}_{1:t}}}$ using Equation (17) as a sampling step because $G_{\phi_{\boldsymbol{y}_{1:t}}}$ is a random variable which has a degenerate (constant) distribution by conditioning on the children.

3.4. Top-down Sampling of Perturbed Log-Probabilities

As an alternative to sampling the perturbed log-probabilities 'bottom-up', we can reverse the process. Starting from the root of the tree diagram with an empty configuration, which we denote by $\boldsymbol{y}_\varnothing$ with length $t = 0$, we conditionally sample the perturbed log-probabilities for the children recursively.

For the root $\boldsymbol{y}_\varnothing$ at level $t = 0$ it holds that $D(\boldsymbol{y}_\varnothing) = D$ and the log-probability is

$$\phi_{\boldsymbol{y}_\varnothing} = \log \sum_{\boldsymbol{y} \in D} \exp \phi_{\boldsymbol{y}} = \log \sum_{\boldsymbol{y} \in D} p(\boldsymbol{y}) = \log 1 = 0.$$

This means that we can sample its perturbed log-probability $G_{\phi_{\boldsymbol{y}_\varnothing}} \sim \text{Gumbel}(0)$, or we can simply set $G_{\phi_{\boldsymbol{y}_\varnothing}} = 0$, which conditions the sampling process on the event that $\max_{\boldsymbol{y} \in D} G_{\phi_{\boldsymbol{y}}} = 0$, which does not affect the result since the sample is determined by the $\arg\max$ / $\arg\operatorname{top} k$, which is independent of the maximum (see Section 2.3).

Given the log-probability $\phi_{\boldsymbol{y}_{1:t}}$ of a node $\boldsymbol{y}_{1:t}$, we can efficiently compute the log-probabilities $\phi_{\boldsymbol{y}_{1:t+1}}$ for its children as $\phi_{\boldsymbol{y}_{1:t+1}} = \phi_{\boldsymbol{y}_{1:t}} + \log p(y_t|\boldsymbol{y}_{\text{pa}(t)})$ (this follows from Equation 14). Similarly, given the *perturbed* log-probability $G_{\phi_{\boldsymbol{y}_{1:t}}}$ and the log-probabilities $\phi_{\boldsymbol{y}_{1:t+1}}$, we can *top-down* sample the perturbed log-probabilities $G_{\phi_{\boldsymbol{y}_{1:t+1}}}$ for each possible $y_{t+1} \in D_{t+1}$. However, there is a dependence between $G_{\phi_{\boldsymbol{y}_{1:t}}}$ and $G_{\phi_{\boldsymbol{y}_{1:t+1}}}$ given by $G_{\phi_{\boldsymbol{y}_{1:t}}} = \max_{y_{t+1} \in D_{t+1}} G_{\phi_{\boldsymbol{y}_{1:t+1}}}$ (Equation 17). Therefore, when sampling $G_{\phi_{\boldsymbol{y}_{1:t+1}}}$, we need to make sure this dependency is satisfied, i.e. we need to sample a set of (independent) Gumbel variables *conditionally upon their maximum*.

3.5. Sampling a Set of Gumbel Variables Conditionally on Their Maximum

To sample a set of Gumbel variables $G_{\phi_i} \sim \text{Gumbel}(\phi_i)$ with a given maximum T, we can:

- Sample G_{ϕ_i} for all i *independently*
- Let $Z = \max_i G_{\phi_i}$ be the observed maximum
- Let $\tilde{G}_{\phi_i} = F^{-1}_{\phi_i,T}(F_{\phi_i,Z}(G_{\phi_i}))$, where $F_{\phi_i,Z}$ and $F^{-1}_{\phi_i,T}$ are the CDF and inverse CDF of truncated Gumbel Distributions.

The CDF of a Gumbel(ϕ) distribution truncated at T is given by

$$F_{\phi,T}(g) = P(G \leq g | G \leq T) = \frac{P(G \leq g \cap G \leq T)}{P(G \leq T)} = \frac{P(G \leq \min(g,T))}{P(G \leq T)} = \frac{F_\phi(\min(g,T))}{F_\phi(T)},$$

where $F_\phi(g) = \exp(-\exp(\phi - g))$ is the CDF of the (untruncated) Gumbel(ϕ) distribution (Equation 4). The *inverse* CDF of the truncated Gumbel distribution is given by

$$F_{\phi,T}^{-1}(u) = \phi - \log(\exp(\phi - T) - \log u),$$

such that the transformation $\tilde{G}_{\phi_i} = F_{\phi_i,T}^{-1}(F_{\phi_i,Z}(G_{\phi_i}))$ can be written explicitly as

$$\begin{aligned}\tilde{G}_{\phi_i} &= F_{\phi_i,T}^{-1}(F_{\phi_i,Z}(G_{\phi_i}))\\ &= \phi_i - \log(\exp(\phi_i - T) - \exp(\phi_i - Z) + \exp(\phi_i - G_{\phi_i}))\\ &= -\log(\exp(-T) - \exp(-Z) + \exp(-G_{\phi_i})). \qquad (18)\end{aligned}$$

This shows that the samples with maximum Z are effectively 'shifted' towards their desired maximum T, in (negative) exponential space through the (inverse) truncated Gumbel CDF. See Appendix A for a numerically stable implementation.

The validity of the sampling procedure is shown by conditioning on $\arg\max_j G_{\phi_j}$.

For $i = \arg\max_j G_{\phi_j}$ it holds that

$$\begin{aligned}&P(\tilde{G}_{\phi_i} \leq g | i = \arg\max_j G_{\phi_j})\\ =&\mathbb{E}_Z\left[P(\tilde{G}_{\phi_i} \leq g | i = \arg\max_j G_{\phi_j}, \max_j G_{\phi_j} = Z)\right]\\ =&\mathbb{E}_Z\left[P(\tilde{G}_{\phi_i} \leq g | G_{\phi_i} = Z)\right]\\ =&\mathbb{E}_Z\left[P(F_{\phi_i,T}^{-1}(F_{\phi_i,Z}(G_{\phi_i})) \leq g | G_{\phi_i} = Z)\right]\\ =&\mathbb{E}_Z\left[P(F_{\phi_i,T}^{-1}(F_{\phi_i,Z}(Z)) \leq g)\right]\\ =&\mathbb{E}_Z\left[P(T \leq g)\right]\\ =&P(T \leq g)\\ =&P(G_{\phi_i} \leq g | G_{\phi_i} = T)\\ =&P(G_{\phi_i} \leq g | \max_j G_{\phi_j} = T, i = \arg\max_j G_{\phi_j}).\end{aligned}$$

For $i \neq \arg\max_j G_{\phi_j}$ it holds that

$$\begin{aligned}&P(\tilde{G}_{\phi_i} \leq g | i \neq \arg\max_j G_{\phi_j})\\ =&\mathbb{E}_Z\left[P(\tilde{G}_{\phi_i} \leq g | i \neq \arg\max_j G_{\phi_j}, \max_j G_{\phi_j} = Z)\right]\\ =&\mathbb{E}_Z\left[P(\tilde{G}_{\phi_i} \leq g | G_{\phi_i} < Z)\right]\\ =&\mathbb{E}_Z\left[P(F_{\phi_i,T}^{-1}(F_{\phi_i,Z}(G_{\phi_i})) \leq g | G_{\phi_i} < Z)\right]\\ =&\mathbb{E}_Z\left[P(G_{\phi_i} \leq F_{\phi_i,Z}^{-1}(F_{\phi_i,T}(g)) | G_{\phi_i} < Z)\right]\\ =&\mathbb{E}_Z\left[F_{\phi_i,Z}(F_{\phi_i,Z}^{-1}(F_{\phi_i,T}(g)))\right]\\ =&\mathbb{E}_Z\left[F_{\phi_i,T}(g)\right]\\ =&F_{\phi_i,T}(g)\\ =&P(G_{\phi_i} \leq g | G_{\phi_i} < T)\\ =&P(G_{\phi_i} \leq g | \max_j G_{\phi_j} = T, i \neq \arg\max_j G_{\phi_j}).\end{aligned}$$

Combining the two cases, it holds that

$$\begin{aligned}P(\tilde{G}_{\phi_i} \leq g) =&P(\tilde{G}_{\phi_i} \leq g | i = \arg\max_j G_{\phi_j})P(i = \arg\max_j G_{\phi_j}) +\\ &P(\tilde{G}_{\phi_i} \leq g | i \neq \arg\max_j G_{\phi_j})P(i \neq \arg\max_j G_{\phi_j})\\ =&P(G_{\phi_i} \leq g | \max_j G_{\phi_j} = T, i = \arg\max_j G_{\phi_j})P(i = \arg\max_j G_{\phi_j}) +\\ &P(G_{\phi_i} \leq g | \max_j G_{\phi_j} = T, i \neq \arg\max_j G_{\phi_j})P(i \neq \arg\max_j G_{\phi_j})\\ =&P(G_{\phi_i} \leq g | \max_j G_{\phi_j} = T).\end{aligned}$$

This procedure allows us to recursively sample $G_{\phi_{\boldsymbol{y}_{1:t}}}$ for the complete tree diagram top-down, which is equivalent to sampling the perturbed log-probabilities for the complete tree diagram bottom-up. This means that for the leaves, or complete configurations $\boldsymbol{y}$, it holds that $G_{\phi_{\boldsymbol{y}}} \sim \text{Gumbel}(\phi_{\boldsymbol{y}})$ *independently.* The benefit of the top-down sampling procedure is that we can choose to sample *only* the parts of the tree diagram needed to obtain the top k leaves.

3.6. Ancestral Gumbel-Top-k Sampling

At the heart of the ancestral Gumbel-Top-k sampling is the idea that we can find the configurations corresponding to the top k perturbed log-probabilities, *without* instantiating the complete domain, by using top-down sampling. To do so, we maintain a *queue* Q of partial configurations $\boldsymbol{y}_{1:t}$, where no partial configuration $\boldsymbol{y}_{1:t} \in Q$ is a prefix of another one. By doing so, each element $\boldsymbol{y}_{1:t} \in Q$ can be seen as the root of a (disjoint) part of the domain $D_{|\boldsymbol{y}_{1:t}} \subseteq D$. We do this in such a way that the queue represents a *partitioning* of the complete domain, so $D = \bigcup_{\boldsymbol{y}_{1:t} \in Q} D_{|\boldsymbol{y}_{1:t}}$.

For each of the partial configurations in the queue, we also keep track of the perturbed log-probability $G_{\phi_{\boldsymbol{y}_{1:t}}}$, which is obtained by top-down sampling. The queue represents the set of leaf nodes of a partially constructed probability tree diagram (see Figure 5). At any point in time, we can remove an element $\boldsymbol{y}_{1:t}$ from the queue and add to the queue the set of extensions $\boldsymbol{y}_{1:t+1}$ for $y_{t+1} \in D^{t+1}$, for which we can sample the perturbed log-probabilities $G_{\phi_{\boldsymbol{y}_{1:t+1}}}$ conditionally upon the value of their parent $G_{\phi_{\boldsymbol{y}_{1:t}}}$, as in standard ancestral sampling. Repeating this process will ultimately result in a queue $Q = D$, sampling all complete configurations $\boldsymbol{y} \in D$, including their perturbed log-probabilities, such that the top k determines the sample without replacement. To improve efficiency, we can bound the size of the queue, limiting the total number of expansions, without changing the result.

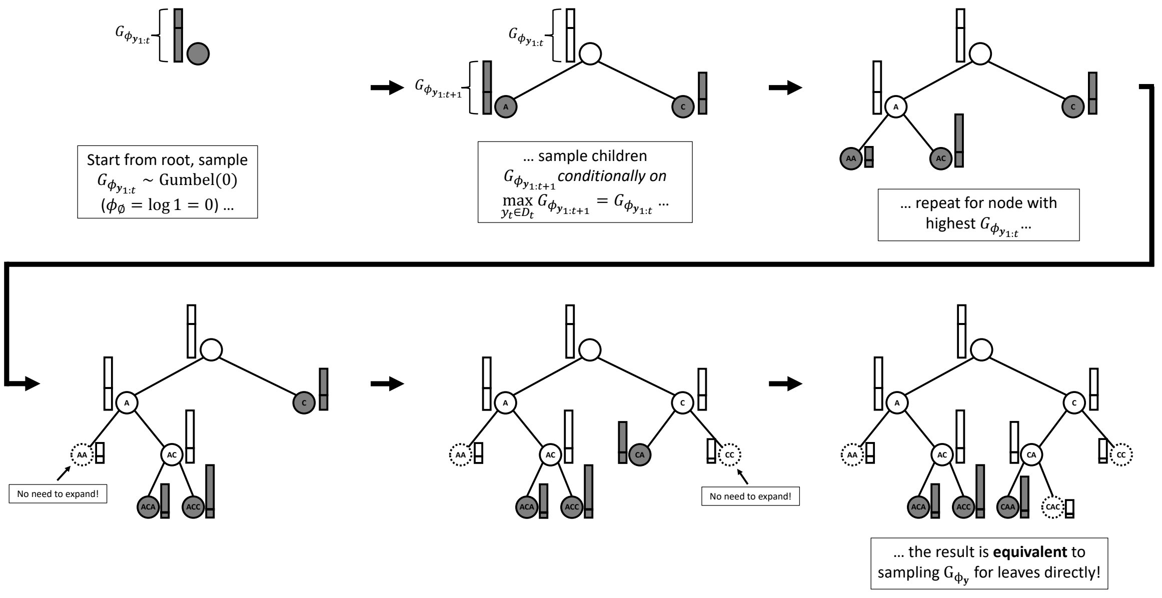

$G_{\phi_{y_{1:t}}}$
$G_{\phi_{y_{1:t}}}$
$G_{\phi_{y_{1:t+1}}}$
A
C
Start from root, sample $G_{\phi_{y_{1:t}}} \sim \text{Gumbel}(0)$ $(\phi_\emptyset = \log 1 = 0)$ …
… sample children $G_{\phi_{y_{1:t+1}}}$ *conditionally on* $\max_{y_t \in D_t} G_{\phi_{y_{1:t+1}}} = G_{\phi_{y_{1:t}}}$ …
AA
AC
… repeat for node with highest $G_{\phi_{y_{1:t}}}$…
ACA
ACC
CA
CC
CAA
CAC
No need to expand!
No need to expand!
… the result is **equivalent** to sampling G_{ϕ_y} for leaves directly!

Figure 5: Ancestral Gumbel-Top-k sampling, with $k = 3$ and $m = 1$. Configurations in the queue are shaded and in each iteration, the single ($m = 1$) incomplete configuration with the highest perturbed log-probability is expanded, sampling the perturbed log-probabilities for the extensions conditionally. This takes 5 iterations and has a total cost of 5 model evaluations to obtain $k = 3$ samples without replacement.

3.7. Bounding of the Queue Size at k

Assuming we are interested in sampling k configurations without replacement, let $\kappa = \kappa(D)$ be the k-th largest of the perturbed log-probabilities $\{G_{\phi_{\boldsymbol{y}}} : \boldsymbol{y} \in D\}$ of complete configurations $\boldsymbol{y} \in D$. We call κ the *threshold* since, using Gumbel-Top-k sampling, $\boldsymbol{y}$ will be part of the k samples without replacement if $G_{\phi_{\boldsymbol{y}}} \geq \kappa$.

Lemma 1 *(Lower bound) Let $\kappa(Q)$ be the k-th largest perturbed log-probability of (partial) configurations in the queue Q. Then $\kappa(Q)$ is a lower bound for κ, i.e. $\kappa \geq \kappa(Q)$.*

Proof For each of the k largest perturbed log-probabilities $G_{\phi_{\boldsymbol{y}_{1:t}}}$ in the queue Q, it holds that $G_{\phi_{\boldsymbol{y}_{1:t}}} \geq \kappa(Q)$ (by definition). By the definition of $G_{\phi_{\boldsymbol{y}_{1:t}}}$ as the maximum of perturbed log-probabilities $G_{\phi_{\boldsymbol{y}}}$ for $\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}$ (Equation 15), there must necessarily be a completion $\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}$ for which $G_{\phi_{\boldsymbol{y}}} = G_{\phi_{\boldsymbol{y}_{1:t}}} \geq \kappa(Q)$. Since for $\boldsymbol{y}_{1:t} \in Q$ the corresponding domains $D_{|\boldsymbol{y}_{1:t}}$ are disjoint, this means that there are (at least) k distinct configurations $\boldsymbol{y} \in D$ such that $G_{\phi_{\boldsymbol{y}}} \geq \kappa(Q)$. Since there are at least k configurations $\boldsymbol{y} \in D$ for which $G_{\phi_{\boldsymbol{y}}} \geq \kappa(Q)$, and κ is the k-th largest perturbed log-probability in D, it must hold that $\kappa \geq \kappa(Q)$. ■

Lemma 2 *(Upper bound) For $\boldsymbol{y}_{1:t} \in Q$, the perturbed log-probability $G_{\phi_{\boldsymbol{y}_{1:t}}}$ is an upper bound for the perturbed log-probabilities of its possible completions, i.e. $G_{\phi_{\boldsymbol{y}_{1:t}}} \geq G_{\phi_{\boldsymbol{y}}}$ for $\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}$.*

Proof It follows directly from the the definition of $G_{\phi_{\boldsymbol{y}_{1:t}}}$ (Equation 15) that

$$G_{\phi_{\boldsymbol{y}_{1:t}}} = \max_{\boldsymbol{y}' \in D_{|\boldsymbol{y}_{1:t}}} G_{\phi_{\boldsymbol{y}'}} \geq G_{\phi_{\boldsymbol{y}}} \quad \forall \boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}.$$

■

Theorem 3 *(Limit queue size) All except the k configurations with the largest perturbed log-probabilities in the queue Q can be discarded while still resulting in a sample without replacement of size k.*

Proof For $\boldsymbol{y}_{1:t} \in Q$ that does *not* belong to one of the top k perturbed log-probabilities, it holds that $G_{\phi_{\boldsymbol{y}_{1:t}}} < \kappa(Q)$ (assuming uniqueness of $G_{\phi_{\boldsymbol{y}_{1:t}}}$ since the domain is continuous) such that for its possible completions $\boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}$ it holds that (using Lemma 1 and 2)

$$G_{\phi_{\boldsymbol{y}}} \leq G_{\phi_{\boldsymbol{y}_{1:t}}} < \kappa(Q) \leq \kappa \quad \forall \boldsymbol{y} \in D_{|\boldsymbol{y}_{1:t}}.$$

If $\boldsymbol{y}_{1:t}$ is not one of the top k in Q, then $G_{\phi_{\boldsymbol{y}}} < \kappa$ for all possible completions of $\boldsymbol{y}_{1:t}$, and any such completion will *not* be in the sample without replacement of size k. This means that $\boldsymbol{y}_{1:t}$ can be discarded from the queue without affecting the result. ■

As a consequence of Theorem 3, if we desire to sample k configurations, then at any point in time, we need to keep at most the k partial configurations with largest perturbed log-probabilities in the queue. This can be implemented by maintaining Q as a priority queue of limited size k, where the priorities are given by the perturbed log-probabilities. Repeatedly removing the first *incomplete* configuration (with the highest perturbed log-probability) from the queue, and replacing it by its extensions, ultimately results in a queue with k complete configurations, corresponding to the k largest perturbed log-probabilities, as is illustrated in Figure 5. By its equivalence to explicit Gumbel-Top-k sampling, this is a sample without replacement of size k (in order!). By repeatedly expanding the partial configuration with highest perturbed log-probability, the algorithm is a *best first search* algorithm.

In a practical implementation, we can directly 'yield' the first complete configuration once it is found, remove it from the queue and decrease the queue size limit by 1. This will yield the first sample in exactly T iterations (where T is the number of variables), as if we used standard ancestral sampling. Then, it will 'jump' back to the partial configuration $\boldsymbol{y}_{1:t}$ with the highest perturbed log-probability and repeat the best first search from there, lazily generating each next sample as the algorithm proceeds. By reusing partially sampled configurations, the algorithm evaluates the model for each possible partial configuration at most once. It is 'anytime optimal' in the sense that to yield $k' \leq k$ samples, it only computes model evaluations that are strictly necessary. The algorithm is a special case of Algorithm 1 with $m = 1$ (see Section 3.8) and a fixed variable selection strategy (Section 3.9).

Algorithm 1 AncestralGumbelTop$k$Sampling($p_{\boldsymbol{\theta}}$, k, m)

```
1: Input: Bayesian network $p_{\boldsymbol{\theta}}(y_v|\text{pa}(v))\ \forall v \in \mathcal{V}$, max. sample size $k$, parallel expansion parameter
   $m$, variable selection strategy $\mathfrak{V}$
2: Initialize RESULT and QUEUE empty
3: add $(\mathcal{S} = \varnothing, \boldsymbol{y}_\mathcal{S} = \varnothing, \phi_{\boldsymbol{y}_\mathcal{S}} = 0, G_{\phi_{\boldsymbol{y}_\mathcal{S}}} = 0)$ to QUEUE
4: while QUEUE not empty do
5:   EXPAND $\leftarrow$ take and remove top $m$ from QUEUE according to $\tilde{G}$
6:   for $(\mathcal{S}, \boldsymbol{y}_\mathcal{S}, \phi_{\boldsymbol{y}_\mathcal{S}}, G_{\phi_{\boldsymbol{y}_\mathcal{S}}}) \in$ EXPAND (in parallel) do
7:     if $\mathcal{S} = \mathcal{V}$ then
8:       add $(\mathcal{S}, \boldsymbol{y}_\mathcal{S}, \phi_{\boldsymbol{y}_\mathcal{S}}, G_{\phi_{\boldsymbol{y}_\mathcal{S}}})$ to QUEUE
9:     else
10:      select $v$ from $\{v \in \mathcal{V} \setminus \mathcal{S} : \text{pa}(v) \subseteq \mathcal{S}\}$ according to variable selection strategy $\mathfrak{V}$
11:      let $\mathcal{S}' = \mathcal{S} \cup \{v\}$
12:      compute $\phi_{\boldsymbol{y}_{\mathcal{S}'}} \leftarrow \phi_{\boldsymbol{y}_\mathcal{S}} + \log p_{\boldsymbol{\theta}}(y_v|\boldsymbol{y}_{pa(v)}) \quad \forall y_v \in D_v$ (here $\boldsymbol{y}_{\mathcal{S}'} = \boldsymbol{y}_\mathcal{S} \cup \{y_v\}$)
13:      sample $G_{\phi_{\boldsymbol{y}_{\mathcal{S}'}}} \sim \text{Gumbel}(\phi_{\boldsymbol{y}_{\mathcal{S}'}}) \quad \forall y_v \in D_v$ conditionally given maximum $G_{\phi_{\boldsymbol{y}_\mathcal{S}}}$
14:      add $(\mathcal{S}', \boldsymbol{y}_{\mathcal{S}'}, \phi_{\boldsymbol{y}_{\mathcal{S}'}}, G_{\phi_{\boldsymbol{y}_{\mathcal{S}'}}})$ to QUEUE for all $y_v \in D_v$
15:    end if
16:  end for
17:  QUEUE $\leftarrow$ take top $k$ from QUEUE according to $\tilde{G}$
18:  while first element from QUEUE is complete, i.e. has $\mathcal{S} = \mathcal{V}$ do
19:    remove first element from QUEUE and add it to RESULT
20:    $k \leftarrow k - 1$
21:  end while
22: end while
23: Return RESULT
```

3.8. Parallelizing the Algorithm

Instead of only expanding the first partial configuration in the queue, we can assume availability of $m \leq k$ parallel processors and expand m partial configurations *in parallel*, replacing all of them by their (disjoint) expansions. This results in fewer required iterations (known as the *span* or *depth*) of the algorithm, but an increase of total *cost* of the algorithm, which is the number of iterations times m. This is because of the cost incurred for 'idle' parallel processors if initially the queue size is smaller than m, and because of redundant model evaluations if nodes are expanded which, in a fully sequential setting, would have been 'pushed' of the queue by other expansions (see Figure 6 for an example).

For $m = k$, the complete queue gets replaced in every iteration by the top k of all expansions, and the resulting algorithm is a limited width *breadth first search* which was originally presented in Kool et al. (2019b) as Stochastic Beam Search. While this algorithm requires the fewest iterations, as it guarantees k samples are generated in exactly T iterations (where T is the number of variables), it is the least efficient in terms of total cost.

Ultimately, the optimal choice of m depends on properties of the model and available parallel hardware, e.g. the number of GPUs[3] (or parallel processors on a single GPU) in modern Deep Learning applications. For example, if a single model evaluation can already fully utilize the GPUs (e.g. if it is a large and highly parallelizable neural network model), then setting $m = 1$ is optimal as this computes only strictly necessary model evaluations. On the other hand, if GPUs cannot be fully utilized for individual model evaluations, or additional GPUs are freely available, one should use $m = k$ to minimize the number of iterations which then directly translates to minimizing running time.

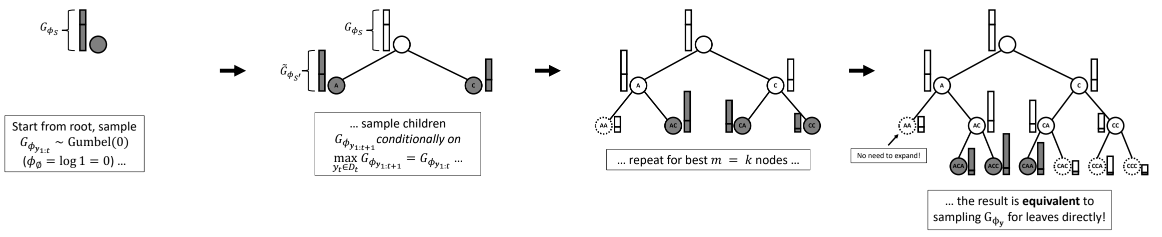

G_{ϕ_S}
Start from root, sample $G_{\phi_{y_{1:t}}} \sim \text{Gumbel}(0)$ $(\phi_\emptyset = \log 1 = 0)$ …
G_{ϕ_S}
$\tilde{G}_{\phi_{S'}}$
A
C
… sample children $G_{\phi_{y_{1:t+1}}}$ *conditionally on* $\max_{y_t \in D_t} G_{\phi_{y_{1:t+1}}} = G_{\phi_{y_{1:t}}}$ …
AA
AC
CA
CC
… repeat for best $m = k$ nodes …
No need to expand!
ACA
ACC
CAA
CAC
CCA
CCC
… the result is **equivalent** to sampling G_{ϕ_y} for leaves directly!

Figure 6: Ancestral Gumbel-Top-k sampling, with $k = m$ $(= 3)$, also known as Stochastic Beam Search (Kool et al., 2019b). This version expands in each iteration all partial configurations in the queue in parallel, traversing the tree diagram per level, i.e. using breadth first search. Compared to ancestral sampling with $m = 1$, this reduces the number of iterations from 5 to 3, but has a total cost of 3 iterations $\times$ 3 processors = 9. The total cost can be divided into a cost of 6 (rather than 5) for model evaluations, in this case since 'CC' is expanded unnecessarily, and a cost of 3 for idle processors if the queue (marked by shaded nodes) size is smaller than m: this cost is 2 in the first iteration and 1 in the second. The result is equivalent to using $m = 1$ (Figure 5) or any other $m < k$.

3. GPU is an abbreviation of Graphical Processing Unit.

3.9. Variable Selection Strategy

We can consider a tree diagram as a tree representing the possible paths (from root to leaf) that standard ancestral sampling can take to generate a sample. In the tree diagram in Figure 4, each level t corresponds to the fixed variable y_t, such that the sample is always generated in the order $y_1, ..., y_T$. However, if the graphical model has multiple topological orderings, any such order yields a valid tree diagram and thus valid order for ancestral sampling. Moreover, the order does not have to be globally fixed, but can vary depending on the partial configuration sampled so far.

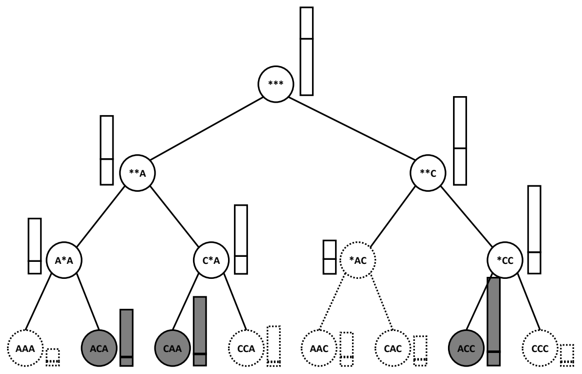


Figure 7: Example of top-down sampling using a dynamic variable selection strategy for a graphical model $p(\boldsymbol{y}) = p(y_1)p(y_2)p(y_3)$.

In general, when ancestral sampling, we are free to choose any variable y_t to sample next, as long as its parents $\boldsymbol{y}_{\text{pa}(t)}$ have been sampled. For example, for a fully factorized model $p(\boldsymbol{y}) = p(y_1)p(y_2)p(y_3)$, we are free to start sampling with y_3, and, *depending on the value of* y_3, select either y_1 or y_2 to sample next. This is an example of a *variable selection strategy*, which is represented by the tree diagram in Figure 7. While the resulting distribution is the same, the variable selection strategy affects which parts of the tree can be discarded (dashed nodes in Figure 7) and thus the number of iterations that are required by the algorithm. Sampling low-entropy variables first is a good idea, as this allows larger partial configurations to be 'reused' for subsequent samples (see Section 5.1.4). Algorithm 1 is presented in the form that can take an arbitrary variable selection strategy.

4. Related Algorithms

In this section we discuss a number of algorithms related to ancestral Gumbel-Top-k sampling.

4.1. The 'Trajectory Generator'

For $m = 1$, our ancestral Gumbel-Top-k sampling algorithm is similar to the 'trajectory generator' with $\epsilon = 0$ described by Lorberbom et al. (2019), where it is used in the context of reinforcement learning (Sutton and Barto, 2018) to generate direct policy gradients. Our algorithm replaces a partial configuration $\boldsymbol{y}_{1:t}$ in the queue by *all* expansions $\boldsymbol{y}_{1:t+1}$, motivated by the idea that this is efficient if we consider computing $p(\boldsymbol{y}_{1:t+1}|\boldsymbol{y}_{1:t})$ for all y_{t+1} to be a single model evaluation, e.g. the forward pass of a neural network with a softmax output layer (Equation 2). In the probability tree diagram, this means that if node is expanded, all childnodes are added to the tree. In the context of reinforcement learning, the algorithm by Lorberbom et al. (2019) is assumed to operate on a 'state-reward tree', where generating a new node corresponds to taking an action which requires an interaction with the environment. Therefore, they only add a *single* child node to the tree in each iteration.

In the implementation by Lorberbom et al. (2019), elements in the queue do not (only) represent leaf nodes of the tree, but the idea that they form a partition of the domain is still maintained. In particular, in our implementation, upon expansion we partition the domain $D_{|\boldsymbol{y}_{1:t}}$ represented by $\boldsymbol{y}_{1:t}$ as $D_{|\boldsymbol{y}_{1:t}} = \cup_{y_{t+1} \in D_t} D_{|\boldsymbol{y}_{1:t+1}}$, whereas Lorberbom et al. (2019) expand only a single child node and partition the domain as $D_{|\boldsymbol{y}_{1:t}} = D_{|\boldsymbol{y}_{1:t+1}} \cup (D_{|\boldsymbol{y}_{1:t}} \setminus D_{|\boldsymbol{y}_{1:t+1}})$. When a single child node $\boldsymbol{y}_{1:t+1}$ is created, this corresponds to the maximum and thus it inherits the perturbed log-probability from the parent. For the parent, a new perturbed log-probability is sampled, truncated at the previous maximum, which then becomes the maximum of perturbed log-probabilities for the remaining domain $D_{|\boldsymbol{y}_{1:t}} \setminus D_{|\boldsymbol{y}_{1:t+1}}$ corresponding to the child nodes that have not yet been instantiated.

4.2. (Stochastic) Beam Search

Beam search is a widely used method for approximate inference in various domains such as machine translation (Sutskever et al., 2014; Bahdanau et al., 2015) and image captioning (Vinyals et al., 2015). In machine learning, beam search is typically a test-time procedure, but there are works that include beam search in the training loop (Daumé et al., 2009; Wiseman and Rush, 2016; Edunov et al., 2018b; Negrinho et al., 2018; Edunov et al., 2018a). Beam search suffers from limited diversity of the results, and variants have been developed that encourage diversity (Li et al., 2016; Shao et al., 2017; Vijayakumar et al., 2018).

We argue that adding stochasticity is also a principled way to increase diversity in a beam search: this motivated the development of Stochastic Beam Search (Kool et al., 2019b). While Stochastic Beam Search is equivalent to ancestral Gumbel-Top-k sampling (only) for $m = k$, the result (a sample without replacement) is equivalent for *any* $m \leq k$. As such, we also consider ancestral Gumbel-Top-k sampling as a principled alternative to a heuristically randomized/diversified beam search, even though it is *not* (in general) a beam search.

We analyze the result of ancestral Gumbel-Top-k sampling by comparing Stochastic Beam Search to a naïve alternative implementation of a randomized beam search. In particular, imagine that we use an ordinary beam search, but replace the deterministic top-k operation by sampling without replacement in each step, e.g. using Gumbel-Top-k sampling, but *without* top-down sampling conditionally. In this naïve approach, a low-probability partial configuration will only be extended to completion, if it gets to be re-sampled, independently, again with low probability, at each step during the beam search. The result is a much lower probability to sample this configuration than assigned by the model.

Intuitively, we should somehow *commit* to a sampling 'decision' made at step t. However, a hard commitment to generate exactly one completion for each of the k partial configurations at step t would prevent generating any two completions from the same partial configuration. By using top-down sampling, Stochastic Beam Search propagates the Gumbel perturbation of a partial configuration to its extensions, which can be seen as a *soft* commitment to that partial configuration. This means that it gets extended as long as its total perturbed log-probability is among the top k, but 'falls off' the beam if, despite the consistent perturbation, another partial configuration becomes more promising. By this procedure, the result is a sample without replacement, which is true to the model, suggesting that Stochastic Beam Search, or equivalently, ancestral Gumbel-Top-k sampling, is a principled alternative to a heuristically randomized beam search.

4.3. Threshold Sampling

While Gumbel-Top-k sampling uses a fixed sample size of k, a related algorithm is *threshold sampling* (Duffield et al., 2005), which instead sets a fixed threshold and returns a variable sized sample. In analogy with Gumbel-Top-k sampling, threshold sampling would sample all configurations in the domain for which the perturbed log-probability exceeds a fixed threshold, instead of the k largest. Threshold sampling is a special case of *Poisson sampling* as each configuration is included in the sample with probability *independent* of the other configurations. This allows to use the sample for statistical estimation using the Horvitz-Thompson estimator (Horvitz and Thompson, 1952). To overcome the limitation of a variable sample size, *priority sampling* (Duffield et al., 2007) uses the $(k+1)$-th largest value as empirical threshold to obtain a fixed sample size k. This is equivalent to Gumbel-Top-k sampling and we experiment with using the resulting estimator in Section 5.3. An alternative method to control the sample size is to use an adaptive threshold (Ting, 2017).

4.4. Rejection Sampling

As an alternative to our algorithm, we can draw samples *without* replacement by rejecting duplicates from samples drawn *with* replacement. However, if the domain is large and the entropy low (e.g. with a translation model where there are only a few valid translations), then rejection sampling requires many samples and consequently many (expensive) model evaluations. Also, we have to estimate how many samples to draw (in parallel) or draw samples sequentially. Our algorithm allows to set $m = k$, which guarantees generating k unique samples with exactly kT model evaluations in *a single pass* of T steps, which is equal to the computational requirement for taking k samples with replacement. When allowing our algorithm to generate samples sequentially, setting $m < k$, it can generate k samples with fewer model evaluations than standard sampling with replacement.

4.5. Naïve Ancestral Sampling Without Replacement

As an alternative to ancestral Gumbel-Top-k sampling, we can also derive an algorithm based on 'standard' sampling without replacement. This means sampling the first configuration $\boldsymbol{y}_1^*$, then renormalizing the probability distribution to the domain $D \setminus \{\boldsymbol{y}_1^*\}$ to sample $\boldsymbol{y}_2^*$, et cetera. The naïve method we propose is similar to Wong and Easton (1980), but instead of an arbitrary binary tree, we use the tree structure (Figure 4) induced by ancestral sampling from the graphical model. This method was concurrently proposed by Shi et al. (2020).

We represent configurations $\boldsymbol{y} \in D$ as 'buckets' on a horizontal axis, with size equal to their probability $p(\boldsymbol{y})$. Sampling from $p(\boldsymbol{y})$ is equivalent to uniformly selecting a point on this axis and returning the configuration of the corresponding bucket. See an example in Figure 8. Sampling without replacement means that to obtain the second sample, we should remove the bucket corresponding to the first sample, as indicated by the dark shading in Figure 8, and sample uniformly a position on the horizontal axis that is not shaded. When using ancestral sampling, we can determine the bucket by first determining the larger bucket, corresponding to the assignment of the first variable y_1, then sequentially narrowing it down to the final bucket by sampling $y_2, ..., y_T$. Note that, although this consists of multiple sampling steps, this can be done using a single random number, as illustrated by the dotted line in Figure 8, such that the result will be equivalent to the result without ancestral sampling with the same random number.

When sampling without replacement, we desire unique *complete configurations*, but we may have duplicate *partial configurations*. This means that if we desire to use naïve ancestral sampling to obtain a second sample without replacement, we cannot remove any partial configurations from the domain, but we need to remove the *probability mass* already sampled from the partial configurations, as illustrated in Figure 8. We can keep track of the probability mass already sampled for each partial configuration by building a prefix tree (or *trie*) representing the samples, removing the probability mass from each partial configuration by backtracking the path that was used to generate each sample. Each next sample can then be generated by sampling a path down this tree using the adjusted probabilities to sample without replacement, until we arrive at a partial configuration that has not been sampled before, which means that the remaining variables in the configuration can be sampled straightforwardly and the tree is grown to include this path.

Similar to ancestral Gumbel-Top-k sampling, naïve ancestral sampling without replacement has the benefits that it can reuse model evaluations for partial configurations and generate samples one at the time. As downside, this naïve method requires more iterations as it starts from the root for each sample, whereas ancestral Gumbel-Top-k sampling directly jumps to a partial configuration from the priority queue. Also, this naïve method requires careful administration of a complete tree structure whereas ancestral Gumbel-Top-k sampling only requires to keep track of a queue of partial configurations (without any tree structure). Lastly, naïve ancestral sampling without replacement is inherently sequential.

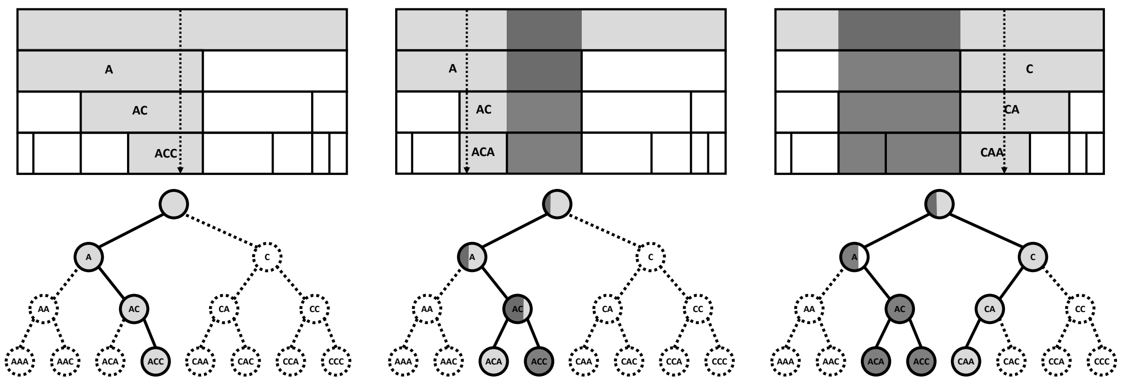


Figure 8: Sampling from a discrete domain visualized as selecting a point uniformly on a horizontal axis. The bucket (with size equal to sampling probability) which contains the random point is the sample. Ancestral sampling can be seen as sequentially narrowing down the 'bucket' that contains the sampled point, indicated by light shading. Sampling without replacement requires removing probability mass corresponding to the buckets that have already been sampled (indicated by dark shaded areas) and selecting the random point uniformly from the remaining area. When using ancestral sampling 'naïvely', we can use a tree to keep track of the probability mass that should be removed for partial configurations that have been sampled before.

4.6. Weighted Reservoir Sampling with Exponential Jumps

Weighted reservoir sampling (Efraimidis and Spirakis, 2006) is an algorithm for sampling without replacement from a stream of items given weights (unnormalized probabilities) to sample each item. Mathematically, it is equivalent to Gumbel-Top-k sampling (see Section 2.4), but it is executed in a streaming manner, i.e. perturbing (unnormalized) log-probabilities as they arrive while keeping track of the k largest perturbations so far.

In the streaming implementation, each next item replaces an item in a priority queue if its perturbed log-probability exceeds the k-th largest so far. Efraimidis and Spirakis (2006) note that the total weight that is ‘skipped’ before a next item enters the queue follows an exponential distribution, which can also be sampled directly, such that one can directly make an ‘exponential jump’ to the next item to insert the queue, without sampling the perturbations for each item individually. This can be seen as a form of top-down sampling.

Using weighted reservoir sampling, or equivalently, ‘streaming’ Gumbel-Top-k sampling, we can derive yet another algorithm for sampling without replacement from a Bayesian Network, by iterating over the complete domain (leaf nodes in the tree diagram in Figure 4) using a tree traversal algorithm, and keeping track of the k-largest perturbed log-probabilities so far. While this needs to enumerate the complete domain, we can use the ‘exponential jumps’ to directly jump to the next sequence in the domain. In doing so, complete subtrees can be skipped directly as the total weight (probability mass) in the subtree is given by the probability of the partial configuration corresponding to the root of the subtree. This means it can be executed without instantiating the complete tree. Additionally, traversing the tree by expanding the highest probability children first will limit the number of additions to the priority queue as more likely elements are inserted earlier.

While this algorithm is closely connected to our method, it is *not* equivalent. In particular, using streaming Gumbel-Top-k sampling with exponential jumps, the queue will be initially filled with k *complete* configurations, which get replaced later by other complete configurations with higher perturbed log-probabilities. The result can only be returned after the complete domain has been traversed. By contrast, ancestral Gumbel-Top-k sampling maintains *partial* configurations in the queue, which can be used to bound parts of the domain, and additionally returns samples without replacement as they are found, in order.

5. Experiments

This section presents the experiments and results.

5.1. Different Methods for Sampling Without Replacement

In our first experiment, we analyze different methods for sampling without replacement:

- *Ancestral Gumbel-Top-k sampling* (Section 3), where we experiment with different values of m to control the paralellizability of the algorithm.

- *Rejection sampling* (Section 4.4) which generates samples *with replacement* (using standard ancestral sampling) sequentially and rejects duplicates. We also implement a ‘parallel’ version of this, that generates m samples *with replacement* in parallel, then rejects the duplicates and repeats this procedure until k unique samples are found.

- *Naïve ancestral sampling* without replacement (Section 4.5). This algorithm is inherently sequential, but we also implement a 'naïve' parallelizable version similar to rejection sampling. This version generates m samples (with replacement) in parallel, removes duplicates and then constructs the tree in Figure 8 to remove the corresponding probability mass from the distribution. Then it uses this tree to generate (again in parallel) m new samples, which are distinct from the first m samples but may contain duplicates, which are again removed, after which the tree is updated. This is repeated until in total k unique samples have been generated.

5.1.1. Data Generation

We generate a random Bayesian Network of 10 variables, where each variable y_i has a Bernoulli distribution. We generate the variables in topological order, and variable i has a dependency on variable $j < i$ (so $j \in \text{pa}(i)$) with probability $c \in [0, 1]$, where c is the *connectivity* factor of the graph. For $c = 0$, the resulting graph will be fully independent (Figure 1a), while for $0 < c < 1$ the result will (most likely) be a sparse Bayesian network (Figure 1c) and for $c = 1$ the result is a 'fully connected' sequence model (Figure 1d). The Bernoulli distribution for $y_i \in \{0, 1\}$ is a mixture between an independent prior and a distribution depending on the parents, which is given by

$$P(y_i = 1|\boldsymbol{y}_{\text{pa}(i)}) = \alpha_i p_i^{\boldsymbol{y}_{\text{pa}(i)}} + (1 - \alpha_i)p_i.$$

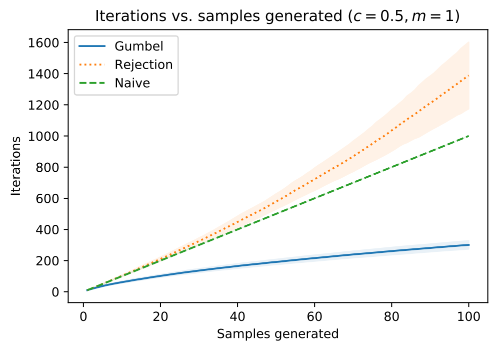
Here $\alpha_i \sim \text{Uniform}(0, 1)$ is a parameter that determines how much the variable y_i is influenced by its parents $\text{pa}(i)$, $p_i \sim \text{Uniform}(0, 1)$ is the 'independent' or 'prior' probability that $y_i = 1$, and $p_i^{\boldsymbol{y}_{\text{pa}(i)}} \sim \text{Uniform}(0, 1) \quad \forall \boldsymbol{y}_{\text{pa}(i)} \in D_{\text{pa}(i)}$ determines the influence of the parents on the probability that $y_i = 1$, given as a table for each of the $2^{|\text{pa}(v)|}$ possible values in $D_{\text{pa}(i)}$.

5.1.2. Number of Iterations

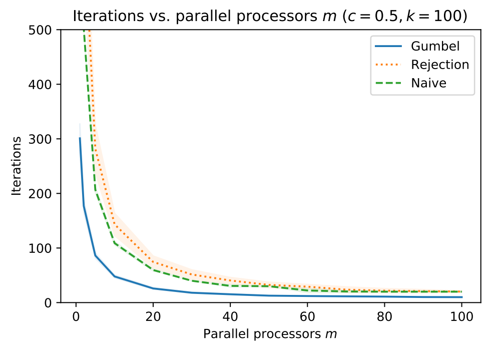
First we set $m = 1$, making all algorithms fully sequential. For our algorithm, we measure the number of iterations (removing $m = 1$ element from the queue and adding its expansions), which is equal to the number of model evaluations. For rejection sampling and the naïve ancestral sampling algorithm, we count as iteration the sampling of one variable, such that the number of iterations is also equal to the number of model evaluations and generating a single sample $\boldsymbol{y}$ with T variables takes T iterations/model evaluations. In Figure 9a we compare the number of iterations (model evaluations) needed for generating different numbers of samples using the different methods. We clearly observe how our method requires the fewest iterations/model evaluations to generate the same number of samples.

5.1.3. Parallelizing the Algorithms

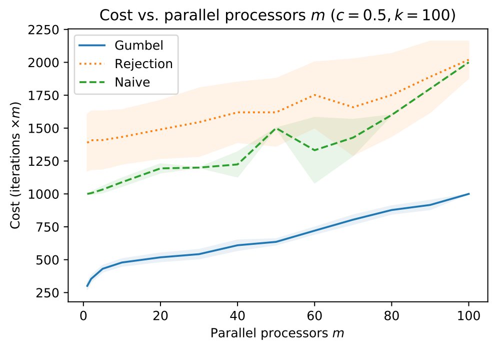
By choosing the number of parallel processors m we can trade off the total time (iterations) it takes to run the algorithm, also known as the *depth* or *span*, against the total *cost* of running the algorithm, which is m times the depth or span. Figure 9b shows for different numbers of parallel processors how many iterations it takes to generate $k = 100$ samples, i.e. it summarizes the result for the previous experiment run for different values of m. It is clear how increasing the parallelization quickly decreases the number of iterations required although the effect diminishes as m increases.



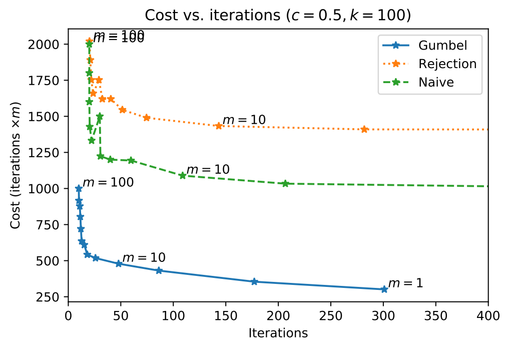
(a) Number of iterations / model evaluations as function of number of samples generated without replacement, without parallelism ($m = 1$).



(b) Number of iterations to generate $k = 100$ samples without replacement for different numbers of parallel processors m.



(c) Total *cost* (iterations $\times m$) as function of number of samples generated without replacement, without parallelism ($m = 1$).



(d) Trade-off between total cost and number of iterations, as controlled by the number of parallel processors m.

Figure 9: Results of sampling from 100 randomly generated models (Bayesian networks) with 10 Bernoulli variables each and a connectivity of $c = 0.5$. For each model, we generated $k = 100$ samples without replacement, using different methods and different numbers of parallel processors m. Rejection sampling and naïve ancestral sampling show a peak at $m = 50$ since they typically generate $3 \times 50 = 150$ samples (cost 1500) to obtain 100 unique samples, whereas $m = 40$ and $m = 60$ can suffice with generating 3×40 or $2 \times 60 = 120$ samples. Results are presented as mean and standard deviation over the 100 different models. Results with $c = 0$ or $c = 1$ (not shown) are similar in terms of iterations and costs.

To visualize the overhead from the parallelization, Figure 9c visualizes the total cost, which is the number of iterations (Figure 9b) multiplied by m. For ancestral Gumbel-Top-k sampling, the increased costs for larger m has two different sources as explained in Section 3.8: redundant model evaluations of partial configurations which would have been pushed of the queue in the sequential setting, and 'idle' parallel resources if the queue size is smaller than m. For rejection sampling, the parallel version is suboptimal because too many samples may be generated since they are generated in batches of size m. For the naïve ancestral sampling algorithm, by the batching in parallel, there may still be duplicates compared to the sequential ($m = 1$) algorithm, and more than k samples may be generated in total.

Ultimately, m determines the trade-off between the number of iterations and the total cost of the algorithm, which is visualized in Figure 9d. Our algorithm with $m = k = 100$ (i.e. Stochastic Beam Search) uses 10 iterations (since there are 10 variables) and has a cost of $10 \times 100 = 1000$, which is the *minimum* for naïve ancestral sampling and rejection sampling. On the other extreme, the sequential algorithm with $m = 1$ uses around 300 iterations with a cost of 300. The difference is large because of the small setting, where, with $m = 100$, many processors are idle in 6 of the 10 iterations since $2^6 = 64 < 100$. We expect the difference to be smaller in real applications, but this experiment clearly shows that there is a trade-off, and suggests that limited parallelization (in this case $m = 10$) rapidly decreases the number of iterations without increasing total cost too much.

5.1.4. Variable Selection Strategy

As explained in Section 3.9, we can select *any* variable from $\{v \in V \setminus \mathcal{S} : \text{pa}(v) \subseteq \mathcal{S}\}$ to sample in each iteration. We experiment with the following variable selection strategies:

- *Fixed* uses the order in which the variables are generated. Since we generated the Bayesian network in topological order, this is valid.

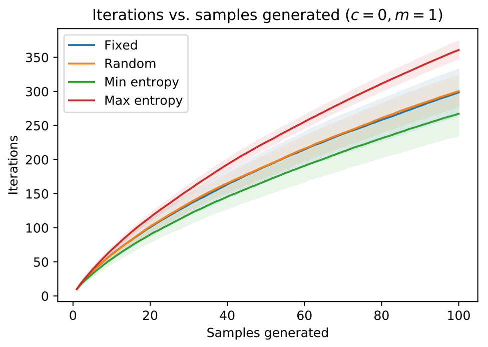
- *Random* chooses the variable from $\{v \in V \setminus \mathcal{S} : \text{pa}(v) \subseteq \mathcal{S}\}$ to sample next uniformly at random.

- *Minimum entropy* computes the (conditional) entropy for all variables $\{v \in V \setminus \mathcal{S} : \text{pa}(v) \subseteq \mathcal{S}\}$ as $-\hat{p}_v \log(\hat{p}_v) - (1 - \hat{p}_v) \log(1 - \hat{p}_v)$, where $\hat{p}_v = P(y_v = 1 | \boldsymbol{y}_{\text{pa}(v)})$ and selects the variable v with minimum entropy.

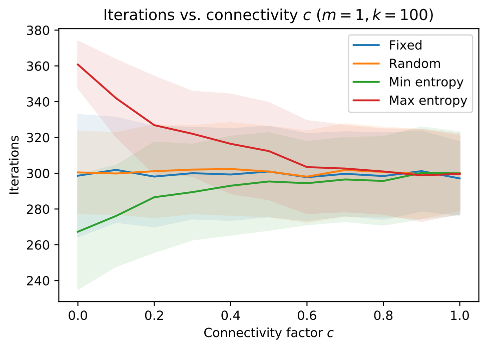
- *Maximum entropy* selects the variable v with maximum entropy.

We note that selecting v based on entropy requires the model to be evaluated, which may be undesirable in some cases. However, theoretically these model evaluations can be cached/reused as most evaluations will be required eventually to sample the remaining variables. In the extreme case of $c = 0$, i.e. the fully independent model in Figure 1a, we can simply precompute all model evaluations (which is a good idea anyway), and using the entropy variable selection strategy reduces to sorting the variables by their entropy before starting the Gumbel-Top-k sampling algorithm.

In Figure 10a we clearly see that selecting variables with minimum entropy first is the best strategy (it requires fewest iterations), whereas selecting based on maximum entropy performs worst, with a random strategy or using the fixed generation order (which can also be considered random) in between. As explained in Section 3.9, this is because selecting



(a) Number of iterations as function of number of samples generated without replacement, for a fully independent model ($c = 0$).



(b) Number of iterations to generate $k = 100$ samples without replacement for different levels of connectivity c of the model.

Figure 10: Results of generating $k = 100$ samples using ancestral Gumbel-Top-k sampling with $m = 1$ and different variable selection strategies, for models with different levels of connectivity c. Results are presented as mean and standard deviation over 100 randomly generated models.

variables with low entropy allows maximum reuse of partial configurations. In Figure 10b we show how the difference between different variable selection strategies decreases as there are more dependencies between variables, where for a 'fully connected' sequence model ($c = 1$, Figure 1d) there is no freedom in variable selection and all strategies perform the same.

5.2. Diverse Beam Search

This experiment was originally presented in Kool et al. (2019b). The results were obtained using Stochastic Beam Search, but are valid for any implementation of ancestral Gumbel-Top-k sampling with parallelism factor $m < k$. In this experiment we compare Stochastic Beam Search as a principled (stochastic) alternative to *Diverse Beam Search* (Vijayakumar et al., 2018) in the context of neural machine translation to obtain a diverse set of translations for a single source sentence $\boldsymbol{x}$. Following the setup by Vijayakumar et al. (2018) we report both diversity as measured by the fraction of unique n-grams in the k translations as well as mean and maximum BLEU score (Papineni et al., 2002) as an indication of the quality of the sample. The maximum BLEU score corresponds to 'oracle performance' reported by Vijayakumar et al. (2018), but we report the mean as well since a single good translation and $k - 1$ completely random sentences scores high on both maximum BLEU score and diversity, while being undesirable. A good method should increase diversity without sacrificing mean BLEU score.

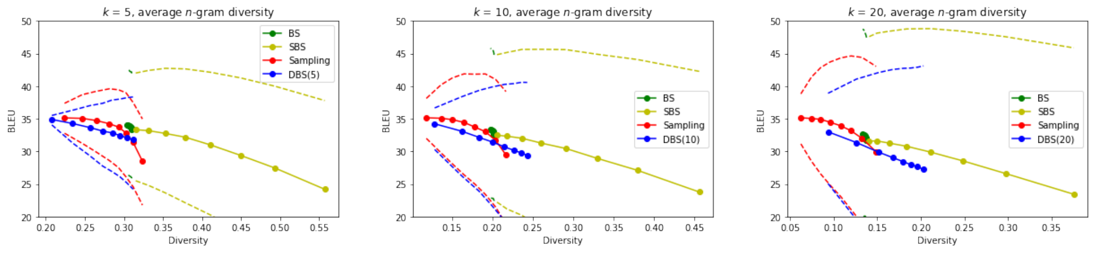
We compare four different sentence generations methods: *Beam Search* (BS), *Sampling* (with replacement), *Stochastic Beam Search* (SBS) (sampling without replacement) and *Diverse Beam Search* with G groups (DBS(G)) (Vijayakumar et al., 2018). For Sampling and Stochastic Beam Search, we control the diversity of samples generated using the *softmax temperature* τ (see Equation 2) used to compute the model probabilities. We use

$\tau = 0.1, 0.2, ..., 0.8$, where a higher τ results in higher diversity. Heuristically, we also vary τ for computing the scores with (deterministic) Beam Search. The diversity of Diverse Beam Search is controlled by the *diversity strengths* parameter, which we vary between $0.1, 0.2, ..., 0.8$. We set the number of groups G equal to the sample size k, which Vijayakumar et al. (2018) reported as the best choice.

We modify the Beam Search in `fairseq` (Ott et al., 2019) to implement Stochastic Beam Search[4], and use the `fairseq` implementations for Beam Search, Sampling and Diverse Beam Search. For theoretical correctness of the Stochastic Beam Search, we disable length-normalization (Wu et al., 2016) and early stopping (and therefore also do not use these parameters for the other methods). We use the pretrained model from Gehring et al. (2017) and use the `wmt14.v2.en-fr.newstest2014` test set[5] consisting of 3003 sentences. We run the four methods with sample sizes $k = 5, 10, 20$ and plot the minimum, mean and maximum BLEU score among the k translations (averaged over the test set) against the average $d = \frac{1}{4}\sum_{n=1}^{4} d_n$ of 1, 2, 3 and 4-gram diversity, where n-gram diversity is defined as

$$d_n = \frac{\text{\# of unique } n\text{-grams in } k \text{ translations}}{\text{total \# of } n\text{-grams in } k \text{ translations}}.$$

In Figure 11, we represent the results as curves, indicating the trade-off between diversity and BLEU score. The points indicate datapoints and the dashed lines indicate the (averaged) minimum and maximum BLEU score. For the same diversity, Stochastic Beam Search achieves higher mean/maximum BLEU score. Looking at a certain BLEU score, we observe that Stochastic Beam Search achieves the same BLEU score as Diverse Beam Search with a significantly larger diversity. For low temperatures (< 0.5), the maximum BLEU score of Stochastic Beam Search is comparable to the deterministic Beam Search, so the increased diversity does not sacrifice the best element in the sample. Note that Sampling achieves higher mean BLEU score at the cost of diversity, which may be because good translations are sampled repeatedly. However, the maximum BLEU score of both Sampling and Diverse Beam Search is lower than with Beam Search and Stochastic Beam Search.



(a) $k = 5$ (b) $k = 10$ (c) $k = 20$

Figure 11: Minimum, mean and maximum BLEU score vs. diversity for different sample sizes k. Points indicate different temperatures/diversity strengths, from 0.1 (low diversity, left in graph) to 0.8 (high diversity, right in graph).

4. Our code is available at `https://github.com/wouterkool/stochastic-beam-search`.
5. Available at `https://s3.amazonaws.com/fairseq-py/data/wmt14.v2.en-fr.newstest2014.tar.bz2`.

5.3. BLEU Score Estimation

In our second experiment, also presented originally in Kool et al. (2019b), we use sampling without replacement to evaluate the expected *sentence level* BLEU score for a translation $\boldsymbol{y}$ given a source sentence $\boldsymbol{x}$. Although we are often interested in *corpus level* BLEU score, estimation of sentence level BLEU score is useful, for example when training using minibatches to directly optimize BLEU score (Ranzato et al., 2016). We leave dependence of the BLEU score on the source sentence $\boldsymbol{x}$ implicit, and write $f(\boldsymbol{y}) = \text{BLEU}(\boldsymbol{y}, \boldsymbol{x})$. We want to estimate the following expectation:

$$\mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}\left[f(\boldsymbol{y})\right] = \sum_{\boldsymbol{y} \in D} p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) f(\boldsymbol{y}). \tag{19}$$

Under a Monte Carlo (MC) sampling *with replacement* scheme with size k, we write S as the set[6] of sampled sequences and estimate (19) using

$$\mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}\left[f(\boldsymbol{y})\right] \approx \frac{1}{k} \sum_{\boldsymbol{y} \in S} f(\boldsymbol{y}). \tag{20}$$

If the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})$ has low entropy (for example if there are only few valid translations), then MC estimation may be inefficient since repeated samples are uninformative. We can use sampling without replacement as an improvement, but we need to use importance weights to correct for the changed sampling probabilities. Using Gumbel-Top-k sampling, we can implement an estimator equivalent to the estimator described by Vieira (2017), which can be seen as a Horvitz-Thompson estimator (Horvitz and Thompson, 1952) used with priority sampling (Duffield et al., 2007):

$$\mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}\left[f(\boldsymbol{y})\right] \approx \sum_{\boldsymbol{y} \in S} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}{q_{\boldsymbol{\theta},\kappa}(\boldsymbol{y}|\boldsymbol{x})} f(\boldsymbol{y}). \tag{21}$$

Using this estimator, we 'sacrifice' the k-th sample to obtain the empirical threshold κ (which is the k-th largest perturbed log-probability, see Section 3.7), and we define S as the set of the $k-1$ sequences corresponding to the $k-1$ largest perturbed log-probabilities. It holds that $\boldsymbol{y} \in S$ if $G_{\phi_{\boldsymbol{y}}} > \kappa$, which highlights the relation to threshold sampling (see Section 4.3). We define

$$q_{\boldsymbol{\theta},a}(\boldsymbol{y}|\boldsymbol{x}) = P(G_{\phi_{\boldsymbol{y}}} > a) = 1 - \exp(-\exp(\phi_{\boldsymbol{y}} - a)). \tag{22}$$

If we would assume a fixed threshold a and *variably* sized sample $S = \{\boldsymbol{y} \in D : G_{\phi_{\boldsymbol{y}}} > a\}$, then $q_{\boldsymbol{\theta},a}(\boldsymbol{y}|\boldsymbol{x}) = P(\boldsymbol{y} \in S)$ and $\frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}{q_{\boldsymbol{\theta},a}(\boldsymbol{y}|\boldsymbol{x})}$ is a standard importance weight. Surprisingly, using a *fixed* sample size k (and empirical threshold κ) also yields in an unbiased estimator, and we include a proof adapted from Duffield et al. (2007) and Vieira (2017) in Appendix B.

Empirically, the estimator (21) has high variance, and in practice we find it is preferred to normalize the importance weights by $W(S) = \sum_{\boldsymbol{y} \in S} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}{q_{\boldsymbol{\theta},\kappa}(\boldsymbol{y}|\boldsymbol{x})}$ (Hesterberg, 1988):

$$\mathbb{E}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}\left[f(\boldsymbol{y})\right] \approx \frac{1}{W(S)} \sum_{\boldsymbol{y} \in S} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}{q_{\boldsymbol{\theta},\kappa}(\boldsymbol{y}|\boldsymbol{x})} f(\boldsymbol{y}). \tag{23}$$

6. Formally, when sampling with replacement, S is a *multiset*.

The estimator (23) is biased but consistent: in the limit $k = |D|$ we sample the entire domain, so we have empirical threshold $\kappa = -\infty$ and $q_{\boldsymbol{\theta},\kappa}(\boldsymbol{y}|\boldsymbol{x}) = 1$ and $W(S) = 1$, such that (23) is equal to (19).

We have to take care computing the importance weights as depending on the entropy the terms in the quotient $\frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}{q_{\boldsymbol{\theta},\kappa}(\boldsymbol{y}|\boldsymbol{x})}$ can become very small, and in our case the computation of $P(G_{\phi_{\boldsymbol{y}}} > a) = 1 - \exp(-\exp(\phi_{\boldsymbol{y}} - a))$ can suffer from catastrophic cancellation. For details, see Appendix C.

Because the model is not trained to use its own predictions as input, at test time errors can accumulate. As a result, when sampling with the default temperature $\tau = 1$, the expected BLEU score is very low (below 10). To improve quality of generated sentences we use lower temperatures and experiment with $\tau = 0.05, 0.1, 0.2, 0.5$. We then use different methods to estimate the BLEU score:

- *Monte Carlo* (MC), using Equation (20).
- *Stochastic Beam Search* (SBS), where we compute estimates using the estimator in Equation (21) and the normalized variant in Equation (23).
- *Beam Search* (BS), where we compute a deterministic beam S (the temperature τ affects the scoring) and compute $\sum_{\boldsymbol{y}\in S} p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) f(\boldsymbol{y})$. This is not a statistical estimator, but a lower bound to the target (19) which serves as a validation of the implementation and gives insight on how many sequences we need at least to capture most of the mass in (19). We also compute the normalized version $\frac{\sum_{\boldsymbol{y}\in S} p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) f(\boldsymbol{y})}{\sum_{\boldsymbol{y}\in S} p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}$, which can heuristically be considered as a 'determinstic estimate'.

In Figure 12 we show the results of computing each estimate 100 times (BS only once as it is deterministic) for three different sentences[7] for temperatures $\tau = 0.05, 0.1, 0.2, 0.5$ and sample sizes $k = 1$ to 250. We report the empirical mean and 2.5-th and 97.5-th percentile. The normalized SBS estimator indeed achieves significantly lower variance than the unnormalized version and for $\tau < 0.5$, it significantly reduces variance compared to MC, without adding observable bias. For $\tau = 0.5$ the results are similar, but we are less interested in this case as the overall BLEU score is lower than for $\tau = 0.2$.

5.4. Conditional Entropy Estimation

Additionally to estimating the BLEU score we use $f(\boldsymbol{y}) = -\log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})$ such that Equation (19) becomes the model entropy (conditioned on the source sentence $\boldsymbol{x}$)

$$\mathbb{E}_{\boldsymbol{y}\sim p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}\left[-\log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})\right].$$

Entropy estimation is useful in optimization settings where we want to include an entropy loss to ensure diversity. It is a different problem than BLEU score estimation as high BLEU score (for a good model) correlates positively with model probability, while for entropy rare $\boldsymbol{y}$ contribute the largest terms $-\log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})$. We use the same experimental setup as for the BLEU score and present the results in Figure 13. The results are similar to the BLEU score experiment: the normalized SBS estimate has significantly lower variance than MC for $\tau < 0.5$ while for $\tau = 0.5$, results are similar. This shows that Stochastic Beam Search can be used to construct practical statistical estimators.

7. These are sentences 1500, 2000 and 2500 from the WMT14 data set.

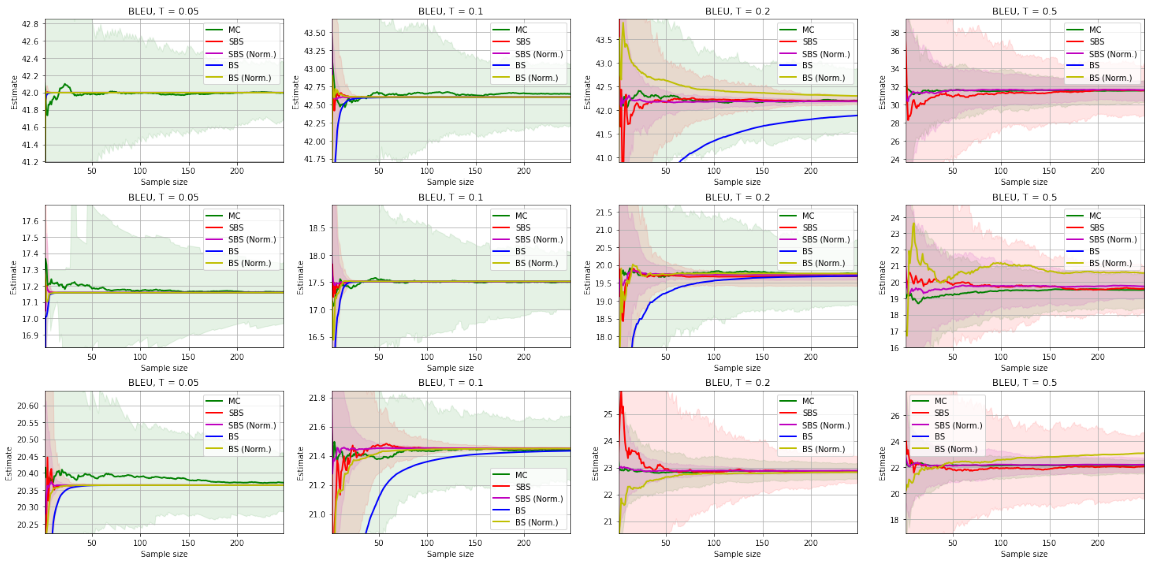


Figure 12: BLEU score estimates for three sentences sampled/decoded by different estimators for different temperatures.

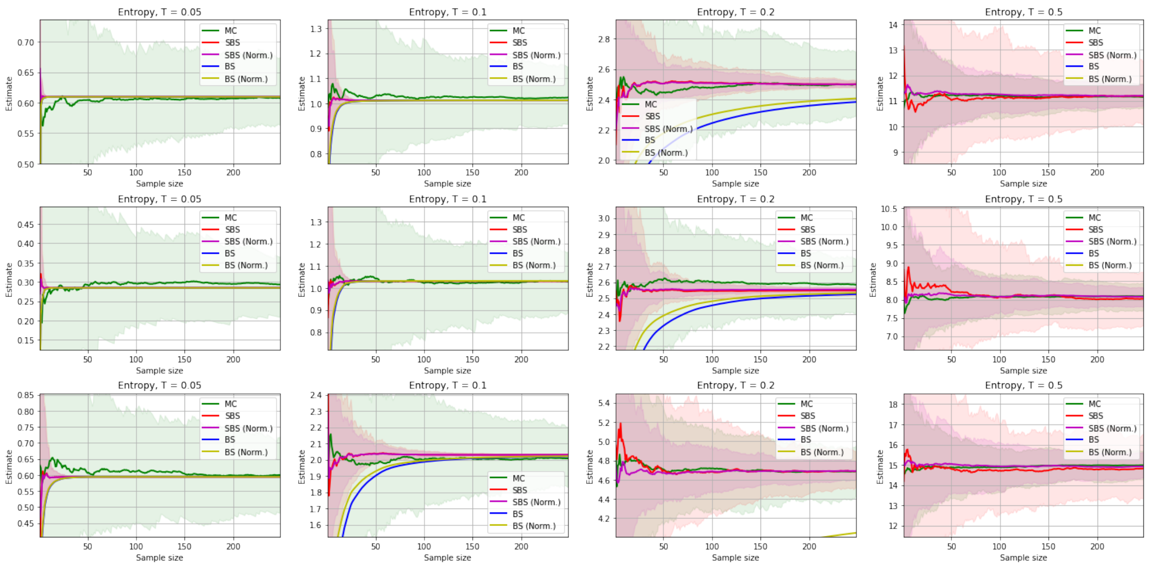


Figure 13: Entropy estimates for three sentences sampled/decoded by different estimators for different temperatures.

6. Possible Extensions of Ancestral Gumbel-Top-k Sampling

Our algorithm can be extended in various ways, to give efficient implementations of two existing algorithms or to perform efficient sampling *with* replacement.

6.1. Memory Augmented Policy Optimization

Our algorithm can be extended to give an efficient implementation of Memory Augmented Policy Optimization (MAPO) (Liang et al., 2018), an extension of REINFORCE (Williams, 1992) that optimizes a policy in reinforcement learning using a replay buffer of good experiences. MAPO computes an exact gradient for the b experiences (configurations) in the buffer, which requires b model evaluations, and uses a sample from the model outside of the buffer, obtained using (potentially inefficient) rejection sampling. We can adapt ancestral Gumbel-Top-k sampling to implement MAPO efficiently, by modifying the priority queue to use a hierarchical criterion, that will put in the front partial configurations which correspond to partial configurations in the buffer,[8] and sort the remaining configurations based on the perturbed log-probability. When using $k = b + 1$, the result will be the b configurations in the buffer (with their model evaluations), and a single sample outside of the buffer.

6.2. Rao-Blackwellized Stochastic Gradients

While MAPO is an estimator that computes an exact gradient for a number of 'good' configurations, Liu et al. (2019) proposed a similar estimator that computes an exact gradient for high probability configurations (instead of 'good' ones), which may have a greater contribution overall. While they consider 1-dimensional categorical distributions only, we note that the estimator can also be used in multi-dimensional settings, where high probability categories can be found by an approximate algorithm such as (deterministic) beam search, but in this case it is challenging to obtain a sample from the remaining domain.

Similar to MAPO, we can make a modification of our algorithm, to yield *both* the high probability configurations, and a sample from the remaining domain. In particular, we can modify Stochastic Beam Search (our algorithm with $m = k$) to maintain in the queue the $k - 1$ partial configurations with highest log-probability (*without* Gumbel perturbation) and one partial configuration which has the highest *perturbed* log-probability among the remaining configurations. This means that we have a 'deterministic' beam of size $k - 1$ as well as a single sample (partial configuration) outside of the beam.[9]

In general, it may be preferred to have fewer exact and more sampled configurations; see the relevant discussions in Fearnhead and Clifford (2003); Liu et al. (2019). Using our algorithm, we could maintain $k-\ell$ configurations by their log-probability, and ℓ configurations by the perturbed log-probability. The resulting ℓ samples without replacement can be converted to (at least) ℓ samples with replacement using resampling (see below), or one can use an estimator based on sampling without replacement (Kool et al., 2019a, 2020).

8. Existence of a partial configuration/trajectory in the buffer can be efficiently checked by representing the replay buffer as a prefix tree (*trie*), similar to Figure 8.
9. The result may be slightly different than deterministic beam search with size $k - 1$, since extensions of the k-th configuration (the sample) may have high log-probability and push other configurations off the beam. If desired, this can be heuristically prevented.

6.3. Sampling with Replacement with Sampling Without Replacement

Sampling without replacement can be used to sample *with replacement* by using a *resampling* algorithm. The basic idea is that using ancestral Gumbel-Top-k sampling (Algorithm 1) 'lazily', we can obtain the first sample $\boldsymbol{y}_1^*$, for which sampling with/without replacement is the same. Then for the second sample, we can take $\boldsymbol{y}_1^*$ with probability $p(\boldsymbol{y}_1^*)$, or sample from $D \setminus \{\boldsymbol{y}_1^*\}$ with probability $1 - p(\boldsymbol{y}_1^*)$, which can be done by continuing incremental sampling without replacement. At any point, if $\boldsymbol{y}_1^*, ..., \boldsymbol{y}_n^*$ are the samples *without replacement* so far, we can get another sample *with replacement* by choosing $\boldsymbol{y}_1^*$ with probability $p(\boldsymbol{y}_1^*)$, $\boldsymbol{y}_2^*$ with probability $p(\boldsymbol{y}_2^*)$, et cetera or find the next sample $\boldsymbol{y}_{n+1}^*$ (without replacement) with probability $1 - \sum_{i=1}^n p(\boldsymbol{y}_i^*)$. Repeating this algorithm to obtain a desired number of samples has a lower computational cost (as measured by model evaluations) than standard sampling with replacement, as resampling is cheap (it does not require additional model evaluations) and the inner ancestral Gumbel-Top-k sampling algorithm uses model evaluations efficiently.

7. Discussion

We introduced *ancestral Gumbel-Top-k sampling*, an algorithm that allows to efficiently draw samples without replacement from a probability distribution represented as a Bayesian network. It allows to control the parallelizability of the algorithm, trading off the number of required iterations versus the total cost of running the algorithm, such that it can make efficient use of modern hardware. We discussed possible extensions of the algorithm, which allow implementations of the gradient estimators by Liang et al. (2018) and Liu et al. (2019) and a resampling algorithm to efficiently obtain samples *with* replacement.

We have discussed related algorithms and empirically shown the benefits of using ancestral Gumbel-Top-k sampling: it enables to generate samples without replacement using a significantly lower computational cost than alternatives. We have analyzed the influence of the number of parallel processors experimentally, suggesting that limited parallelism quickly decreases the number of required iterations without increasing total cost too much. Additionally, we have shown how selecting the order of sampling based on entropy can reduce the cost of the algorithm, which is especially useful for sampling from fully independent models where the 'optimal' order of sampling can be determined upfront.

In the context of sequence models, Gumbel-Top-k sampling relates to sampling (with replacement) and (deterministic) beam search and can be seen as a method that combines the advantages of both. Our experiments support the idea that Gumbel-Top-k sampling can be used as a drop-in replacement in places where sampling (with replacement) or (deterministic) beam search is used. In fact, our experiments show that sampling without replacement can be used to yield lower-variance estimators and high-diversity samples from a neural machine translation model.

We hope that our method motivates future work to develop improved statistical learning methods, especially in the context of Deep Learning, based on sampling without replacement, a direction of research that has, in fact, already started (Kool et al., 2019a, 2020). We believe that ancestral Gumbel-Top-k sampling has potential to increase both computational and statistical efficiency in Deep Learning applications that involve discrete computations, such as neural machine translation (Sutskever et al., 2014; Bahdanau et al., 2015) and image captioning (Vinyals et al., 2015).

Acknowledgments

This research was funded by ORTEC.

Appendix A. Sampling a Set of Gumbels with Maximum k

As explained in Section 3.4, to sample a set of Gumbels with maximum T, let $G_{\phi_i} \sim \text{Gumbel}(\phi_i)$, let $Z = \max_i G_{\phi_i}$ and define

$$\tilde{G}_{\phi_i} = F^{-1}_{\phi_i,T}(F_{\phi_i,Z}(G_{\phi_i})) = -\log(\exp(-T) - \exp(-Z) + \exp(-G_{\phi_i})). \tag{24}$$

Direct computation of (24) can be unstable as large terms need to be exponentiated. Instead, we compute

$$\begin{aligned} v_i &= T - G_{\phi_i} + \text{log1mexp}(G_{\phi_i} - Z), \\ \tilde{G}_{\phi_i} &= T - \max(0, v_i) - \text{log1pexp}(-|v_i|) \end{aligned}$$

where we have defined

$$\begin{aligned} \text{log1mexp}(a) &= \log(1 - \exp(a)), \quad a \leq 0 \\ \text{log1pexp}(a) &= \log(1 + \exp(a)). \end{aligned}$$

This is equivalent as

$$\begin{aligned} & T - \max(0, v_i) - \log(1 + \exp(-|v_i|)) \\ =& T - \log(1 + \exp(v_i)) \\ =& T - \log\left(1 + \exp\left(T - G_{\phi_i} + \log\left(1 - \exp\left(G_{\phi_i} - Z\right)\right)\right)\right) \\ =& T - \log\left(1 + \exp\left(T - G_{\phi_i}\right)\left(1 - \exp\left(G_{\phi_i} - Z\right)\right)\right) \\ =& T - \log\left(1 + \exp\left(T - G_{\phi_i}\right) - \exp\left(T - Z\right)\right) \\ =& -\log\left(\exp(-T) + \exp(-G_{\phi_i}) - \exp(-Z)\right) \\ =& \tilde{G}_{\phi_i} \end{aligned}$$

The first step can be easily verified by considering the cases $v_i < 0$ and $v_i \geq 0$. log1mexp and log1pexp can be computed accurately using $\text{log1p}(a) = \log(1+a)$ and $\text{expm1}(a) = \exp(a) - 1$ (Mächler, 2012):

$$\text{log1mexp}(a) = \begin{cases} \log(-\text{expm1}(a)) & a > -0.693 \\ \text{log1p}(-\exp(a)) & \text{otherwise,} \end{cases}$$

$$\text{log1pexp}(a) = \begin{cases} \text{log1p}(\exp(a)) & a < 18 \\ x + \exp(a) & \text{otherwise.} \end{cases}$$

Appendix B. Unbiasedness of the Importance Weighted Estimator

We give a proof of unbiasedness of the importance weighted estimator, which is adapted from the proofs by Duffield et al. (2007) and Vieira (2017). For generality of the proof, we enumerate categories in the domain D by $i = 1, ..., n$ and we consider *general* random keys h_i for $i = 1, ..., n$ (not necessarily Gumbel perturbations). As was noted by Vieira (2017), the actual distribution of the keys does *not* influence the unbiasedness of the estimator, but does determine the effective sampling scheme. Using Gumbel perturbed log-probabilities as keys (e.g. $h_i = G_{\phi_i}$) is equivalent to the PPSWOR scheme described by Vieira (2017). For simplicity, we write $p_i = p_{\boldsymbol{\theta}}(i)$ and $q_i(a) = q_{\boldsymbol{\theta},a}(i) = P(h_i > a)$ (see Equation 22), and we define $h_{1:n} = \{h_1, ..., h_n\}$ and $h_{-i} = \{h_1, ..., h_{i-1}, h_{i+1}, ..., h_n\} = h_{1:n} \setminus \{h_i\}$.

Given a *fixed threshold* a, it holds that $q_i(a) = P(i \in S)$ is the probability that category i is included in the sample S, so it can be thought of as the *inclusion probability* of i. Given a *fixed sample size* k, let κ be the $(k+1)$-th largest element of $h_{1:n}$, so κ is the *empirical threshold*. Let κ_i' be the k-th largest element of h_{-i} (the k-th largest of all *other* elements). We first prove Lemma 4, which is then used to prove Theorem 5, which states that the importance weighted estimator is an unbiased estimator of $\mathbb{E}[f(i)]$, for a given function $f(i)$.

Lemma 4 *The expected weight of each term in the importance weighted estimator is 1:*

$$\mathbb{E}_{h_{1:n}}\left[\frac{\mathbb{1}_{\{i \in S\}}}{q_i(\kappa)}\right] = 1.$$

Proof We make use of the observation (slightly rephrased) by Duffield et al. (2007) that conditioning on h_{-i}, we know κ_i', and the event $i \in S$ implies that $\kappa = \kappa_i'$ since i will only be in the sample if $h_i > \kappa_i'$, which means that κ_i' is the $(k+1)$-th largest value of $h_{-i} \cup \{h_i\} = h_{1:n}$. The reverse is also true: if $\kappa = \kappa_i'$ then h_i must be larger than κ_i' since otherwise the $(k+1)$-th largest value of $h_{1:n}$ will be smaller than κ_i'. By separating the expectation over $h_{1:n}$ it follows that

$$\begin{aligned}
&\mathbb{E}_{h_{1:n}}\left[\frac{\mathbb{1}_{\{i \in S\}}}{q_i(\kappa)}\right] \\
=&\mathbb{E}_{h_{-i}}\left[\mathbb{E}_{h_i}\left[\left.\frac{\mathbb{1}_{\{i \in S\}}}{q_i(\kappa)}\right| h_i\right]\right] \\
=&\mathbb{E}_{h_{-i}}\left[\mathbb{E}_{h_i}\left[\left.\frac{\mathbb{1}_{\{i \in S\}}}{q_i(\kappa)}\right| h_{-i}, i \in S\right] P(i \in S | h_{-i}) + \mathbb{E}_{h_i}\left[\left.\frac{\mathbb{1}_{\{i \in S\}}}{q_i(\kappa)}\right| h_{-i}, i \notin S\right] P(i \notin S | h_{-i})\right] \\
=&\mathbb{E}_{h_{-i}}\left[\mathbb{E}_{h_i}\left[\left.\frac{1}{q_i(\kappa)}\right| h_{-i}, i \in S\right] q_i(\kappa_i') + 0\right] \\
=&\mathbb{E}_{h_{-i}}\left[\mathbb{E}_{h_i}\left[\left.\frac{1}{q_i(\kappa)}\right| \kappa = \kappa_i'\right] q_i(\kappa_i')\right] \\
=&\mathbb{E}_{h_{-i}}\left[\frac{1}{q_i(\kappa_i')} q_i(\kappa_i')\right] = \mathbb{E}_{h_{-i}}[1] = 1.
\end{aligned}$$

■

Theorem 5 *The importance weighted estimator is an unbiased estimator of* $\mathbb{E}[f(i)]$*:*

$$\mathbb{E}_{h_{1:n}}\left[\sum_{i \in S} \frac{p_i}{q_i(\kappa)} f(i)\right] = \mathbb{E}[f(i)].$$

Proof

$$\begin{aligned}
\mathbb{E}_{h_{1:n}}\left[\sum_{i \in S} \frac{p_i}{q_i(\kappa)} f(i)\right] &= \mathbb{E}_{h_{1:n}}\left[\sum_{i=1}^{n} \frac{p_i}{q_i(\kappa)} f(i) \mathbb{1}_{\{i \in S\}}\right] \\
&= \sum_{i=1}^{n} p_i f(i) \cdot \mathbb{E}_{h_{1:n}}\left[\frac{\mathbb{1}_{\{i \in S\}}}{q_i(\kappa)}\right] \\
&= \sum_{i=1}^{n} p_i f(i) \cdot 1 \\
&= \mathbb{E}[f(i)].
\end{aligned}$$

■

Appendix C. Numerical Stability of Importance Weights

We have to take care computing the importance weights as depending on the entropy the terms in the quotient $\frac{p_i}{q_i(\kappa)}$ (using notation from Appendix B) can become very small, and in our case the computation of $q_i(\kappa) = 1 - \exp(-\exp(\phi_i - \kappa))$ (Equation (22)) can suffer from catastrophic cancellation. We can rewrite this expression using the more numerically stable implementation $\text{expm1}(x) = \exp(x) - 1$ as $q_i(\kappa) = -\text{expm1}(-\exp(\phi_i - \kappa))$ but in some cases this still suffers from instability as $\exp(\phi_i - \kappa)$ can underflow if $\phi_i - \kappa$ is small. Instead, for $\phi_i - \kappa < -10$ we use the identity

$$\log(1 - \exp(-z)) = \log(z) - \frac{z}{2} + \frac{z^2}{24} - \frac{z^4}{2880} + \mathcal{O}(z^6)$$

to directly compute the log importance weight using $z = \exp(\phi_i - \kappa)$ and $\phi_i = \log p_i$ (we assume ϕ_i is normalized). Since $q_i(\kappa) = 1 - \exp(-z)$, we have

$$\begin{aligned}
\log\left(\frac{p_i}{q_i(\kappa)}\right) &= \log p_i - \log q_i(\kappa) \\
&= \log p_i - \log\left(1 - \exp(-z)\right) \\
&= \log p_i - \left(\log(z) - \frac{z}{2} + \frac{z^2}{24} - \frac{z^4}{2880} + \mathcal{O}(z^6)\right) \\
&= \log p_i - \left(\phi_i - \kappa - \frac{z}{2} + \frac{z^2}{24} - \frac{z^4}{2880} + \mathcal{O}(z^6)\right) \\
&= \kappa + \frac{z}{2} - \frac{z^2}{24} + \frac{z^4}{2880} + \mathcal{O}(z^6).
\end{aligned}$$

If $\phi_i - \kappa < -10$ then $0 < z < 10^{-6}$ so this computation will not lose any significant digits.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2015.

Matej Balog, Nilesh Tripuraneni, Zoubin Ghahramani, and Adrian Weller. Lost relatives of the Gumbel trick. In *International Conference on Machine Learning (ICML)*, pages 371–379, 2017.

Yutian Chen and Zoubin Ghahramani. Scalable discrete sampling as a multi-armed bandit problem. In *International Conference on Machine Learning (ICML)*, pages 2492–2501, 2016.

Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

Nick Duffield, Carsten Lund, and Mikkel Thorup. Learn more, sample less: control of volume and variance in network measurement. *Transactions on Information Theory*, 51 (5):1756–1775, 2005.

Nick Duffield, Carsten Lund, and Mikkel Thorup. Priority sampling for estimation of arbitrary subset sums. *Journal of the ACM (JACM)*, 54(6):32, 2007.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 489–500, 2018a.

Sergey Edunov, Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. Classical structured prediction losses for sequence to sequence learning. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 355–364, 2018b.

Pavlos S Efraimidis and Paul G Spirakis. Weighted random sampling with a reservoir. *Information Processing Letters*, 97(5):181–185, 2006.

Stefano Ermon, Carla P Gomes, Ashish Sabharwal, and Bart Selman. Embed and project: Discrete sampling with universal hashing. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2085–2093, 2013.

Paul Fearnhead and Peter Clifford. On-line inference for hidden Markov models via particle filters. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(4): 887–899, 2003.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning (ICML)*, pages 1243–1252, 2017.

Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures.* Number 33. US Govt. Print. Office, 1954.

Tamir Hazan and Tommi Jaakkola. On the partition function and random maximum a-posteriori perturbations. In *International Conference on Machine Learning (ICML)*, pages 1667–1674. Omnipress, 2012.

Tamir Hazan, Subhransu Maji, and Tommi Jaakkola. On sampling from the Gibbs distribution with random maximum a-posteriori perturbations. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1268–1276, 2013.

Timothy Classen Hesterberg. *Advances in importance sampling.* PhD thesis, Stanford University, 1988.

Daniel G Horvitz and Donovan J Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association*, 47(260): 663–685, 1952.

Arthur B Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11): 558–562, 1962.

Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 reinforce samples, get a baseline for free! In *Deep Reinforcement Learning Meets Structured Prediction Workshop at the International Conference on Learning Representations (ICLR)*, 2019a.

Wouter Kool, Herke van Hoof, and Max Welling. Stochastic beams and where to find them: The Gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning (ICML)*, pages 3499–3508, 2019b.

Wouter Kool, Herke van Hoof, and Max Welling. Estimating gradients for discrete random variables by sampling without replacement. In *International Conference on Learning Representations (ICLR)*, 2020.

Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

Jiwei Li, Will Monroe, and Dan Jurafsky. A simple, fast diverse decoding algorithm for neural generation. *arXiv preprint arXiv:1611.08562*, 2016.

Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc V Le, and Ni Lao. Memory augmented policy optimization for program synthesis and semantic parsing. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 9994–10006, 2018.

Runjing Liu, Jeffrey Regier, Nilesh Tripuraneni, Michael Jordan, and Jon Mcauliffe. Rao-Blackwellized stochastic gradients for discrete distributions. In *International Conference on Machine Learning (ICML)*, pages 4023–4031, 2019.

Guy Lorberbom, Chris J Maddison, Nicolas Heess, Tamir Hazan, and Daniel Tarlow. Direct policy gradients: Direct optimization of policies in discrete action spaces. *arXiv preprint arXiv:1906.06062*, 2019.

Robert Duncan Luce. Individual choice behavior. 1959.

Martin Mächler. Accurately computing $\log(1 - \exp(-|a|))$ assessed by the Rmpfr package, 2012. URL `https://cran.r-project.org/web/packages/Rmpfr/vignettes/log1mexp-note.pdf`.

Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3086–3094, 2014.

Renato Negrinho, Matthew Gormley, and Geoffrey J Gordon. Learning beam search policies via imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 10673–10682, 2018.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

George Papandreou and Alan L Yuille. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *International Conference on Computer Vision (ICCV)*, pages 193–200, 2011.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, 2002.

Robin L Plackett. The analysis of permutations. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 24(2):193–202, 1975.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3528–3536, 2015.

Yuanlong Shao, Stephan Gouws, Denny Britz, Anna Goldie, Brian Strope, and Ray Kurzweil. Generating high-quality and informative conversation responses with sequence-to-sequence models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2210–2219, 2017.

Kensen Shi, David Bieber, and Charles Sutton. Incremental sampling without replacement for sequence models. *arXiv preprint arXiv:2002.09067*, 2020.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3104–3112, 2014.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

Daniel Tarlow, Ryan Adams, and Richard Zemel. Randomized optimum models for structured prediction. In *Artificial Intelligence and Statistics*, pages 1221–1229, 2012.

Daniel Ting. Adaptive threshold sampling and estimation. *arXiv preprint arXiv:1708.04970*, 2017.

Tim Vieira. Gumbel-max trick and weighted reservoir sampling, 2014. URL `https://timvieira.github.io/blog/post/2014/08/01/gumbel-max-trick-and-weighted-reservoir-sampling/`.

Tim Vieira. Estimating means in a finite universe, 2017. URL `https://timvieira.github.io/blog/post/2017/07/03/estimating-means-in-a-finite-universe/`.

Ashwin K Vijayakumar, Michael Cogswell, Ramprasaath R Selvaraju, Qing Sun, Stefan Lee, David J Crandall, and Dhruv Batra. Diverse beam search for improved description of complex scenes. In *AAAI*, 2018.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3156–3164, 2015.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Sam Wiseman and Alexander M Rush. Sequence-to-sequence learning as beam-search optimization. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1296–1306, 2016.

Chak-Kuen Wong and Malcolm C. Easton. An efficient method for weighted sampling without replacement. *SIAM Journal on Computing*, 9(1):111–113, 1980.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

John I Yellott. The relationship between Luce's choice axiom, Thurstone's theory of comparative judgment, and the double exponential distribution. *Journal of Mathematical Psychology*, 15(2):109–144, 1977.