GFlowNet Foundations

Yoshua Bengio*   YOSHUA.BENGIO@MILA.QUEBEC
Mila, Universié de Montréal, CIFAR, IVADO

Salem Lahlou*   LAHLOSAL@MILA.QUEBEC
Mila, Universié de Montréal

Tristan Deleu*   DELEUTRI@MILA.QUEBEC
Mila, Universié de Montréal

Edward J. Hu   EDWARD@EDWARDJHU.COM
Mila, Universié de Montréal, Microsoft

Mo Tiwari   MOTIWARI@STANFORD.EDU
Stanford University

Emmanuel Bengio   BENGIOEM@MILA.QUEBEC
Mila, McGill University

Editor: David Sontag

Abstract

Generative Flow Networks (GFlowNets) have been introduced as a method to sample a diverse set of candidates in an active learning context, with a training objective that makes them approximately sample in proportion to a given reward function. In this paper, we show a number of additional theoretical properties of GFlowNets, including a new local and efficient training objective called detailed balance for the analogy with MCMC. GFlowNets can be used to estimate joint probability distributions and the corresponding marginal distributions where some variables are unspecified and, of particular interest, can represent distributions over composite objects like sets and graphs. GFlowNets amortize the work typically done by computationally expensive MCMC methods in a single but trained generative pass. They could also be used to estimate partition functions and free energies, conditional probabilities of supersets (supergraphs) given a subset (subgraph), as well as marginal distributions over all supersets (supergraphs) of a given set (graph). We introduce variations enabling the estimation of entropy and mutual information, continuous actions and modular energy functions.

1. Introduction

Building upon the introduction of Generative Flow Networks (GFlowNets) by Bengio et al. (2021), we provide here an in-depth formal foundation and expansion of the set of theoretical results in ways that may be of interest for the active learning scenario of Bengio et al. (2021) but also much more broadly.

---

*. Equal Contribution

1.1 What is a GFlowNet ?

GFlowNets have properties which make them well-suited to perform amortized probabilistic inference in general, whether for sampling or for marginalizing. Sampling takes place at training time while run-time sampling or computations of marginalized quantities can be done in a single pass through a sequence of constructive stochastic steps. This makes GFlowNets an interesting alternative to Monte-Carlo Markov chains (MCMC) and related to amortized variational inference (Malkin et al., 2023).

Because sampling of a compositional object s can be achieved through a sequence of stochastic steps, very rich multimodal distributions $P_T(s)$ over such objects can be represented, and the offline training objectives make it possible to explore and discover modes of the distribution of interest. The key property of GFlowNets is that their sampling policy is trained to make the probability $P_T(s)$ of sampling an object s approximately proportional to the value $R(s)$ of a given reward function applied to that object. We also talk of an energy function $\mathcal{E}(s) = -\log R(s)$, i.e., the reward function is non-negative and corresponds to an unnormalized probability. Whereas one typically trains a generative model from a dataset of positive examples, a GFlowNet is trained to match the given energy or reward function and convert it into a sampler. We view that sampler as a generative policy because the composite object s is constructed through a sequence of smaller stochastic steps (see Fig. 1), often corresponding to constructively composing different elements of s, like the edges of a graph.

This conversion of an energy function or unnormalized probability function to a sampler is similar to what MCMC methods achieve but once trained, GFlowNets will generate a sample in one shot instead of generating a long sequence of samples whose distribution would gradually approach the desired one. GFlowNets thus avoid the lengthy stochastic search in the space of such objects and the associated mode-mixing intractability challenge of MCMC methods (Jasra et al., 2005; Bengio et al., 2013; Pompe et al., 2020). Multiple iid samples can be obtained from the GFlowNet by calling the sampler multiple times. GFlowNets exchange that intractability of sampling with MCMC for the challenge of amortized training of the generative policy. The latter problem would be equally intractable if the modes of the reward function did not have a inherent (but not necessarily known) structure over which the learner could generalize, i.e., the learner had almost no chance to correctly guess where to find new modes based on (i.e., training on) those it had already visited.

The energy function or reward function (exponential of minus energy) is evaluated only at the end of the sequential construction process for objects s, in what we call a terminating state. Every such constructive sequence starts in the single initial state s_0 and ends in a terminal state. As illustrated in Figure 2, we can visualize the set of all trajectories starting from s_0 and ending in a terminal state s. The term "flow" in "generative flow networks" refers to unnormalized probabilities that can be learned by GFlowNet learning procedures. The flow in an intermediate state s is a weighted sum of the non-negative rewards of the terminating states reachable from s. Those weights are such as to avoid double-counting: if we were to inject a fixed flow of liquid in s_0 and dispatch that liquid in each child of any state s proportionally to the GFlowNet policy for choosing a child of s, we would obtain the flow at each state and the flow at terminating states would match the reward function at those states. As shown in greater detail here and for the first time in the first GFlowNet
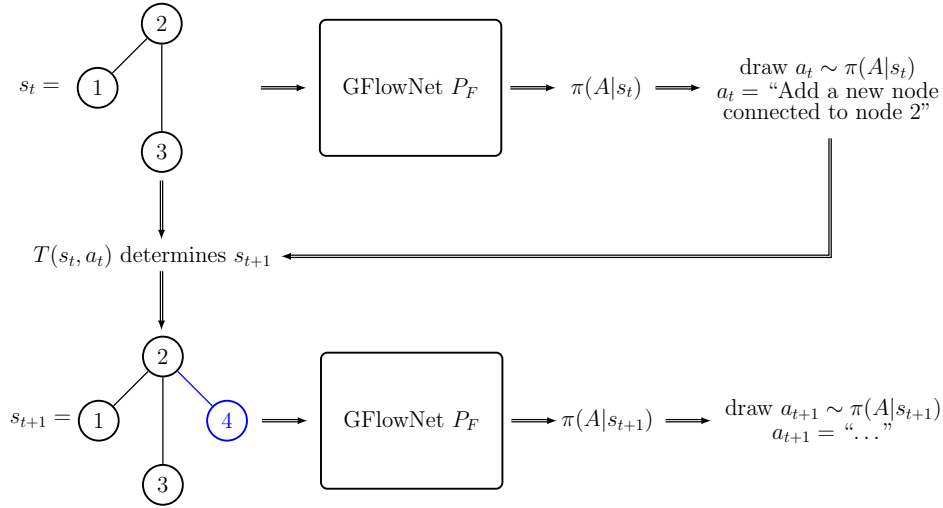
Figure 1: A diagram of how a GFlowNet iteratively constructs an object. We adopt notation that is common in the reinforcement learning literature: s_t represents the state of the partially constructed object (in this case, a graph) at time t, a_t represents the action taken by the GFlowNet at time t to transition to state $s_{t+1} = T(s_t, a_t)$. In this diagram, the GFlowNet takes a 3-node graph as input and determines an action to take. The action, combined with the environment transition function $T(s_t, a_t)$, determines s_{t+1}: a four-node graph. This process repeats until an exit action is sampled and the sample is complete.

paper (Bengio et al., 2021), this can be achieved with a flow constraint at each state: the sum of incoming flows must match the sum of outgoing flows.

1.2 Contributions of this paper

In this paper, an important contribution is the notion of *conditional* GFlowNet, which enables estimation of intractable sums corresponding to marginalization over many steps of object construction, and can thus be used to compute free energies[1] over different types of joint distributions, perhaps most interestingly over sets and graphs. This marginalization also enables estimation of entropies, conditional entropies and mutual information. GFlowNets can thus be generalized to estimate multiple flows corresponding to modeling a rich outcome (rather than a scalar reward function) .

We refer the reader to Bengio et al. (2021) and Sec. 7 for a discussion of related approaches and differences with common generative models and reinforcement learning (RL) methods. In an RL context, two interesting properties of GFlowNets already noted in that paper are that they (1) can be trained in an offline manner with trajectories sampled from a distribution different from the one represented by the GFlowNet and (2) they match the reward function in probability rather than try to find a configuration which maximizes rewards or returns. The latter property is particularly interesting in the context of exploration, to ensure the configurations sampled from the generative policy are both interesting and di-

---

1. In machine learning, a free energy is the logarithm of an unnormalized marginal probability, a generally intractable sum of exponentiated negative energies.
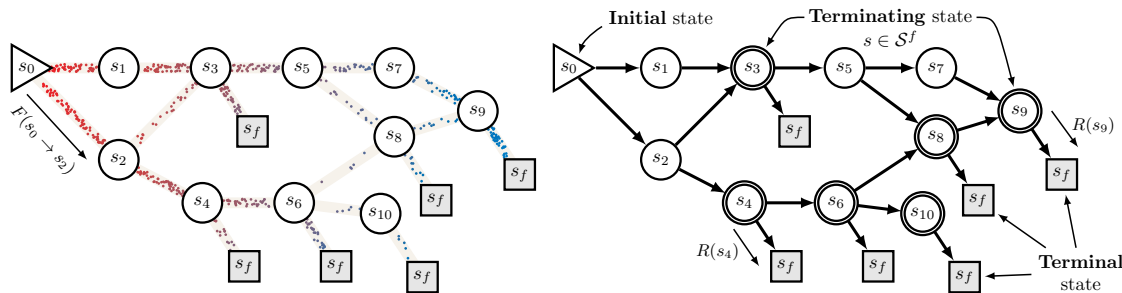
Figure 2: Illustration of the structure of a Generative Flow Network (GFlowNet), as a pointed DAG over states s, with particles flowing along edges to represent the flow function. Any object sampled by the GFlowNet policy can be obtained by starting from initial state s_0 and then at each step choosing a child with probability proportional to the GFlowNet policy's transition probability. This process stops when a terminating action is chosen from a terminating state s (yielding a terminal state s_f), at which point a reward $R(s)$ is obtained. The figure shows a tiny GFlowNet and the possible trajectories from s_0 to any of the terminal states. It illustrates that in general a state can be reached through several trajectories. GFlowNet algorithms learn a policy such that the probability of sampling terminating state s is proportional to $R(s)$. It tries to learn a flow function $F(s)$ and $F(s \rightarrow s')$ over all states (including intermediate states) s and transitions $s \rightarrow s'$ with $F(s) = R(s)$ at terminal states and $F(s_0)$ being the sum of rewards over all terminal states. A sufficient property to achieve this is that at each state the sum of incoming flows equals the sum of outgoing flows.

verse. It is also interesting to transform GFlowNets into amortized probabilistic inference machines: if we choose the reward function to be a prior (over some random variable) times a likelihood (how well some data is fit given that choice of random variable value), then the GFlowNet policy learns to sample from the corresponding Bayesian posterior (which is proportional to prior times likelihood). The ability of GFlowNets to generate a diverse set of samples then corresponds to the ability to sample from the modes of the target distribution.

An important source of inspiration for GFlowNets is the way information propagates in temporal-difference RL methods (Sutton and Barto, 2018). Both rely on a principle of coherence for credit assignment which may only be achieved asymptotically when training converges. While exact gradient calculation may be intractable, because the number of paths in state space to consider is exponentially large, both methods rely on local coherence between different components and a training objective that states that if all the learned components are coherent with each other locally, then we obtain a system that estimates the quantities of interest globally. Examples include estimation of expected discounted returns in temporal-difference methods and probability measures with GFlowNets.

This paper extends the theory of the original GFlowNet construction (Bengio et al., 2021) in several directions, including a new local training objective called detailed balance (for the analogy with the detailed balance condition of Monte-Carlo Markov chains) which avoids forming explicit sums required by the previously proposed flow matching loss, as well as formulations enabling the calculation of marginal probabilities (or free energies)

for subsets of variables, more generally for subsets of larger sets, or subgraphs, and their application to estimating entropy and mutual information. Finally, whereas the basic formulation of GFlowNets assumes a given reward or energy function, this paper considers how the energy function could be jointly learned with the GFlowNet, opening the door to novel energy-based modeling methodologies and a modular structure for both the energy function and the GFlowNet.

1.3 GFlowNets in other works

In addition to the theory presented in this paper, Malkin et al. (2023) and Zimmermann et al. (2022) prove some partial equivalences between GFlowNets and hierarchical variational methods, providing yet more theoretical evidence for the efficacy of GFlowNets in learning to sample proportionally to a given reward function. These works also provide evidence for the superiority of GFlowNets in off-policy settings.

GFlowNets have found a wide array of applications due to the associated diversity of generated samples. In contexts where a cheap proxy for the true reward function exists, GFlowNets have been used to surface samples under which to query the proxy before more expensive evaluation under the true reward function. In these settings, the diversity of samples generated by GFlowNets can be used for robustness to proxy misspecification and to incorporate epistemic uncertainty. For example, Zhang et al. (2023) use GFlowNets to produce sample schedules for operations in a computation graph, where evaluating the runtimes of sample schedules via a proxy is fast but evaluating the same schedules on target hardware is expensive. In active learning problems, Jain et al. (2022, 2023) use GFlowNet sampling as a subroutine inside an active learning loop as a substitute for Bayesian Optimization or RL-based methods. Jain et al. (2022) apply GFlowNets to search for novel anti-microbial peptides, discover DNA sequences that have high binding activity with human transcription factors, and to find proteins with high fluorescence. Additionally, Jain et al. (2023) develops preference-conditional GFlowNets, where a preference weight vector is used to scalarize multiple objective functions into a single reward. The authors apply their techniques to various molecule and DNA sequence generation tasks and find that their methods are able to find different Pareto-optimal samples along the Pareto frontier.

GFlowNets have found applications in several other machine learning problems. For example, Zhang et al. (2022) simultaneously train an energy-based model and a GFlowNet; the energy function is trained with samples from a GFlowNet, which, in turn, uses the energy function to form its reward. Their method results in a generative model for binary vectors in high dimensions, e.g., binarized digits. Deleu et al. (2022) use a GFlowNet for structure learning; the GFlowNet produces samples that approximates the true posterior over causal graphs given a dataset. Their method works on both observational and interventional data, and compares favorably to MCMC- and variational inference-based methods. Hu et al. (2023) find maximum-likelihood estimates of latent variable models with discrete compositional latents by jointly training a GFlowNet to approximately sample from the generally intractable posterior in the E-step of the expectation-maximization (EM) algorithm.

2. Flow Networks and Markovian Flows

2.1 Some elements of graph theory

In this section, we recall some basic definitions and properties of graphs, which are the basis of flow networks and GFlowNets.

Definition 1 *A directed graph is a tuple $G = (\mathcal{S}, \mathbb{A})$, where $\mathcal{S}$ is a finite set of states, and $\mathbb{A}$ a subset of $\mathcal{S} \times \mathcal{S}$ representing directed edges. Elements of $\mathbb{A}$ are denoted $s \to s'$ and called* **edges** *or* **transitions**.
A **trajectory** *in such a graph is a sequence $\tau = (s_1, \ldots, s_n)$ of elements of $\mathcal{S}$ such that every transition $s_t \to s_{t+1} \in \mathbb{A}$ and $n > 1$. We denote $s \in \tau$ to mean that s is in the trajectory τ, i.e., $\exists t \in \{1, \ldots, n\}$ $s_t = s$, and similarly $s \to s' \in \tau$ to mean that $\exists t \in \{1, \ldots, n-1\}$ $s_t = s, s_{t+1} = s'$. For convenience, we also use the notation $\tau = s_1 \to \cdots \to s_n$. The* **length** *of a trajectory is the number of edges in it (the length of $\tau = (s_1, \ldots, s_n)$ is thus $n - 1$).*
A **directed acyclic graph** *(DAG) is a directed graph in which there is no trajectory $\tau = (s_1, \ldots, s_n)$ satisfying $s_n = s_1$.*

Given a DAG $G = (\mathcal{S}, \mathbb{A})$, and two states $s, s' \in \mathcal{S}$, if there exists a trajectory in G starting in s and ending in s', then we write $s < s'$. The binary relationship "$<$" defines a **strict partial order** (i.e. it is irreflexive, asymmetric and transitive). We write $s \leq s'$ if $s < s'$ or $s = s'$. The binary relation "$\leq$" is a (non-strict) **partial order** (i.e. it is reflexive, antisymmetric and transitive).

If there is no order relation between s and s', we write $s \lessgtr s'$.

Definition 2 *Given a DAG $G = (\mathcal{S}, \mathbb{A})$, the* **parent set** *of a state $s \in \mathcal{S}$, which we denote $Par(s)$, contains all of the direct parents of s in G, i.e., $Par(s) = \{s' \in \mathcal{S} \ : \ s' \to s \in \mathbb{A}\}$; similarly, the* **child set** *$Child(s)$ contains all of the direct children of s in G, i.e., $Child(s) = \{s' \in \mathcal{S} \ : \ s \to s' \in \mathbb{A}\}$.*

Definition 3 *Given a DAG $G = (\mathcal{S}, \mathbb{A})$. G is called a* **pointed DAG** *if there exist two states $s_0, s_f \in \mathcal{S}$ that satisfy:*

$$\forall s \in \mathcal{S} \setminus \{s_0\} \ \ s_0 < s \text{ and } \forall s \in \mathcal{S} \setminus \{s_f\} \ \ s < s_f.$$

s_0 is called the **source state** *or* **initial state**. *s_f is called the* **sink state** *or* **final state**. *Because "$<$" is a strict partial order, these two states are unique.*

A **complete trajectory** *in such a DAG is any trajectory starting in s_0 and ending in s_f. We denote such a trajectory as $\tau = (s_0, s_1, \ldots, s_n, s_{n+1} = s_f)$.*

We denote by $\mathcal{T}$ *the set of all complete trajectories in G, and by $\mathcal{T}^{partial}$ the set of (possibly incomplete) trajectories in G.*

A state $s \in \mathcal{S}$ *is called a* **terminating state** *if it is a parent of the sink state, i.e. $s \to s_f \in \mathbb{A}$. The transition $s \to s_f$ is called a* **terminating edge**. *We denote by:*

- $\mathbb{A}^{-f} = \{s \to s' \in \mathbb{A}, \ s' \neq s_f\}$, *the set of non-terminating edges in G,*

- $\mathbb{A}^f = \{s \to s' \in \mathbb{A}, \ s' = s_f\} = \mathbb{A} \setminus \mathbb{A}^{-f}$, *the set of terminating edges in G,*

- $\mathcal{S}^f = \{s \in \mathcal{S}, \ s \to s_f \in \mathbb{A}^f\} = Par(s_f)$, *the set of terminating states in G.*
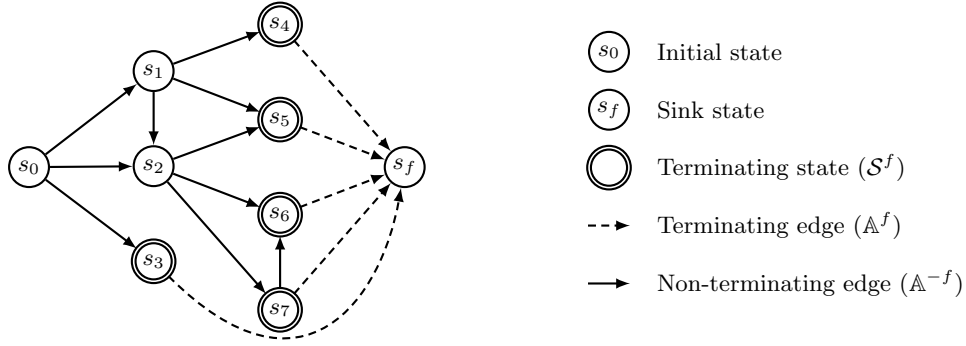
6

Figure 3: Example of a pointed DAG G illustrating the notions of initial state (s_0), final or sink state (s_f), terminating states in $\mathcal{S}^f$, with a transition to s_f called a terminating edge, in $\mathbb{A}^f$. A terminating state may have other children different from the sink state (e.g., the terminating state s_7).

In Fig. 3, we visualize the concepts introduced in the previous definitions.

Note that the constraint of a single source state and single sink state is only a mathematical convenience since a bijection exists between general DAGs and those with this constraint (by the addition of a unique source/sink state connected to all the other source/sink states).

Definition 4 *Let G be a pointed DAG with source state s_0 and sink state s_f. A **forward** (resp. **backward) probability function consistent with** G is any non-negative function $\hat{P}_F$ (resp. $\hat{P}_B$) defined on $\mathbb{A}$ that satisfies $\forall s \in \mathcal{S} \setminus \{s_f\}$, $\sum_{s' \in Child(s)} \hat{P}_F(s' \mid s) = 1$ (resp. $\forall s \in \mathcal{S} \setminus \{s_0\}, \sum_{s' \in Par(s)} \hat{P}_B(s' \mid s) = 1$).*

With pointed DAGs, consistent forward and backward probability functions, that are probabilities over states, can be used to define probabilities over trajectories, i.e. probability measures on some subsets of $\mathcal{T}^{partial}$. The following lemma shows how to construct such factorized probability measures:

Lemma 5 *Let $G = (\mathcal{S}, \mathbb{A})$ be a pointed DAG, and consider a forward probability function $\hat{P}_F$, and a backward probability function $\hat{P}_B$ both consistent with G. For any state $s \in \mathcal{S} \setminus \{s_f\}$, we denote by $\mathcal{T}_{s,f} \subseteq \mathcal{T}^{partial}$ the set of trajectories in G starting in s and ending in s_f; and for any state $s \in \mathcal{S} \setminus \{s_0\}$, we denote by $\mathcal{T}_{0,s} \subseteq \mathcal{T}^{partial}$ the set of trajectories in G starting in s_0 and ending in s.*

Consider the extensions of $\hat{P}_F$ and $\hat{P}_B$ on $\mathcal{T}^{partial}$ defined by:

$$\forall \tau = (s_1, \ldots, s_n) \in \mathcal{T}^{partial} \quad \hat{P}_F(\tau) := \prod_{t=1}^{n-1} \hat{P}_F(s_{t+1} \mid s_t) \tag{1}$$

$$\forall \tau = (s_1, \ldots, s_n) \in \mathcal{T}^{partial} \quad \hat{P}_B(\tau) := \prod_{t=1}^{n-1} \hat{P}_B(s_t \mid s_{t+1}) \tag{2}$$

We have the following:

$$\forall s \in \mathcal{S} \setminus \{s_f\} \quad \sum_{\tau \in \mathcal{T}_{s,f}} \hat{P}_F(\tau) = 1 \tag{3}$$

$$\forall s' \in \mathcal{S} \setminus \{s_0\} \quad \sum_{\tau \in \mathcal{T}_{0,s'}} \hat{P}_B(\tau) = 1 \tag{4}$$

Proof For convenience, we will use $\mathcal{T}_{s \to s', s_f}$ to denote the set of trajectories starting with $s \to s'$ and ending in s_f, and $\mathcal{T}_{0, s \to s'}$ to denote the set of trajectories starting in s_0 and ending with $s \to s'$. This allows to write:

$$\forall s \neq s_f \quad \mathcal{T}_{s,f} = \bigcup_{s' \in Child(s)} \mathcal{T}_{s \to s', s_f}, \quad \{\mathcal{T}_{s \to s', s_f}, \ s' \in Child(s)\} \text{ pairwise disjoint,}$$

$$\forall s' \neq s_0 \quad \mathcal{T}_{0,s'} = \bigcup_{s \in Par(s')} \mathcal{T}_{0, s \to s'}, \quad \{\mathcal{T}_{0, s \to s'}, \ s \in Par(s')\} \text{ pairwise disjoint.}$$

Additionally, for any $s \neq s_f$, we denote by $d_{s,f}$ the maximum trajectory length in $\mathcal{T}_{s,f}$; and for any $s' \neq s_0$, we denote by $d_{0,s'}$ the maximum trajectory length in $\mathcal{T}_{0,s}$.

We will prove Eq. (3) by strong induction on $d_{s,f}$ and Eq. (4) by strong induction on $d_{0,s'}$.

Base cases: If $d_{s,f} = 1$ and $d_{0,s'} = 1$, then $\mathcal{T}_{s,f} = \{(s \to s_f)\}$ and $\mathcal{T}_{0,s'} = \{(s_0 \to s')\}$. Hence, $\sum_{\tau \in \mathcal{T}_{s,f}} \hat{P}_F(\tau) = \hat{P}_F(s \to s_f) = \hat{P}_F(s_f \mid s) = 1$ given that s_f is the only child of s (otherwise $d_{s,f}$ cannot be 1), and $\sum_{\tau \in \mathcal{T}_{0,s'}} \hat{P}_B(\tau) = \hat{P}_B(s_0 \mid s') = 1$ given that s_0 is the only parent of s' (otherwise $d_{0,s'}$ cannot be 1).

Induction steps: Consider $s \neq s_f$ such that $d_{s,f} > 1$ and $s' \neq s_0$ such that $d_{0,s'} > 1$. Because of the disjoint unions written above, we have:

$$\sum_{\tau \in \mathcal{T}_{s,f}} \hat{P}_F(\tau) = \sum_{\tilde{s} \in Child(s)} \sum_{\tau \in \mathcal{T}_{s \to \tilde{s}, f}} \hat{P}_F(\tau) = \sum_{\tilde{s} \in Child(s)} \hat{P}_F(\tilde{s} \mid s) \sum_{\tau \in \mathcal{T}_{\tilde{s}, f}} \hat{P}_F(\tau) = 1,$$

$$\sum_{\tau \in \mathcal{T}_{0,s'}} \hat{P}_B(\tau) = \sum_{\tilde{s}' \in Par(s')} \sum_{\tau \in \mathcal{T}_{0, \tilde{s}' \to s}} \hat{P}_B(\tau) = \sum_{\tilde{s}' \in Par(s')} \hat{P}_B(\tilde{s}' \mid s') \sum_{\tau \in \mathcal{T}_{0, \tilde{s}'}} \hat{P}_B(\tau) = 1,$$

where we used the induction hypotheses in the third equality of each line. ∎

2.2 Trajectories and Flows

We augment pointed DAGs it with a function F called a *flow*. An analogy which helps to picture flows is a stream of particles flowing through a network where each particle starts at s_0 and flowing through some trajectory terminating in s_f. The flow $F(\tau)$ associated with each complete trajectory τ contains the number of particles sharing the same path τ.

Definition 6 *Given a pointed DAG, a **trajectory flow** (or "**flow**") is any non-negative function $F : \mathcal{T} \mapsto \mathbb{R}^+$ defined on the set of* complete trajectories $\mathcal{T}$. *F induces a* measure

over the σ-algebra $\Sigma = 2^{\mathcal{T}}$, the power set on the set of complete trajectories $\mathcal{T}$. In particular, for every subset $A \subseteq \mathcal{T}$, we have

$$F(A) = \sum_{\tau \in A} F(\tau). \tag{5}$$

The pair (G, F) is called a **flow network**.

This definition ensures that $(\mathcal{T}, 2^{\mathcal{T}}, F)$ is a measure space. We abuse the notation here, using F to denote both a function of complete trajectories, and its corresponding measure over $(\mathcal{T}, 2^{\mathcal{T}})$. A special case is when the event A is the singleton trajectory $\{\tau\}$, where we just write its measure as $F(\tau)$. We also abuse the notation to define the flow through either a particular state s, or through a particular edge $s{\rightarrow}s'$ in the following way.

Definition 7 *The* **flow through a state** *(or state flow) $F : \mathcal{S} \mapsto \mathbb{R}^+$ corresponds to the measure of the set of complete trajectories going through a particular state:*

$$F(s) := F(\{\tau \in \mathcal{T} \, : \, s \in \tau\}) = \sum_{\tau \in \mathcal{T} \, : \, s \in \tau} F(\tau). \tag{6}$$

Similarly, the **flow through an edge** *(or edge flow) $F : \mathbb{A} \mapsto \mathbb{R}^+$ corresponds to the measure of the set of complete trajectories going through a particular edge:*

$$F(s{\rightarrow}s') := F(\{\tau \in \mathcal{T} \, : \, s{\rightarrow}s' \in \tau\}) = \sum_{\tau \in \mathcal{T} \, : \, s{\rightarrow}s' \in \tau} F(\tau). \tag{7}$$

Note that with this definition, we have $F(s{\rightarrow}s') = 0$ if $s{\rightarrow}s' \notin \mathbb{A}$ is not an edge in the pointed DAG (since $F(\emptyset) = 0$). We call the flow of a terminating transition $F(s{\rightarrow}s_f)$ a *terminating flow*. The following proposition relates the state flows and the edge flows:

Proposition 8 *Given a flow network (G, F). The state flows and edge flows satisfy:*

$$\forall s \in \mathcal{S} \setminus \{s_f\} \quad F(s) = \sum_{s' \in Child(s)} F(s \rightarrow s') \tag{8}$$

$$\forall s' \in \mathcal{S} \setminus \{s_0\} \quad F(s') = \sum_{s \in Par(s')} F(s \rightarrow s') \tag{9}$$

Proof Given $s \neq s_f$, the set of complete trajectories going through s is the (disjoint) union of the sets of trajectories going through $s \rightarrow s'$, for all $s' \in Child(s)$:

$$\{\tau \in \mathcal{T} \, : \, s \in \tau\} = \bigcup_{s' \in Child(s)} \{\tau \in \mathcal{T} \, : \, s \rightarrow s' \in \tau\}.$$

Therefore, it follows that:

$$F(s) = \sum_{\tau \, : \, s \in \tau} F(\tau) = \sum_{s' \in Child(s)} \sum_{\tau \, : \, s \rightarrow s' \in \tau} F(\tau) = \sum_{s' \in Child(s)} F(s \rightarrow s')$$

Similarly, Eq. (9) follows by writing the set of complete trajectories going though $s' \neq s_0$ as the (disjoint) union of the sets of trajectories going through $s \rightarrow s'$ for all $s \in Par(s')$. ∎

2.3 Flow Induced Probability Measures

Definition 9 *Given a flow network (G, F), the* **total flow** Z *is the measure of the whole set $\mathcal{T}$, corresponding to the sum of the flows of all the complete trajectories:*

$$Z := F(\mathcal{T}) = \sum_{\tau \in \mathcal{T}} F(\tau). \tag{10}$$

Proposition 10 *The flow through the initial state equals the flow through the final state equals the total flow Z.*

Proof Since $\forall \tau \in \mathcal{T}$, $s_0, s_f \in \tau$, applying Eq. (6) to s_0 and s_f yields

$$F(s_0) = \sum_{\tau \in \mathcal{T}} F(\tau) = Z, \tag{11}$$

$$F(s_f) = \sum_{\tau \in \mathcal{T}} F(\tau) = Z. \tag{12}$$

$\blacksquare$

Intuitively, Prop. 10 justifies the use of the term "flow", introduced by Bengio et al. (2021), by analogy with a stream of particles flowing from the initial state to the final states.

We use the letter Z in Def. 9, often used to denote the partition function in probabilistic models and statistical mechanics, because it is a normalizing constant which can turn the measure space $(\mathcal{T}, 2^{\mathcal{T}}, F)$ defined above into the probability space $(\mathcal{T}, 2^{\mathcal{T}}, P)$:

Definition 11 *Given a flow network (G, F), the **flow probability** is the probability measure P over the measurable space $(\mathcal{T}, 2^{\mathcal{T}})$ associated with F:*

$$\forall A \subseteq \mathcal{T} \quad P(A) := \frac{F(A)}{F(\mathcal{T})} = \frac{F(A)}{Z}. \tag{13}$$

For two events $A, B \subseteq \mathcal{T}$, the conditional probability $P(A \mid B)$ thus satisfies:

$$P(A \mid B) := \frac{F(A \cap B)}{F(B)}. \tag{14}$$

Similar to the flow F, we abuse the notation P to define the probability of going through a state:

$$\forall s \in \mathcal{S} \qquad P(s) := \frac{F(s)}{Z}, \tag{15}$$

and similarly for the probability of going through an edge. Note that $P(s)$ *does not* correspond to a distribution over states, in the sense that $\sum_{s \in \mathcal{S}} P(s) \neq 1$; in particular, it is easy to see that $P(s_0) = 1$ (in other words, the probability of a trajectory passing through the initial state s_0 is 1). Additionally, for a trajectory $\tau \in \mathcal{T}$, we also use the abuse of notation $P(\tau)$ instead of $P(\{\tau\})$ to denote the probability of going through a specific trajectory τ.

Definition 12 *Given a flow network (G, F), the* **forward transition probability** *operator P_F is a function on $\mathcal{S} \times \mathcal{S}$, that is a special case of the conditional probabilities induced by F (Eq. (14)):*

$$\forall s \to s' \in \mathbb{A} \quad P_F(s' \mid s) := P(s \to s' \mid s) = \frac{F(s \to s')}{F(s)}. \tag{16}$$

Similarly, the **backwards transition probability** *is the operator defined by:*

$$\forall s \to s' \in \mathbb{A} \quad P_B(s \mid s') := P(s \to s' \mid s') = \frac{F(s \to s')}{F(s')}. \tag{17}$$

Note how P_F and P_B are consistent with G (in the sense of Def. 4), as a consequence of Prop. 8.

Because flows define probabilities over states and edges, they can be used to define probability distributions over the terminating states of a graph (denoted by $\mathcal{S}^f = Par(s_f)$) as follows:

Definition 13 *Given a flow network (G, F), the* ***terminating state probability*** *P_T is the probability over terminating states $\mathcal{S}^f$ under the flow probability P:*

$$\forall s \in \mathcal{S}^f \quad P_T(s) := P(s \to s_f) = \frac{F(s \to s_f)}{Z} \tag{18}$$

Contrary to the probability $P(s)$ of going through a state s, the terminating state probability P_T is a well-defined distribution over the terminating states $s \in \mathcal{S}^f$, in the following sense:

Proposition 14 *The terminating state probability P_T is a well-defined distribution over the terminating states $s \in \mathcal{S}^f$, in that $P_T(s) \geq 0$ for all $s \in \mathcal{S}^f$, and*

$$\sum_{s \in \mathcal{S}^f} P_T(s) = 1.$$

Proof Since the flow $F(s \to s_f)$ is non-negative, it is easy to see that $P_T(s) \geq 0$. Moreover, using the definition of $\mathcal{S}^f = Par(s_f)$, Prop. 8 (relating the edge flows and the state flows), and Prop. 10 ($F(s_f) = Z$), we have

$$\sum_{s \in \mathcal{S}^f} P_T(s) = \frac{1}{Z} \sum_{s \in \mathcal{S}^f} F(s \to s_f) = \frac{1}{Z} \sum_{s \in Par(s_f)} F(s \to s_f) = \frac{F(s_f)}{Z} = 1.$$

∎

The terminating state probability is particularly important in the context of estimating flow networks (see Sec. 3), as it shows that a flow network (G, F) induces a probability distribution over terminating states which is proportional to the terminating flows $F(s \to s_f)$, the normalization constant Z being given by initial flow $F(s_0)$.

2.4 Markovian Flows

Defining a flow requires the specification of $|\mathcal{T}|$ non-negative values (one for every trajectory $\tau \in \mathcal{T}$), which is generally exponential in the number of graph edges. Markovian flows however have the remarkable property that they can be defined with much fewer "numbers", given that trajectory flows factorize according to G.

Definition 15 *Let (G, F) be a flow network, with flow probability measure P. F is called a **Markovian flow** (or equivalently (G, F) a **Markovian flow network**) if, for any state $s \neq s_0$, outgoing edge $s \to s'$, and for any trajectory $\tau = (s_0, s_1, \ldots, s_n = s) \in \mathcal{T}^{partial}$ starting in s_0 and ending in s:*

$$P(s \to s' \mid \tau) = P(s \to s' \mid s) = P_F(s' \mid s). \tag{19}$$

Note that the Markovian property does not hold for all of the flows as defined in the previous sections (e.g. Fig. 4). Intuitively, a flow can be considered non-Markovian if a particle in the "flow stream" can remember its past history; if not, its future behavior can only depend on its current state and the flow must be Markovian. In this work, we will primarily be concerned with Markovian flows, though later we will re-introduce a form of memory via state-conditional flows that allow each flow "particle" to remember parts of its history. The following proposition shows that Markovian flows have the property that the flows at (or the probabilities of) complete trajectories factorize according the the graph, and that it is a sufficient condition for defining Markovian flows.

Proposition 16 *Let (G, F) be a flow network, and P the corresponding flow probability. The following three statements are equivalent:*

1. *F is a Markovian flow*

2. *There exists a unique probability function $\hat{P}_F$ consistent with G such that for all complete trajectories $\tau = (s_0, \ldots, s_{n+1} = s_f)$:*

$$P(\tau) = \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1}). \tag{20}$$

   Moreover, the probability function $\hat{P}_F$ is exactly the forward transition probability associated with the flow probability P: $\hat{P}_F = P_F$.

3. *There exists a unique probability function $\hat{P}_B$ consistent with G such that for all complete trajectories $\tau = (s_0, \ldots, s_{n+1} = s_f)$:*

$$P(\tau) = \prod_{t=1}^{n+1} \hat{P}_B(s_{t-1} \mid s_t). \tag{21}$$

   Moreover, the probability function $\hat{P}_B$ is exactly the backwards transition probability associated with the flow probability P: $\hat{P}_B = P_B$.

Proof Recall from Lemma 5 the notations $\mathcal{T}_{0,s}$ to denote the set of partial trajectories from s_0 to s, and $\mathcal{T}_{s',f}$ to denote the set of partial trajectories from s' to s_f. We will prove the equivalences $1 \Leftrightarrow 2$ and $1 \Leftrightarrow 3$.

- $1 \Rightarrow 2$: Suppose that F is a Markovian flow. Then using the laws of probability, the Markov property in Eq. (19), and $P(s_0) = 1$, for some complete trajectory $\tau = (s_0, \ldots, s_{n+1} = s_f)$:

$$P(\tau) = P(s_0 \to s_1 \to \ldots \to s_{n+1}) = P(s_0 \to s_1) \prod_{t=1}^{n} P(s_t \to s_{t+1} \mid s_0 \to \ldots \to s_t)$$

$$= P(s_0 \to s_1) \prod_{t=1}^{n} P(s_t \to s_{t+1} \mid s_t)$$

$$= P(s_0) P_F(s_1 \mid s_0) \prod_{t=1}^{n} P_F(s_{t+1} \mid s_t)$$

$$= \prod_{t=1}^{n+1} P_F(s_t \mid s_{t-1}),$$

where the second line uses to Markov property, and the third line uses the definition of the forward transition probability P_F. P_F thus satisfies Eq. (20) for all complete trajectories.

To show uniqueness of P_F, assume Eq. (20) is satisfied by some $\hat{P}_F$ for all complete trajectories. By definition of the forward transition probability:

$$P_F(s' \mid s) := P(s \to s' \mid s) = \frac{P(s \to s')}{P(s)}.$$

Any complete trajectory τ going through a state s can be (uniquely) decomposed into a partial trajectory $\tau' \in \mathcal{T}_{0,s}$ from s_0 to s, and a partial trajectory $\tau'' \in \mathcal{T}_{s,f}$ from s to s_f. Using the definition of $P(s)$, we have:

$$P(s) = \sum_{\tau \,:\, s \in \tau} P(\tau) = \sum_{\tau \,:\, s \in \tau} \prod_{(s_t \to s_{t+1}) \in \tau} \hat{P}_F(s_{t+1} \mid s_t)$$

$$= \left[\sum_{\tau' \in \mathcal{T}_{0,s}} \prod_{(s_t \to s_{t+1}) \in \tau'} \hat{P}_F(s_{t+1} \mid s_t) \right] \underbrace{\left[\sum_{\tau'' \in \mathcal{T}_{s,f}} \prod_{(s_t \to s_{t+1}) \in \tau''} \hat{P}_F(s_{t+1} \mid s_t) \right]}_{= 1 \quad \text{(Lemma 5)}}$$

$$= \sum_{\tau' \in \mathcal{T}_{0,s}} \prod_{(s_t \to s_{t+1}) \in \tau'} \hat{P}_F(s_{t+1} \mid s_t).$$

Similarly, any complete trajectory going through $s \to s'$ can be (uniquely) decomposed into a partial trajectory $\tau' \in \mathcal{T}_{0,s}$ from s_0 to s, and a partial trajectory $\tau'' \in \mathcal{T}_{s',f}$ from

s' to s_f. Again, using the definition of $P(s \to s')$:

$$P(s \to s') = \sum_{\tau : (s \to s') \in \tau} P(\tau) = \sum_{\tau : (s \to s') \in \tau} \prod_{(s_t \to s_{t+1}) \in \tau} \hat{P}_F(s_{t+1} \mid s_t)$$

$$= \underbrace{\left[\sum_{\tau' \in \mathcal{T}_{0,s}} \prod_{(s_t \to s_{t+1}) \in \tau'} \hat{P}_F(s_{t+1} \mid s_t) \right]}_{= P(s)} \hat{P}_F(s' \mid s) \underbrace{\left[\sum_{\tau'' \in \mathcal{T}_{s',f}} \prod_{(s_t \to s_{t+1}) \in \tau''} \hat{P}_F(s_{t+1} \mid s_t) \right]}_{= 1 \quad \text{(Lemma 5)}}$$

$$= P(s) \hat{P}_F(s' \mid s).$$

Combining the two results above, we get:

$$P_F(s' \mid s) = \frac{P(s \to s')}{P(s)} = \hat{P}_F(s' \mid s).$$

- $2 \Rightarrow 1$: Suppose that there exists a probability function $\hat{P}_F$ consistent with G such that for some complete trajectory $\tau = (s_0, \ldots, s_{n+1} = s_f)$

$$P(\tau) = \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1}).$$

For the same reasons as those used to justify the uniqueness in the $1 \Rightarrow 2$ proof, $\hat{P}_F$ is necessarily equal to the forward transition probability P_F, associated with P.

We now want to show that the flow F associated with P is Markovian, by showing the Markov property from Eq. (19). Let $\tau' \in \mathcal{T}_{0,s}$ be any partial trajectory from s_0 to s; using the definition of conditional probability:

$$P(s \to s' \mid \tau') = \frac{P(s_0 \to \ldots \to s \to s')}{P(s_0 \to \ldots \to s)}.$$

Following the same idea as above, we will now rewrite $P(s_0 \to \ldots \to s)$, as a sum over complete trajectories that share the same prefix trajectory τ'. Any such complete trajectory τ can be (uniquely) decomposed into this common prefix τ', and a partial trajectory $\tau'' \in \mathcal{T}_{s,f}$ from s to s_f.

$$P(s_0 \to \ldots \to s) = \sum_{\tau : \tau' \subseteq \tau} P(\tau) = \sum_{\tau : \tau' \subseteq \tau} \prod_{(s_t \to s_{t+1}) \in \tau} P_F(s_{t+1} \mid s_t)$$

$$= \left[\prod_{s_{t-1} \to s_t \in \tau'} P_F(s_t \mid s_{t-1}) \right] \underbrace{\left[\sum_{\tau'' \in \mathcal{T}_{s,f}} \prod_{(s_t \to s_{t+1}) \in \tau''} P_F(s_{t+1} \mid s_t) \right]}_{= 1 \quad \text{(Lemma 5)}}$$

$$= \prod_{s_{t-1} \to s_t \in \tau'} P_F(s_t \mid s_{t-1}).$$

14

Similarly, any complete trajectory τ that share the same prefix trajectory $(s_0, \ldots, s, s')$ can be (uniquely) decomposed into this common prefix, and a partial trajectory $\tau'' \in \mathcal{T}_{s',f}$ from s' to s_f, leading to:

$$P(s_0 \to \ldots \to s \to s') = P(s_0 \to \ldots \to s) P_F(s' \mid s)$$

Combining the two results above, we can conclude that P satisfies the Markov property, and therefore that the flow F is Markovian:

$$P(s' \to s \mid \tau') = \frac{P(s_0 \to \ldots \to s \to s')}{P(s_0 \to \ldots \to s)} = P_F(s' \mid s) = P(s' \to s \mid s)$$

- $\{1, 2\} \Rightarrow 3$: Suppose that F is a Markovian flow. We have shown above that this is equivalent to P being decomposed into a product of forward transition probabilities P_F. For some complete trajectory $\tau = (s_0, \ldots, s_{n+1} = s_f)$:

$$P(\tau) = \prod_{t=1}^{n+1} P_F(s_t \mid s_{t-1}) = \prod_{t=1}^{n+1} \frac{P(s_{t-1} \to s_t)}{P(s_{t-1})} = \prod_{t=1}^{n+1} \frac{P(s_{t-1} \to s_t)}{P(s_t)} = \prod_{t=1}^{n+1} P_B(s_{t-1} \mid s_t),$$

where the third equality uses the fact that $P(s_0) = P(s_f) = 1$, and using the definition of the backwards transition probability P_B. The proof of uniqueness of P_B is similar to that of P_F in $1 \Rightarrow 2$, and uses:

$$P(s \to s') = \sum_{\tau : (s \to s') \in \tau} P(\tau) = \sum_{\tau : (s \to s') \in \tau} \prod_{(s_t \to s_{t+1}) \in \tau} \hat{P}_B(s_t \mid s_{t+1})$$

$$= \underbrace{\left[\sum_{\tau' \in \mathcal{T}_{0,s}} \prod_{(s_t \to s_{t+1}) \in \tau'} \hat{P}_B(s_t \mid s_{t+1}) \right]}_{=1 \quad (\text{Lemma 5})} \hat{P}_B(s \mid s') \underbrace{\left[\sum_{\tau'' \in \mathcal{T}_{s',f}} \prod_{(s_t \to s_{t+1}) \in \tau''} \hat{P}_B(s_t \mid s_{t+1}) \right]}_{=P(s')}$$

$$= P(s') \hat{P}_B(s \mid s'),$$

- $3 \Rightarrow 1$: Similar to the proof of $2 \Rightarrow 1$, $\hat{P}_B$ is necessarily equal to the backwards transition probability P_B associated with P. Additionally, P_B is related to the forward transition probability P_F:

$$P(s \to s') = P_B(s \mid s') P(s') = P_F(s' \mid s) P(s).$$

We can therefore write the decomposition of P in terms of P_F, instead of P_B. For some complete trajectory $\tau = (s_0, \ldots, s_{n+1} = s_f)$:

$$P(\tau) = \prod_{t=1}^{n+1} P_B(s_{t-1} \mid s_t) = \prod_{t=1}^{n+1} \frac{P(s_{t-1})}{P(s_t)} P_F(s_{t+1} \mid s_t) = \frac{P(s_0)}{P(s_f)} \prod_{t=1}^{n+1} P_F(s_{t+1} \mid s_t)$$

$$= \prod_{t=1}^{n+1} P_F(s_{t+1} \mid s_t),$$

where we used the fact that $P(s_0) = P(s_f) = 1$. Using "$2 \Rightarrow 1$", we can conclude that F is a Markovian flow.

■

The decomposition of Eq. (20) shows how Markovian flows can be used to draw terminating states from the terminating state probability P_T (Eq. (18)). Namely, we have the following result:

Corollary 17 *Let (G, F) be a Markovian flow network, and P_F the corresponding forward transition probability. Consider the procedure starting from $s = s_0$, and iteratively drawing one sample from $P_F(. \mid s)$ until reaching s_f. Then the probability of the procedure terminating in a state s is $P_T(s)$.*

Proof First, note that the procedure terminates with probability 1, given that G is acyclic.

For the procedure to terminate in a state s, it means that the trajectory $\tau \in \mathcal{T}$ implicitly constructed during the procedure contains the edge $s \to s_f$. The probability of the procedure terminating in s is thus:

$$\sum_{\tau \in \mathcal{T}: s \to s_f \in \tau} \underbrace{\prod_{s' \to s'' \in \tau} P_F(s'' \mid s')}_{P(\tau), \text{ according to Eq. (20)}} = P(s \to s_f) = P_T(s)$$

■

The following proposition shows that, as a consequence of the Prop. 16, we obtain three different parametrizations of Markovian flows.

Proposition 18 *Given a pointed DAG $G = (\mathcal{S}, \mathbb{A})$, a Markovian flow on G is* completely *and* uniquely *specified by one of the following:*

1. *the combination of the total flow $\hat{Z}$ and the forward transition probabilities $\hat{P}_F(s' \mid s)$ for all edges $s \to s' \in \mathbb{A}$,*

2. *the combination of the total flow $\hat{Z}$ and the backward transition probabilities $\hat{P}_B(s \mid s')$ for all edges $s \to s' \in \mathbb{A}$.*

3. *the combination of the terminating flows $\hat{F}(s \to s_f)$ for all terminating edges $s \to s_f \in \mathbb{A}^f$ and the backwards transition probabilities $\hat{P}_B(s \mid s')$ for all non-terminating edges $s \to s' \in \mathbb{A}^{-f}$,*

Proof In the first two settings, we define a flow function $F : \mathcal{T} \to \mathbb{R}^+$, at a trajectory $\tau = (s_0, s_1, \ldots, s_n, s_{n+1} = s_f)$ as:

1. $F(\tau) := \hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1})$,

2. $F(\tau) := \hat{Z} \prod_{t=1}^{n+1} \hat{P}_B(s_{t-1} \mid s_t)$

We need to prove that it is the only Markovian flow that can be defined for both settings. The proof for the third setting will follow from that of the second setting.

    First setting:

First, we need to show that the total flow Z associated with the flow function F (Eq. (10)) matches $\hat{Z}$. This is a consequence of Lemma 5:

$$Z = \sum_{\tau \in \mathcal{T}} F(\tau) = \hat{Z} \underbrace{\sum_{\tau = (s_0, s_1, \ldots, s_{n+1} = s_f) \in \mathcal{T}} \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1})}_{=1 \quad \text{, according to Lemma 5}} = \hat{Z}$$

Then, we need to show that the forward transition probability function P_F associated with F (Eq. (16)) matches $\hat{P}_F$, and that the flow F is Markovian. To this end, note that the corresponding flow probability P satisfies Eq. (20). Thus, as a consequence of Prop. 16, F is a Markovian flow, and its forward transition probability function is $\hat{P}_F$.

As a last requirement, we need to show that if a Markovian flow F' has a partition function $Z' = \hat{Z}$ and a forward transition probability function $P_F' = \hat{P}_F$, then it is necessarily equal to F. This is a direct consequence of Prop. 16, given that for any $\tau = (s_0, \ldots, s_{n+1} = s_f) \in \mathcal{T}$:

$$F'(\tau) = Z' \prod_{t=1}^{n+1} P_F'(s_t \mid s_{t-1}) = \hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1}) = F(\tau)$$

Second setting:

First, we show that as a consequence of Lemma 5, the total flow Z associated with F matches $\hat{Z}$:

$$Z = \sum_{\tau \in \mathcal{T}} F(\tau) = \hat{Z} \underbrace{\sum_{\tau = (s_0, s_1, \ldots, s_{n+1} = s_f) \in \mathcal{T}} \prod_{t=1}^{n+1} \hat{P}_B(s_{-1} \mid s_t)}_{=1 \quad \text{, according to Lemma 5}} = \hat{Z}$$

Second, we note that the flow probability P associated with F satisfies Eq. (21). Thus, as a consequence of Prop. 16, F is a Markovian flow, and its backward transition probability function is $\hat{P}_B$.

Finally, if a Markovian flow F' has a partition function $Z' = \hat{Z}$ and a backward transition probability function $P_B' = \hat{P}_B$, then following Prop. 16, $\forall \tau \in \mathcal{T}, \ F'(\tau) = F(\tau)$.

Third setting:

From the terminating flows $\hat{F}(s \to s_f)$ and the backwards transition probabilities $\hat{P}_B(s \mid s')$ for non-terminating edges, we can uniquely define a total flow $\hat{Z}$, and extend $\hat{P}_B$ to all edges as follows:

$$\hat{Z} := \sum_{s \in Par(s_f)} \hat{F}(s \to s_f)$$

$$\hat{P}_B(s \mid s') := \begin{cases} \hat{P}_B(s \mid s') & \text{if } s' \neq s_f \\ \frac{\hat{F}(s \to s_f)}{\hat{Z}} & \text{otherwise.} \end{cases}$$

This takes us back to the second setting, for which we have already proven that with $\hat{Z}$ and $\hat{P}_B$ defined for all edges, a Markovian flow is uniquely defined.

∎

2.5 Flow Matching Conditions

In Prop. 18, we saw how forward and backward probability functions can be used to uniquely define a Markovian flow. We will show in the next proposition how non-negative functions of states and edges can be used to define a Markovian flow. Such functions cannot be unconstrained (as $\hat{P}_F$ and $\hat{Z}$ in Prop. 18 e.g.), as we have seen in Prop. 8.

Proposition 19 *Let $G = (\mathcal{S}, \mathbb{A})$ be a pointed DAG. Consider a non-negative function $\hat{F}$ taking as input either a state $s \in \mathcal{S}$ or a transition $s{\rightarrow}s' \in \mathbb{A}$. Then $\hat{F}$ corresponds to a flow if and only if the* **flow matching conditions***:*

$$\forall s' > s_0, \ \ \hat{F}(s') = \sum_{s \in Par(s')} \hat{F}(s{\rightarrow}s')$$

$$\forall s' < s_f, \ \ \hat{F}(s') = \sum_{s'' \in Child(s')} \hat{F}(s'{\rightarrow}s'') \tag{22}$$

are satisfied. More specifically, $\hat{F}$ uniquely defines a Markovian flow F matching $\hat{F}$ on states and transitions:

$$\forall \tau = (s_0, \ldots, s_{n+1} = s_f) \in \mathcal{T} \ \ F(\tau) = \frac{\prod_{t=1}^{n+1} \hat{F}(s_{t-1}{\rightarrow}s_t)}{\prod_{t=1}^{n} \hat{F}(s_t)}. \tag{23}$$

Proof Necessity is a direct consequence of Prop. 8. Let's show sufficiency. Let $\hat{P}_F$ be the forward probability function defined by:

$$\forall s \rightarrow s' \in \mathbb{A} \ \ \hat{P}_F(s' \mid s) := \frac{\hat{F}(s \rightarrow s')}{\hat{F}(s)}.$$

$\hat{P}_F$ is consistent with G given that $\hat{F}$ satisfies the flow matching conditions (Eq. (22)). Let $\hat{Z} = \hat{F}(s_0)$. According to Prop. 18, there exists a unique Markovian flow F with forward transition probability function $P_F = \hat{P}_F$ and partition function $Z = \hat{Z}$, and such that for a trajectory $\tau = (s_0, \ldots, s_{n+1} = s_f) \in \mathcal{T}$:

$$\forall \tau = (s_0, \ldots, s_{n+1} = s_f) \in \mathcal{T} \ \ F(\tau) = \hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1}) = \frac{\prod_{t=1}^{n+1} \hat{F}(s_{t-1}{\rightarrow}s_t)}{\prod_{t=1}^{n} \hat{F}(s_t)}. \tag{24}$$

Additionally, similar to the proof of Prop. 16, we can write for any state $s' \neq s_0$:

$$\begin{aligned}
F(s') &= \hat{Z} \sum_{\tau \in \mathcal{T}_{0,s'}} \prod_{(s_t \rightarrow s_{t+1}) \in \tau} \hat{P}_F(s_{t+1} \mid s_t) \\
&= \hat{Z} \frac{\hat{F}(s')}{\hat{F}(s_0)} \underbrace{\sum_{\tau \in \mathcal{T}_{0,s'}} \prod_{(s_t \rightarrow s_{t+1}) \in \tau} \hat{P}_B(s_t \mid s_{t+1})}_{=1 \text{ , according to Lemma 5}} \\
&= \hat{F}(s'),
\end{aligned}$$

18

where $\hat{P}_B(s' \mid s) := \frac{\hat{F}(s \to s')}{\hat{F}(s')}$ defines a backward probability function consistent with G. And because $\forall s \to s' \in \mathbb{A}$ $P_F(s' \mid s) = \hat{P}_F(s' \mid s)$, it follows that $\forall s \to s' \in \mathbb{A}$ $F(s \to s') = \hat{F}(s \to s')$.

To show uniqueness, let's consider a Markovian flow F' that matches $\hat{F}$ on states and edges. Following Prop. 16, for any trajectory $\tau = (s_0, \dots, s_{n+1} = s_f) \in \mathcal{T}$

$$F'(\tau) = \hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1}) = \frac{\prod_{t=1}^{n+1} \hat{F}(s_{t-1} \to s_t)}{\prod_{t=1}^{n} \hat{F}(s_t)} = F(\tau).$$

$\blacksquare$

Note how Eq. (22) can be used to recursively define the flow in all the states if Z is given and either the forward or the backwards transition probabilities are given. Either way, we would start from the flow at one of the extreme states s_0 or s_f and then distribute it recursively through the directed acyclic graph of the flow network, either going forward or going backward. A setting of particular interest, that will be central in Sec. 3, is when we are given all the terminal flows $F(s \to s_f)$, and we would like to deduce a state flow function $F(s)$ and a forward transition probability function $P_F(s' \mid s)$ for the rest of the flow network.

Next, we will see how to parametrize Markovian flows using forward and backward probability functions consistent with the DAG. Unlike the condition in Prop. 19, the new condition does not involve a sum over transitions, which could be problematic if each state can have a large number of successors or if the state-space is continuous. Interestingly, the resulting condition is analogous to the *detailed balance* condition of Monte-Carlo Markov chains.

Definition 20 *Given a pointed DAG $G = (\mathcal{S}, \mathbb{A})$, a forward transition probability function $\hat{P}_F$ and a backward transition probability function $\hat{P}_B$ consistent with G, $\hat{P}_F$ and $\hat{P}_B$ are **compatible** if there exists an edge flow function $\hat{F} : \mathbb{A} \to \mathbb{R}^+$ such that*

$$\forall s \to s' \in \mathbb{A} \ \hat{P}_F(s' \mid s) = \frac{\hat{F}(s \to s')}{\sum_{s' \in Child(s)} \hat{F}(s \to s')}, \quad \hat{P}_B(s \mid s') = \frac{\hat{F}(s \to s')}{\sum_{s'' \in Par(s')} \hat{F}(s'' \to s')} \tag{25}$$

Proposition 21 *Let $G = (\mathcal{S}, \mathbb{A})$ be a pointed DAG. Consider a non-negative function $\hat{F}$ over states, a forward transition probability function $\hat{P}_F$ and a backwards transition probability function $\hat{P}_B$ consistent with G. Then, $\hat{F}, \hat{P}_B$, and $\hat{P}_F$ jointly correspond to a flow if and only if the **detailed balance conditions** holds:*

$$\forall s \to s' \in \mathbb{A} \ \ \hat{F}(s)\hat{P}_F(s' \mid s) = \hat{F}(s')\hat{P}_B(s \mid s'). \tag{26}$$

More specifically, $\hat{F}, \hat{P}_F$, and $\hat{P}_B$ uniquely define a Markovian flow F matching $\hat{F}$ on states, and with transition probabilities matching $\hat{P}_F$ and $\hat{P}_B$. Furthermore, when this condition is satisfied, the forward and backward transition probability functions $\hat{P}_F$ and $\hat{P}_B$ are compatible.

Proof For necessity, consider a flow F, with state flow function denoted F, and forward and backward transitions P_F and P_B. It is clear from the definition of P_F and P_B (Def. 12) that Eq. (26) holds. We prove the sufficiency of the condition by first defining the edge flow

$$\forall s \to s' \in \mathbb{A} \quad \hat{F}(s \to s') := \hat{F}(s)\hat{P}_F(s' \mid s). \tag{27}$$

We then sum both sides of Eq. (26) over s, yielding

$$\forall s' > s0 \quad \sum_{s \in Par(s')} \hat{F}(s)\hat{P}_F(s' \mid s) = \hat{F}(s') \sum_{s \in Par(s')} \hat{P}_B(s \mid s') = \hat{F}(s') \tag{28}$$

where we used the fact that $\hat{P}_B$ is a normalized probability distribution. Combining this with Eq. (27), we get

$$\forall s' > s_0 \quad \hat{F}(s') = \sum_{s \in Par(s')} \hat{F}(s \to s') \tag{29}$$

which is the first equality of the flow-matching condition (Eq. (22)) of Prop. 19. We can obtain the second equality by first using the normalization of $\hat{P}$, and then using our definition of the edge flow (Eq. (27)):

$$\forall s' > s_0 \quad \hat{F}(s') = \hat{F}(s') \sum_{s'' \in Child(s')} \hat{P}_F(s'' \mid s')$$

$$= \sum_{s'' \in Child(s')} \hat{F}(s')\hat{P}_F(s'' \mid s')$$

$$= \sum_{s'' \in Child(s')} \hat{F}(s' \to s''). \tag{30}$$

Following Prop. 19, there exists a unique Markovian flow F with state and edge flows given by $\hat{F}$. Using Eq. (27) and Eq. (26), it follows that F has transition probabilities $\hat{P}_F$ and $\hat{P}_B$ as required. The uniqueness is also a consequence of Eq. (27). This proves sufficiency.

To show that $\hat{P}_F$ and $\hat{P}_B$ are compatible (Def. 20), we first combine Eq. (27) and Eq. (30) (with relabeling of variables) to obtain

$$\forall s \to s' \in \mathbb{A} \quad \hat{P}_F(s' \mid s) = \frac{\hat{F}(s \to s')}{\sum_{s' \in Child(s)} \hat{F}(s \to s')},$$

we then isolate $\hat{P}_B$ in Eq. (26), yielding

$$\forall s \to s' \in \mathbb{A} \quad \hat{P}_B(s \mid s') = \frac{\hat{F}(s)}{\hat{F}(s')}\hat{P}_F(s' \mid s) = \frac{\hat{F}(s \to s')}{\hat{F}(s')} = \frac{\hat{F}(s \to s')}{\sum_{s'' \in Par(s')} \hat{F}(s'' \to s')},$$

We thus get Eq. (25) of Def. 20, as desired. ∎

At first glance, it may seem that when $\hat{P}_B$ is unconstrained, the detailed balance condition can trivially be achieved by setting

$$\forall s \to s' \in \mathbb{A} \quad \hat{P}_B(s \mid s') = \frac{\hat{P}_F(s' \mid s)\hat{F}(s)}{\hat{F}(s')} \tag{31}$$

However, because we also have the constraint $\sum_{s \in Par(s')} \hat{P}_B(s \mid s') = 1$, then Eq. (31) can only be satisfied if the flows are consistent with the forward transition:

$$\sum_{s \in Par(s')} \hat{P}_F(s' \mid s)\hat{F}(s) = \hat{F}(s').$$

2.6 Backwards Transitions can be Chosen Freely

Consider the setting in which we are given terminating flows to be matched, i.e. where the goal is to find a flow function with the right terminating flows. This is the setting introduced in Bengio et al. (2021), and that will be studied in Sec. 3. In this case, Prop. 18 tells us that in order to fully determine the forward transition probabilities and the state or state-action flows, it is not sufficient in general to specify only the terminating flows; it is also necessary to specify the backwards transition probabilities on the edges other than the terminal ones (the latter being given by the terminating flows).

What this means is that the terminating flows do not specify the flow completely, e.g., because many different paths can land in the same terminating state. The preference over such different ways to achieve the same final outcome is specified by the backwards transition probability P_B (except for $P_B(s \mid s_f)$ which is a function of the terminating flows and Z). For example, we may want to give equal weight to all parents of a node s, or we may prefer shorter paths, which can be achieved if we keep track in the state s of the length of the shortest path to the node s, or we may let a learner discover a P_B that makes learning P_F or F easier.

2.7 Equivalence Between Flows

In the previous sections, we have seen that Markovian flows have the property that trajectory flows or probabilities factorize according to the DAG, and we have seen different ways of characterizing Markovian flows. In Sec. 3, we show how to approximate Markovian flows in order to define probability measures over terminating states. In this section, through an equivalence relation between trajectory flows, we justify the focus on Markovian flows. Given a pointed DAG $G = (\mathcal{S}, \mathbb{A})$, we denote by:

- $\mathcal{F}(G)$: the set of flows on G, i.e. the set of functions from $\mathcal{T}$, the set of complete trajectories in G, to $\mathbb{R}^+$,

- $\mathcal{F}_{Markov}(G)$: the set of flows in $\mathcal{F}(G)$ that are Markovian.

Definition 22 *Let $G = (\mathcal{S}, \mathbb{A})$ be a pointed DAG, and $F_1, F_2 \in \mathcal{F}(G)$ two trajectory flow functions. We say that F_1 and F_2 are equivalent if they coincide on edge-flows, i.e.:*

$$\forall s \to s' \in \mathbb{A} \quad F_1(s \to s') = F_2(s \to s')$$

Fig. 4 shows four flow functions in a simple pointed DAG that are pairwise equivalent.

This defines an equivalence relation (i.e., a relation that is reflexive, symmetric, and transitive). Hence, each flow F belongs to an equivalence class, and the set of flows $\mathcal{F}(G)$ can be partitioned into equivalence classes. Note that if two flows are equivalent, then the corresponding state flow functions also coincide (as a direct consequence of Prop. 8).
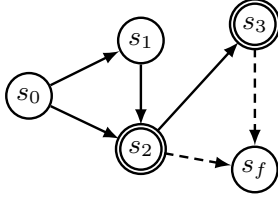
τ	$F_1(\tau)$	$F_2(\tau)$	$F_3(\tau)$	$F_4(\tau)$
s_0, s_2, s_f	1	4/5	1	6/5
s_0, s_1, s_2, s_f	1	6/5	1	4/5
s_0, s_2, s_3, s_f	1	6/5	2	9/5
s_0, s_1, s_2, s_3, s_f	2	9/5	1	6/5

Figure 4: Equivalent flows and Markovian flows. Flows F_1 and F_2 are equivalent. F_3 and F_4 are equivalent, but not equivalent to F_1 and F_2. F_2 and F_4 are Markovian. F_1 and F_3 are not Markovian. F_1, F_2, F_3 and F_4 coincide on the terminating flows i.e. at $s_2 \to s_f$ and $s_3 \to s_f$.

Proposition 23 *Given a pointed DAG G. If two flow function $F_1, F_2 \in \mathcal{F}_{Markov}(G)$ are equivalent, then they are equal. Additionally, for any flow function $F' \in \mathcal{F}(G)$, there exists a unique Markovian flow function $F \in \mathcal{F}_{Markov}(G)$ such that F and F' are equivalent.*

Proof Because F_1 and F_2 are Markovian, then for any trajectory $\tau = (s_0, \ldots, s_{n+1} = s_f)$:

$$F_1(\tau) = \frac{\prod_{t=1}^{n+1} F_1(s_{t-1} \to s_t)}{\prod_{t=1}^{n} F_1(s_t)}$$
$$= \frac{\prod_{t=1}^{n+1} F_2(s_{t-1} \to s_t)}{\prod_{t=1}^{n} F_2(s_t)}$$
$$= F_2(\tau),$$

where we combined the definition of equivalent flows and Prop. 16.

Given a flow function F', because its state and edge flow functions satisfy the flow matching conditions (as a consequence of Prop. 8), then according to Prop. 19, the flow F defined by:

$$\forall \tau := (s_0, \ldots, s_{n+1} = s_f) \in \mathcal{T} \quad F(\tau) = \frac{\prod_{t=1}^{n+1} F'(s_{t-1} \to s_t)}{\prod_{t=1}^{n} F'(s_t)}$$

is Markovian, and coincides with F' on state and edge flows. Combining this with the statement above, we conclude that F is the unique Markovian flow that is equivalent to F'. ∎

The previous proposition shows that in each equivalence class stands out a particular flow function, that has a property the other flows in the same equivalence class don't have: it is Markovian.

A consequence of this is that, if we care essentially about state and edge flows, instead of dealing with the full set of flows $\mathcal{F}(G)$, it suffices to restrict any flow learning problem to the set of Markovian flows $\mathcal{F}_{Markov}(G)$. The advantage of this restriction is that defining a flow requires the specification of $F(\tau)$ for all trajectories $\tau \in \mathcal{T}$, whereas defining a Markovian flow requires the specification of $F(s \to s')$ for all edges $s \to s' \in \mathbb{A}$, which is generally exponentially smaller than $\mathcal{T}$ (note that the edge flows still need to satisfy the flow-matching conditions in Prop. 19). Thus, in order to approximate or learn a flow

function that satisfies some conditions on its edge or state values, it suffices to approximate or learn a Markovian flow, by learning the edge flow function, which is a much smaller object than the actual flow function.

3. GFlowNets: Learning a Flow

With the theoretical preliminaries established in Sec. 1 and Sec. 2, we now consider the general class of problems introduced by Bengio et al. (2021) where some constraints or preferences over flows are given. Our goal is to find functions such as the state flow function $F(s)$ or the transition probability function $P(s \to s' \mid s)$ that best match these desiderata using corresponding estimators $\hat{F}(s)$ and $\hat{P}(s \to s' \mid s)$ which may not correspond to a proper flow. Such learning machines are called Generative Flow Networks (or GFlowNets for short). We focus on scenarios where we are given a target *reward function* $R : \mathcal{S}^f \to \mathbb{R}^+$, and aim at estimating flows F that satisfy:

$$\forall s \in \mathcal{S}^f \quad F(s \to s_f) = R(s) \tag{32}$$

Because of the equivalences that exist in the set of flows, then without loss of generality, we choose GFlowNets to approximate Markovian flows only. We are thus interested in the following set of flows:

$$\mathcal{F}_{Markov}(G, R) = \{F \in \mathcal{F}_{Markov}(G), \ \forall s \in \mathcal{S}^f \ \ F(s \to s_f) = R(s)\} \tag{33}$$

For now, we informally define a **GFlowNet** as an **estimator of a Markovian flow** function $F \in \mathcal{F}_{Markov}(G, R)$. We provide a more formal definition later-on.

With an estimator $\hat{F}$ of such a Markovian flow F, we can define an approximate forward transition probability function $\hat{P}_F$, as in Prop. 16, in order to draw trajectories $\tau \in \mathcal{T}$ (the set of complete trajectories in G) by iteratively sampling each state given the previous one, starting at s_0 and then with $s_{t+1} \sim \hat{P}_F(. \mid s_t)$ until we reach the sink state $s_{n+1} = s_f$ for some n.

Next, we will clarify how such an estimator can be obtained.

3.1 GFlowNets as an Alternative to MCMC Sampling

The main established methods to approximately sample from the distribution associated with an energy function $\mathcal{E}$ are Monte-Carlo Markov chain (MCMC) methods, which require significant computation (running a potentially very long Markov chain) to obtain samples. Instead, the GFlowNet approach amortizes upfront computation to train a generator that yields very efficient computation (a single configuration is constructed, no chain needed) for each new sample. For example, Bengio et al. (2021) build a GFlowNet that constructs a molecule via a small sequence of actions, each of which adds an atom or a molecular substructure to an existing molecule represented by a graph, starting from an empty graph. Only one such configuration needs to be considered, in contrast with MCMC methods, which require potentially very long chains of such configurations, and suffer from the challenge of mode-mixing (Jasra et al., 2005; Bengio et al., 2013; Pompe et al., 2020), which can take time exponentially long in the distance between modes. In GFlowNets, this computational challenge is avoided but the computational demand is converted to that of training the

GFlowNet. To see how this can be extremely beneficial, consider having already constructed some configurations x and obtained their unnormalized probability or reward $R(x)$. With these pairs $(x, R(x))$, a machine learning system could potentially generalize about the value of R elsewhere, and if it is a generative model, sample new x's in places of large $R(x)$. Hence, if there is an underlying statistical structure in how the modes of R are related to each other, a generative learner that generalizes could guess the presence of modes it has not visited yet, taking advantage of the patterns it has already uncovered from the $(x, R(x))$ pairs it has seen. On the other hand, if there is no structure (the modes are randomly placed), then we should not expect GFlowNets to do significantly better than MCMC because training becomes intractable in high-dimensional spaces (since it requires visiting every area of the configuration space to ascertain its reward).

3.2 GFlowNets and flow-matching losses

We have seen in Sec. 2.4 and Sec. 2.5 different ways of parametrizing a flow. For example, with a partition function and forward transition probabilities, or with edge flows that satisfy the flow matching conditions. Because there are many ways to parametrize GFlowNets, we start with an abstract formulation for them, where $o \in \mathcal{O}$ represents a parameter configuration (e.g., resulting from or while training of a GFlowNet), $\Pi(o)$ gives the corresponding probability measure over trajectories $\tau \in \mathcal{T}$, and $\mathcal{H}$ maps a Markovian flow F to its parametrization o. In the following definition, we show what conditions should be satisfied in order for such a parametrization to be valid.

Definition 24 *Given a pointed DAG $G = (\mathcal{S}, \mathbb{A})$, with an initial and sink states s_0 and s_f respectively, and a target reward function $R : \mathcal{S}^f \to \mathbb{R}^+$, we say that the triplet $(\mathcal{O}, \Pi, \mathcal{H})$ is a **flow parametrization** of (G, R) if:*

1. *$\mathcal{O}$ is a non-empty set,*

2. *Π is a function mapping each object $o \in \mathcal{O}$ to an element $\Pi(o) \in \Delta(\mathcal{T})$, the set of probability distributions on $\mathcal{T}$,*

3. *$\mathcal{H}$ is an injective functional from $\mathcal{F}_{Markov}(G, R)$ to $\mathcal{O}$,*

4. *For any $F \in \mathcal{F}_{Markov}(G, R)$, $\Pi(\mathcal{H}(F))$ is the probability measure associated with the flow F (Def. 11).*

*To each object $o \in \mathcal{O}$, the distribution $\Pi(o)$ implicitly defines a **terminating state probability** measure:*

$$\forall s \in \mathcal{S}^f \quad P_T(s) := \sum_{\tau \in \mathcal{T}: s \to s_f \in \tau} \Pi(o)(\tau), \tag{34}$$

where the dependence on o in P_T is omitted for clarity.

The intuition behind the introduction of $(\mathcal{O}, \Pi, \mathcal{H})$ is that we can define a probability measure over $\mathcal{T}$ for each object $o \in \mathcal{O}$, but only some of these objects correspond to a Markovian flow with the right terminating flows. For such objects o (i.e. those that can

be written as $o = \mathcal{H}(F)$ for some flow $F \in \mathcal{F}_{Markov}(G, R)$), the probability measure P_T corresponds to the distribution of interest, according to Def. 13, i.e.:

$$\forall s \in \mathcal{S}^f \quad P_T(s) \propto R(s)$$

GFlowNets thus provide a solution to the generally intractable problem of sampling from a target reward function R, or its associated **energy function**:

$$\forall s \in \mathcal{S}^f \quad \mathcal{E}(s) := -\log R(s) \tag{35}$$

Directly approximating flows $F \in \mathcal{F}_{Markov}(G, R)$ is a hard problem, whereas with some sets $\mathcal{O}$, searching for an object $o \in \mathcal{H}(\mathcal{F}_{Markov}(G, R)) \subseteq \mathcal{O}$ is a simpler problem that can be tackled with function approximation techniques.

Note that not the set $\mathcal{O}$ cannot be arbitrary, as there needs to be a way to define an injective function from $\mathcal{F}_{Markov}(G, R)$ to $\mathcal{O}$. Below, for a given DAG G, we show three examples clarifying the abstract concept of parametrization:

Example 1 Edge-flow parametrization: *Consider $\mathcal{O}_{edge} = \mathcal{F}(\mathbb{A}^{-f}, \mathbb{R}^+)$, the set of functions from $\mathbb{A}^{-f}$ to $\mathbb{R}^+$, and the functionals $\mathcal{H}_{edge} : \mathcal{F}_{Markov}(G, R) \to \mathcal{O}_{edge}$ and $\Pi_{edge} : \mathcal{O}_{edge} \to \Delta(\mathcal{T})$ defined by:*

$$\mathcal{H}_{edge}(F) : (s \to s') \in \mathbb{A}^{-f} \mapsto F(s \to s'),$$

$$\forall \tau = (s_0, \ldots, s_n = s_f) \in \mathcal{T} \quad \Pi_{edge}(\hat{F})(\tau) \propto \prod_{t=1}^{n} P_{\hat{F}}(s_t \mid s_{t-1}),$$

where

$$P_{\hat{F}}(s' \mid s) = \begin{cases} \dfrac{\hat{F}(s \to s')}{\sum_{s'' \neq s_f} \hat{F}(s \to s'') + R(s)} & \text{if } s' \neq s_f \\[2ex] \dfrac{R(s)}{\sum_{s'' \neq s_f} \hat{F}(s \to s'') + R(s)} & \text{if } s' = s_f \end{cases} \tag{36}$$

The injectivity of $\mathcal{H}_{edge}$ follows directly from Prop. 23 (two Markovian flows that coincide on both their terminating and non-terminating edge flow values are equal). And for any Markovian flow $F \in \mathcal{F}_{Markov}(G, R)$, $\Pi_{edge}(\mathcal{H}_{edge}(F))$ equals the probability measure associated with F, as is shown in Prop. 16.

$(\mathcal{O}_{edge}, \Pi_{edge}, \mathcal{H}_{edge})$ is thus a valid flow parametrization of (G, R).

Example 2 Forward transition probability parametrization: *Consider the set $\mathcal{O}_{PF} = \mathcal{O}_1 \times \mathcal{O}_2$, where $\mathcal{O}_1 = \mathcal{F}(\mathcal{S} \setminus \{s_f\}, \mathbb{R}^+)$ is the set of function from $\mathcal{S} \setminus \{s_f\}$ to $\mathbb{R}^+$ and $\mathcal{O}_2$ is the set of forward probability functions $\hat{P}_F$ consistent with G , and the functionals $\mathcal{H}_{PF} : \mathcal{F}_{Markov}(G, R) \to \mathcal{O}_{PF}$ and $\Pi_{PF} : \mathcal{O}_{PF} \to \Delta(\mathcal{T})$ defined by:*

$$\mathcal{H}_{PF}(F) = \left(s \in \mathcal{S} \setminus \{s_f\} \mapsto F(s), (s \to s') \in \mathbb{A} \mapsto P_F(s' \mid s) \right),$$

$$\forall \tau = (s_0, \ldots, s_n = s_f) \in \mathcal{T} \quad \Pi_{PF}(\hat{F}, \hat{P}_F)(\tau) \propto \prod_{t=1}^{n} \hat{P}_F(s_t \mid s_{t-1}),$$

where P_F is the forward transition probability function associated with F (Eq. (16)). To verify that $\mathcal{H}_{PF}$ is injective, consider $F_1, F_2 \in \mathcal{F}_{Markov}(G, R)$ such that $\mathcal{H}_{PF}(F_1) = \mathcal{H}_{PF}(F_2)$. It means that $\forall s \in \mathcal{S}^f$, $F_1(s) = F_2(s)$, and $\forall s \to s' \in \mathbb{A}$, $\frac{F_1(s \to s')}{F_1(s)} = \frac{F_2(s \to s')}{F_2(s)}$. It follows that $\forall s \to s' \in \mathbb{A}$, $F_1(s \to s') = F_2(s \to s')$. Which, according to Prop. 23, means that $F_1 = F_2$. And for any Markovian flow $F \in \mathcal{F}_{Markov}(G, R)$, $\Pi_{PF}(\mathcal{H}_{PF}(F))$ equals the probability measure associated with F, as is shown in Prop. 16.

$(\mathcal{O}_{PF}, \Pi_{PF}, \mathcal{H}_{PF})$ is thus a valid flow parametrization of (G, R).

Example 3 Transition probabilities parametrization: *Similar to Ex. 2, we can parametrize a Markovian flow using the state-flow function and both its forward and backward transition probabilities, i.e. with $\mathcal{O}_{PFB} = \mathcal{O}_{PF} \times \mathcal{O}_3$, $\mathcal{H}_{PFB}$, and Π_{PFB} defined as:*

$$\mathcal{H}_{PFB}(F) = \left(\mathcal{H}_{PF}(F), (s \to s') \in \mathbb{A}^{-f} \mapsto P_B(s \mid s'), \right),$$

$$\forall \tau = (s_0, \ldots, s_n = s_f) \in \mathcal{T} \quad \Pi_{PFB}(\hat{F}, \hat{P}_F, \hat{P}_B)(\tau) \propto \prod_{t=1}^{n} \hat{P}_F(s_t \mid s_{t-1}),$$

where P_B is the function defined by Eq. (17). and $\mathcal{O}_3$ is the set of backward probability functions $\hat{P}_B$ consistent with G. The injectivity of $\mathcal{H}_{PFB}$ is a direct consequence of that of $\mathcal{H}_{PF}$. And for any Markovian flow F, $\Pi_{PFB}(\mathcal{H}_{PFB}(F))$ equals the probability measure associated with F, as is shown in Prop.3.

$(\mathcal{O}_{PFB}, \Pi_{PFB}, \mathcal{H}_{PFB})$ is thus a valid flow parametrization of (G, R).

We now have all the ingredients to formally define a GFlowNet:

Definition 25 A **GFlowNet** is a tuple $(G, R, \mathcal{O}, \Pi, \mathcal{H})$, where:

- $G = (\mathcal{S}, \mathbb{A})$ is a pointed DAG with initial state s_0 and sink state s_f,

- $R : \mathcal{S}^f \to \mathbb{R}^+$ a target reward function,

- $(\mathcal{O}, \Pi, \mathcal{H})$ a flow parametrization of (G, R).

Each object $o \in \mathcal{O}$ is called a GFlowNet configuration. When it is clear from context, we will use the term GFlowNet to refer to both $(G, R, \mathcal{O}, \Pi, \mathcal{H})$ and a particular configuration o; similar to how the term "Neural Network" refers to both the class of functions that can be represented with a particular architecture, and to a particular element of that class / weight configuration.

If $o \in \mathcal{H}(\mathcal{F}_{Markov}(G, R))$, then the corresponding terminating state probability measure (Eq. (34)) is proportional to the target reward R.

Once we have a GFlowNet $(G, R, \mathcal{O}, \Pi, \mathcal{H})$, we still need a way to find objects $o \in \mathcal{H}(\mathcal{F}_{Markov}(G, R)) \subseteq \mathcal{O}$. To this end, it suffices to design a **loss function** $\mathcal{L}$ on $\mathcal{O}$ that equals zero on objects $o \in \mathcal{H}(\mathcal{F}_{Markov}(G, R))$ and only on those objects. If our loss function $\mathcal{L}$ is chosen to be non-negative, then an approximation of the target distribution (on $\mathcal{S}^f$) is obtained by approximating the minimum of the function $\mathcal{L}$. This provides a recipe for casting the search problem of interest to a minimization problem, as we typically do in machine learning. Such loss functions can be easily designed for the natural parametrizations we considered in Ex. 1, Ex. 2, and Ex. 3, as we will illustrate below.

Definition 26 *Let* $(G, R, \mathcal{O}, \Pi, \mathcal{H})$ *be a GFlowNet. A **flow-matching loss** is any function* $\mathcal{L} : \mathcal{O} \to \mathbb{R}^+$ *such that:*

$$\forall o \in \mathcal{O} \quad \mathcal{L}(o) = 0 \iff \exists F \in \mathcal{F}_{Markov}(G, R) \quad o = \mathcal{H}(F) \tag{37}$$

We say that $\mathcal{L}$ *is **edge-decomposable**, if there exists a function* $L : \mathcal{O} \times \mathbb{A} \to \mathbb{R}^+$ *such that:*

$$\forall o \in \mathcal{O} \quad \mathcal{L}(o) = \sum_{s \to s' \in \mathbb{A}} L(o, s \to s'),$$

We say that $\mathcal{L}$ *is **state-decomposable**, f there exists a function* $L : \mathcal{O} \times \mathcal{S} \to \mathbb{R}^+$ *such that:*

$$\forall o \in \mathcal{O} \quad \mathcal{L}(o) = \sum_{s \in \mathcal{S}} L(o, s),$$

We say that $\mathcal{L}$ *is **trajectory-decomposable** if there exists a function* $L : \mathcal{O} \times \mathcal{T} \to \mathbb{R}^+$ *such that:*

$$\forall o \in \mathcal{O} \quad \mathcal{L}(o) = \sum_{\tau \in \mathcal{T}} L(o, \tau)$$

As mentioned above, with a such a loss function, our search problems can be written as minimization problems of the form

$$\min_{o \in \mathcal{O}} \mathcal{L}(o), \tag{38}$$

which can be tackled with gradient-based learning if the function $\mathcal{L}$ is differentiable. Note that with an edge-decomposable flow-matching loss, the minimization problem in Eq. (38) is equivalent to:

$$\min_{o \in \mathcal{O}} \mathbb{E}_{(s \to s') \sim \pi_T}[L(o, s \to s')], \tag{39}$$

where π_T is any full support probability distribution on $\mathbb{A}$, i.e. a probability distribution such that $\forall s \to s' \in \mathbb{A} \quad \pi_T(s \to s') > 0$. A similar statement can be made for state-decomposable or trajectory-decomposable flow-matching losses.

Example 4 *Consider the edge-flow parametrization* $(\mathcal{O}_{edge}, \Pi_{edge}, \mathcal{H}_{edge})$*, and the function* $L_{FM} : \mathcal{O}_{edge} \times \mathcal{S} \to \mathbb{R}^+$ *defined for each* $\hat{F} \in \mathcal{O}_{edge}$ *and* $s' \in \mathcal{S}$ *as*

$$L_{FM}(\hat{F}, s') = \begin{cases} \left(\log \left(\frac{\delta + \sum_{s \in Par(s')} \hat{F}(s \to s')}{\delta + R(s') + \sum_{s'' \in Child(s') \setminus \{s_f\}} \hat{F}(s' \to s'')} \right) \right)^2 & \text{if } s' \neq s_f, \\ 0 & \text{otherwise} \end{cases}$$

where $\delta \geq 0$ *is a hyper-parameter. The function* $\mathcal{L}_{FM}$ *mapping each* $\hat{F} \in \mathcal{O}_{edge}$ *to*

$$\mathcal{L}_{FM}(\hat{F}) = \sum_{s \in \mathcal{S}} L_{FM}(\hat{F}, s) \tag{40}$$

is a flow-matching loss, that is (by definition) state-decomposable.

To see this, let $\hat{F} \in \mathcal{O}_{edge}$ such that $\mathcal{L}_{FM}(\hat{F}) = 0$, and extend it to terminating edge:

$$\forall s \in \mathcal{S}^f \quad \hat{F}(s \to s_f) := R(s)$$

Now that $\hat{F}$ is defined for all edges in G, we can write that

$$\forall s' \in \mathcal{S} \quad \sum_{s \in Par(s')} \hat{F}(s \to s') = \sum_{s'' \in Child(s)} \hat{F}(s' \to s'').$$

Which, according to Prop. 19, means that there exists a Markovian flow $F \in \mathcal{F}_{Markov}(G, R)$ such that $\mathcal{H}_{edge}(F) = \hat{F}$. The converse

$$\forall F \in \mathcal{F}_{Markov}(G, R) \quad \mathcal{L}_{FM}(\mathcal{H}_{edge}(F)) = 0$$

is a trivial consequence of Prop. 19.

This is the loss function proposed in Bengio et al. (2021). δ allows to reduce the importance given to small flows (those smaller than δ), and the usage of the square of the log-ratio is justified as a way to ensure that states with large flows do not contribute to the gradients of $\mathcal{L}_{FM}$ much more than states with small flows.

Example 5 Detailed-balance loss: *Consider the transition probabilities parametrization $(\mathcal{O}_{PFB}, \Pi_{PFB}, \mathcal{H}_{PFB})$, and the function $L_{DB} : \mathcal{O}_{PFB} \times \mathbb{A} \to \mathbb{R}^+$ defined for each $(\hat{F}, \hat{P}_F, \hat{P}_B) \in \mathcal{O}_{PFB}$ and $s \to s' \in \mathbb{A}$ as*

$$L_{DB}(\hat{F}, \hat{P}_F, \hat{P}_B, s \to s') = \begin{cases} \left(\log\left(\frac{\delta + \hat{F}(s)\hat{P}_F(s'|s)}{\delta + \hat{F}(s')\hat{P}_B(s|s')}\right)\right)^2 & \text{if } s' \neq s_f, \\ \left(\log\left(\frac{\delta + \hat{F}(s)\hat{P}_F(s'|s)}{\delta + R(s)}\right)\right)^2 & \text{otherwise,} \end{cases}$$

where $\delta \geq 0$ is a hyper-parameter. The function $\mathcal{L}_{DB}$ mapping each $(\hat{F}, \hat{P}_F, \hat{P}_B) \in \mathcal{O}_{PFB}$ to

$$\mathcal{L}_{DB}(\hat{F}, \hat{P}, \hat{P}_B)) = \sum_{s \to s' \in \mathbb{A}} L_{DB}(\hat{F}, \hat{P}, \hat{P}_B, s \to s')$$

is a flow-matching loss that is (by definition) edge-decomposable. The proof of this statement is similar to the one of the example above, using Prop. 21.

According to Sec. 2.6, the reward function does not completely specify the flow. Thus, the detailed-balance loss of Ex. 5 can be used with the $(\mathcal{O}_{PF}, \Pi_{PF}, \mathcal{H}_{PF})$ parametrization, using any function $\hat{P}_B \in \mathcal{O}_3$ as input to the detailed-balance loss.

Example 6 Trajectory-balance loss: *This loss has been introduced in Malkin et al. (2022) for the parametrization $(\mathcal{O}_{TB}, \Pi_{TB}, \mathcal{H}_{TB})$, where $\mathcal{O}_{TB} = \mathcal{O}_1 \times \mathcal{O}_2 \times \mathcal{O}_3$, with $\mathcal{O}_1 = \mathbb{R}^+$ parametrizes the partition function $\hat{Z}$, and $\mathcal{O}_2$ and $\mathcal{O}_3$ introduced in Ex. 2 and Ex. 3 (the set of forward and backward probabilities consistent with G). $\mathcal{H}_{TB}$ maps a Markovian flow in $\mathcal{F}_{Markov}(G, R)$ to the corresponding triplet (Z, P_F, P_B), and Π_{TB} maps a parametrization $(\hat{Z}, \hat{P}_F, \hat{P}_B)$ to a probability over trajectories defined by $\hat{P}_F$ as in Ex. 2. Prop. 18 justifies the validity of this parametrization. The loss $\mathcal{L}_{TB}$ maps each $(\hat{Z}, \hat{P}_F, \hat{P}_B) \in \mathcal{O}_{TB}$ to:*

$$\mathcal{L}_{TB}(\hat{Z}, \hat{P}_F, \hat{P}_B) = \sum_{\tau \in \mathcal{T}} L_{TB}(\hat{Z}, \hat{P}_F, \hat{P}_B, \tau),$$

where

$$\forall \tau = (s_0, \ldots, s_{n+1} = s_f) \in \mathcal{T} \quad L_{TB}(\hat{Z}, \hat{P}_F, \hat{P}_B, \tau) = \left(\log \frac{\hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t \mid s_{t-1})}{R(s_n) \prod_{t=1}^{n} \hat{P}_B(s_{t-1} \mid s_t)} \right)^2.$$

$$(41)$$

Malkin et al. (2022) prove that $\mathcal{L}_{TB}$ is a flow-matching loss and call it trajectory balance. *It is trajectory-decomposable by definition.*

Training by stochastic gradient descent: In the examples of the previous section, given a GFlowNet $(G, R, \mathcal{O}, \Pi, \mathcal{H})$ and a flow-matching loss $\mathcal{L}$, objects $o \in \mathcal{O}$ are themselves functions or combinations of functions, and we can thus parametrize $\mathcal{O}$ with function approximators such as Neural Networks. However, most of the times, the evaluation (let alone the minimization) of $\mathcal{L}(o)$ is intractable, given that even with a full support distribution, only a subset of edges (or states or trajectories) can be visited in finite time. In practice, with an edge-decomposable loss e.g., we resort to a stochastic gradient, such as

$$\nabla_o L(o, s \rightarrow s'), \quad s \rightarrow s' \sim \pi_o \qquad (42)$$

for edge-decomposable losses, or

$$\nabla_o L(o, \tau), \quad \tau \sim \pi_o \qquad (43)$$

for trajectory-decomposable losses, where π_o, called the **training distribution**, is a distribution over edges or trajectories that can be associated with $\Pi(o)$, corresponding to the online setting in RL, or defined in other ways, corresponding to the behavior policy in offline RL, see Sec. 3.3.3 below.

3.3 Extensions

In this section, we discuss possible relaxations to the GFlowNet training paradigm introduced thus far.

3.3.1 Introducing Time Stamps to Allow Cycles

Note that the state-space of a GFlowNet can easily be modified to accommodate an underlying state space for which the transitions do not form a DAG, e.g., to allow cycles. Let $\mathcal{S}$ be such an underlying state-space. Define the augmented state space $\mathcal{S}' = \mathcal{S} \times \mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of natural numbers, and $s'_t = (s_t, t)$ is the augmented state, where t is the position of the state s_t in the trajectory. With this augmented state space, we automatically avoid cycles. Furthermore, we may design or train the backwards transition probabilities $P_B(s'_t \mid s'_{t+1} = (s_{t+1}, t+1))$ to create a preference for shorter paths towards s_{t+1}, as discussed in Sec. 2.6. Note that we can further generalize this setup by replacing $\mathbb{N}$ with any totally ordered indexing set; the augmented state space will still have an associated DAG. The ordering "$<$" in the original state-space is lifted to the augmented state-space: $(s_t, t) < (s'_{t'}, t')$ if and only if $t < t'$ and $s_t < s'_t$.

3.3.2 STOCHASTIC REWARDS

We also consider the setting in which the given reward is stochastic rather than being a deterministic function of the state, yielding training procedures based on stochastic gradient descent. For example, with the trajectory balance loss of Eq. (41), if $R(s)$ is stochastic (even when given s), we can think of what is being really optimized is the squared loss with $\log R(s)$ replaced by its expectation (given s). This is a straightforward consequence of minimizing the expected value of a squared error loss (as for example in neural networks trained with a squared error loss and a stochastic target output, where the neural network effectively tries to estimate the expected value of that target).

3.3.3 GFLOWNETS CAN BE TRAINED OFFLINE

As discussed in Sec. 3.2, we do not need to train a GFlowNet using samples from its own trajectory distribution $\hat{P} = \Pi(o)$. Those training trajectories can be drawn from any training distribution π_T with full support, as already shown by Bengio et al. (2021). It means that a GFlowNet can be trained offline, as in offline reinforcement learning (Ernst et al., 2005; Riedmiller, 2005; Lange et al., 2012).

It should also be noted that with a proper adaptive choice of π_T, and assuming that computing R is cheaper or comparable in cost to running the GFlowNet on a trajectory, it should be more efficient to continuously draw new training samples from π_T than to rehearse the same trajectories multiple times. An exception would be rehearsing the trajectories leading to high rewards if these are rare.

How should one choose the training distribution π_T? It needs to cover the support of R but if it were uniform it would be very wasteful and if it were equal to the current GFlowNet policy π it might not have sufficient effective support and thus miss modes of R, i.e., regions where $R(x)$ is substantially greater than 0 but $R(x) \gg P_T(x)$. Hence the training distribution should be sampled from an exploratory policy that visits places that have not been visited yet and may have a high reward. High epistemic uncertainty around the current policy would make sense and the literature on acquisition functions for Bayesian optimization (Srinivas et al., 2010) may be a good guide. More generally, this means the training distribution should be adaptive. For example, π_T could be the policy of a second GFlowNet trained mostly to match a different reward function that is high when the losses observed by the main GFlowNet are large. It would also be good to regularly visit those trajectories corresponding to known large R, i.e., according to samples from π, to make sure those are not forgotten, even temporarily.

3.4 Exploiting Data as Known Terminating States

In some applications we may have access to a dataset of $(s, R(s))$ pairs and we would like to use them in a purely offline way to train a GFlowNet, or we may want to combine such data with queries of the reward function R to train the GFlowNet. For example, the dataset may contain examples of some of the high-reward terminating states s which would be difficult to obtain by sampling from a randomly initialized GFlowNet. How can we compute a gradient update for the GFlowNet parameters using such $(s, R(s))$ pairs?

If we choose to parametrize the backwards transition probabilities P_B (which is necessary for implementing the detailed balance loss), then we can just sample a trajectory

τ leading to s using P_B and use these trajectories to update the flows and forward transition probabilities along the traversed transitions. However, this alone is not guaranteed to produce the correct GFlowNet sampling distribution because the empirical distribution over training trajectories τ defined as above does not have full support. Suppose for example that the dataset only contains high-reward terminating states with $R(s) = 1$. The GFlowNet could then just sample trajectories uniformly (which would be wrong, we would like the probability of most states not in the training set to be very small). On the other hand, if we combine the distribution of trajectories leading to terminal transitions in the dataset with a training distribution whose support covers all possible trajectories, then the offline property of GFlowNet guarantees that we can recover a flow-matching model.

4. Conditional Flows and Free energies

A remarkable property of flow networks is that we can recover the normalizing constant Z from the initial state flow $F(s_0)$ (Prop. 10). Z also gives us the partition function associated with a given terminal reward function R specifying the terminating flows.

What about internal states s with $s_0 < s < s_f$? If we had something like a normalizing constant for only the terminating flows achievable from s, we would be able to obtain a form of marginalization given state s, i.e., a conditional probability for terminating states $s' \geq s$, given s. Naturally, one could ask: does the flow $F(s)$ through state s give us that kind of marginalization over only the downstream terminating flows? Unfortunately in general, the answer to this question is *no*, as illustrated in Fig. 5: in this example $F(s_2) = 4$, whereas the sum of terminating flows achievable from s_2 is 6 (the terminating states reachable from s_2 are $\{s_5, s_6, s_7\}$). The discrepancy is caused by the flow through (s_0, s_1, s_5) that contributes to the terminating flow $F(s_5 \rightarrow s_f)$, but not to $F(s_2)$ since there is no order relation between s_1 and s_2.

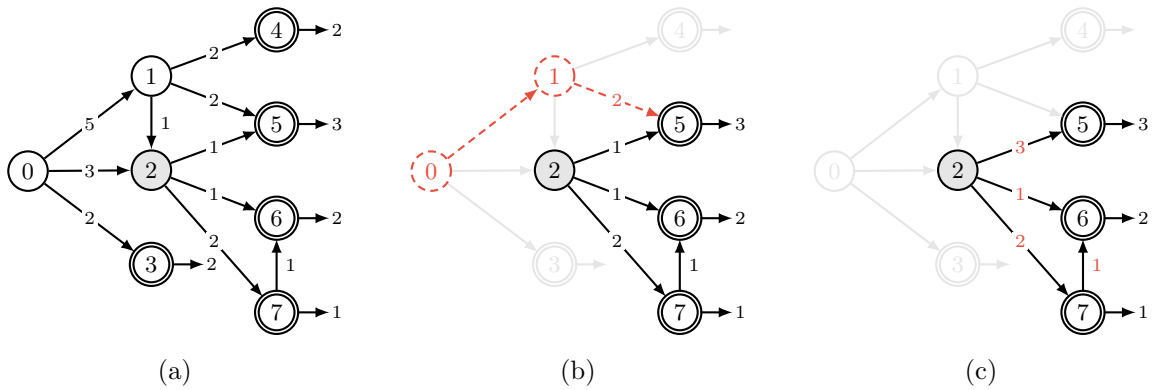(a)   (b)   (c)

Figure 5: Example of a state-conditional flow network. (a) The original (Markovian) flow network. (b) The subgraph of states reachable from s_2; there is a flow through (s_0, s_1, s_5) that contributed to $F(s_5 \rightarrow s_f)$, but not to $F(s_2)$, showing that $F(s_2)$ does not marginalize the rewards of its descendant. (c) State-conditional flow network F_{s_2}, which differs from the original flow F on the subgraph, but satisfies the desired marginalization property.

In Sec. 5.3, we show how GFlowNets applied to sampling sets of random variables can be used to estimate the marginal probability for the values given to a subset of the variables. It requires computing the kind of intractable sum discussed above (over the rewards associated with all the descendants of a state s, with s corresponding to such a subset of variables and a descendant to a full specification of all the variables). That motivates the following definition:

Definition 27 *Given a pointed DAG $G = (\mathcal{S}, \mathbb{A})$, the corresponding partial order denoted by $\geq$, and a function $\mathcal{E} : \mathcal{S} \to \mathbb{R}$, called the* **energy function***, we define the* **free energy** *$\mathcal{F}(s)$ of a state s as:*

$$e^{-\mathcal{F}(s)} := \sum_{s' : s' \geq s} e^{-\mathcal{E}(s')}. \tag{44}$$

Free energies are generic formulations for the marginalization operation (i.e. summing over a large number of terms) associated with energy functions, and we find their estimation to open the door to interesting applications where expensive MCMC methods would typically be the main approach otherwise.

4.1 Conditional flow networks

In Sec. 2.2, we defined a flow network as a DAG, augmented with some function F over the set of complete trajectories $\mathcal{T}$. We can extend this notion of flow networks by conditioning each component on some information x. In general, this conditioning variable can represent any conditioning information, either external to the flow network (but influencing the terminating flows), or internal (e.g., x can be a property of complete trajectories over another flow network, like passing through a particular state).

Definition 28 *Let $\mathcal{X}$ be a set of conditioning variables. We consider a family of DAGs $G_x = (\mathcal{S}_x, \mathcal{A}_x)$ indexed by $x \in \mathcal{X}$, along with a family of initial and terminal states denoted by $(s_0 \mid x) \in \mathcal{S}_x$ and $(s_f \mid x) \in \mathcal{S}_x$ respectively. For each DAG G_x, we denote by $\mathcal{T}_x$ the set of complete trajectories in G_x, and we denote by $\mathcal{T}$ their union:*

$$\mathcal{T} = \bigcup_{x \in \mathcal{X}} \mathcal{T}_x.$$

A ***conditional flow network*** *is the specification of $\mathcal{X}$, the family $\{G_x, \ x \in \mathcal{X}\}$, along with a conditional flow function F, i.e. a function $F : \mathcal{X} \times \mathcal{T} \to \mathbb{R}^+$ such that $F(x, \tau) = 0$ if $\tau \notin \mathcal{T}_x$. For clarity, we will denote, for each $x \in \mathcal{X}$, by F_x the function mapping each $\tau \in \mathcal{T}_x$ to $F(x, \tau)$. Similar to Sec. 2.2, F_x induces a measure of the σ-algebra $2^{\mathcal{T}_x}$ for each x.*

Conditional flow networks effectively represent a family of flow networks, indexed by the value of x. Since conditional flow networks are defined using the same components as an unconditional flow network, they inherit from all the properties of flow networks for all DAGs G_x and flow functions F_x. In particular, we can directly extend the notion of probability distribution over flows, state and edge flows, forward and backward transition probabilities (Sec. 2.3), of Markovian flows (Sec. 2.4), and any flow matching condition

(Sec. 2.5) to conditional flows; the only difference is that now every term explicitly depends of the conditioning variable x.

In Sec. 4.2 and Sec. 4.3, we will elaborate two important examples of conditional flow networks: flow networks conditioned on external information that changes the reward $R(s \mid x)$, and state-conditional flow networks that depend on internal information, i.e., previously visited states.

4.2 Reward-conditional flow networks

Definition 29 *Let $\mathcal{X}$ be a set of conditioning variables. Consider a flow network given by a pointed DAG $G = (\mathcal{S}, \mathbb{A})$ and a flow function F. Consider a family $\mathcal{R}$ of non-negative functions of $\mathcal{S}$: $\{R_x : \mathcal{S}^f \to \mathbb{R}^+, \ x \in \mathcal{X}\}$. A **reward-conditional flow network** compatible with the family $\mathcal{R}$ is a conditional flow network (Def. 28), with $G_x = G$ for every $x \in \mathcal{X}$, such that the edge-flow functions induced by the conditional flow function F satisfy:*

$$\forall x \in \mathcal{X} \quad \forall s \in \mathcal{S}^f \quad F_x(s \to s_f) = R_x(s).$$

We will use the notations $R_x(s)$ and $R(s \mid x)$ interchangeably.

Note that the definition above implies that all the DAGs of a reward-conditional flow network are identical, and only the terminating flows differ amongst the members of the family.

Example 7 *We will see in Sec. 4.4 that we can estimate a conditional flow network using a GFlowNet (Sec. 3), given a reward function $R(s \mid x)$. In an Energy-Based Model, the model $P_\theta(s)$ is associated with a given energy function $\mathcal{E}_\theta(s)$, parametrized by θ, with*

$$P_\theta(s) = \frac{\exp(-\mathcal{E}_\theta(s))}{Z(\theta)}.$$

This model can be parametrized using a reward-conditional flow network, conditioned on θ with the reward function $R(s \mid \theta) = \exp(-\mathcal{E}_\theta(s))$. We show in Sec. 4.5 how to use such a conditional flow-network to learn an Energy-Based Model.

4.3 State-conditional flow networks

Definition 30 *Consider a flow network given by a DAG $G = (\mathcal{S}, \mathbb{A})$ and a flow function F. For each state $s \in \mathcal{S}$, let G_s be the subgraph of G containing all the states s' such that $s' \geq s$. A **state-conditional flow network** is given by the family $\{G_s, \ s \in \mathcal{S}\}$, along with a conditional flow function $F : \mathcal{S} \times \mathcal{T} \to \mathbb{R}^+$, where $\mathcal{T} = \bigcup_{s \in \mathcal{S}} \mathcal{T}_s$, and $\mathcal{T}_s$ the set of complete trajectories in $\mathcal{G}_s$, that satisfies:*

$$F_s(s' \to s_f) = F(s' \to s_f). \tag{45}$$

Note that in the definition above, we abused the notation F to refer to both flow functions and edge flow functions, but also used F_s to refer to the conditional flow function (or the corresponding edge flow function) $\tau \mapsto F(s, \tau)$. Unlike the reward-conditional flow networks defined in Sec. 4.2, the structure of the DAG in a state-conditional flow network

depends on the anchor state s. In particular, this means that the initial state $(s_0 \mid s) = s$ changes, but the final state $(s_f \mid s) = s_f$ remains unchanged, for any state s.

Since the definition of a state-conditional flow network depends on an original flow network, we must ensure that this definition is indeed correct, i.e. that such a state-conditional flow network that satisfies the conditions in Eq. (45) exists.

Proposition 31 *For any flow network given by a DAG $G = (\mathcal{S}, \mathbb{A})$ and a flow F, we can define a state-conditional flow network as per Def. 30.*

Proof Let $s \in \mathcal{S}$ be a state. Since the structure of the DAG G_s is clearly well-defined, we just need to show that there exists a flow function $F_s : \mathcal{T}_s \to \mathbb{R}^+$ that satisfies Eq. (45). If such a function exists for every $s \in \mathcal{S}$, then it would suffice to define the conditional flow function $F : \mathcal{S} \times \mathcal{T} \to \mathbb{R}^+$ as:

$$F(s, \tau) = \begin{cases} F_s(\tau) & \text{if } \tau \in \mathcal{T}_s \\ 0 & \text{otherwise.} \end{cases}$$

Let $A_{s'|s}$ be the set of complete trajectories in $\mathcal{T}_s$ terminating in $s' \geq s$; the condition in Eq. (45) then reads:

$$F_s(s' \to s_f) = F_s(A_{s'|s}) = \sum_{\tau \in A_{s'|s}} F_s(\tau) = F(s' \to s_f). \tag{46}$$

Note that in Eq. (46), $F(s' \to s_f)$ is a given quantity because the flow F is known. Since the sets of trajectories $\{A_{s'|s}, , s' \geq s\}$ form a partition of all the complete trajectories $\mathcal{T}_s$, Eq. (46) is a system of linear equations, whose unknowns are $F_s(\tau)$ for all $\tau \in \mathcal{T}_s$, where each equation involves separate sets of unknowns. Therefore there exists at least a solution $F_s(\tau)$ of this system.

We can construct such a solution in the following way. For some $\tau \in \mathcal{T}_s$, we can first start by selecting the complete trajectories $\bar{\tau} \in \mathcal{T}$ that contain τ:

$$C_\tau = \{\bar{\tau} \in \mathcal{T} : \tau \subseteq \bar{\tau}\}.$$

The key difference between the DAG G and the subgraph G_s though is that G may contain trajectories that terminate in some $s' \geq s$ but do not pass through s, and those are therefore not covered by the trajectories of G_s. Let $U_{s'|s}$ be the set of complete trajectories of G defined as

$$U_{s'|s} = \{\bar{\tau} \in \mathcal{T} : \exists s'' > s, s'' \in \bar{\tau}, s' \in \bar{\tau}, s \notin \bar{\tau}\}.$$

For all $\tau \in \mathcal{T}_s$ such that τ terminates in some $s' \geq s$, we can therefore construct the flow $F_s(\tau)$ as

$$F_s(\tau) := F(C_\tau) + \frac{1}{n} F(U_{s'|s}),$$

where $n = |A_{s'|s}|$ is the number of trajectories $\tau' \in \mathcal{T}_s$ that terminate in s'. It is easy to verify that $F_s(\tau)$ is a solution of Eq. (46). ∎

While we saw in Prop. 10 that the initial flow $F(s_0)$ was equal to the partition function, the initial state-conditional flow also benefit from a marginalizing property, and is now related to the free energy at s.

Proposition 32 *Given a state-conditional flow network (G, F) as in Def. 30, for any state s, the initial flow of the state-conditional flow network corresponds to marginalizing the terminating flows $F(s' \to s_f)$ for $s' \geq s$:*

$$F_s(s_0 \mid s) = F_s(s) = \sum_{s' : s' \geq s} F(s' \to s_f) = \exp(-\mathcal{F}(s)),$$

where $\mathcal{F}(s)$ is the free energy associated to the energy function $\mathcal{E}(s') = -\log F(s' \to s_f)$.

Proof This is a direct consequence of Prop. 10, applied to the state-conditional flow function $\mathcal{F}_s$, along with Def. 27.

$$F_s(s_0 \mid s) = \sum_{\tau \in \mathcal{T}_s} F_s(\tau) = \sum_{s' : s' \geq s} F_s(s' \to s_f)$$
$$= \sum_{s' : s' \geq s} F(s' \to s_f) = \sum_{s' : s' \geq s} e^{-\mathcal{E}(s')}$$

∎

Note that the definition of state-conditional flow networks is consistent with our original definition of (unconditional) flow networks in Section 2.1, in the sense that the original flow network is a valid state-conditional flow networks anchored at the initial state.

Another quantity of interest that state-conditional flow networks allow us to evaluate, is the probability of terminating a trajectory in a state s' if all terminating edge flows were diverted towards an earlier state $s < s'$:

Corollary 33 *Consider a flow network given by a DAG $G = (\mathcal{S}, \mathbb{A})$ and a flow F, from which we define any state-conditional flow network, as per Def. 30. Given a state s, the flow function F_s induces a probability distribution over $\{s'' \in \mathcal{S}^f : s'' \geq s\} \subseteq \mathcal{S}^f$, that we denote by $P_T(. \mid s)$.*

Under this measure, the probability of terminating a trajectory in G_s in a state s' (i.e. the last edge of the trajectory is $s' \to s_f$) is:

$$P_T(s' \mid s) = \mathbf{1}_{s' \geq s} e^{-\mathcal{E}(s') + \mathcal{F}(s)}, \tag{47}$$

where $\mathcal{E}$ is the energy function mapping each state s' that is parent of s_f to $-\log F(s' \to s)$, and $\mathcal{F}$ is the corresponding free energy function.

Proof Because F_s is a flow function, Def. 11 and Prop. 10 tell us that:

$$P_T(s' \mid s) = \begin{cases} \dfrac{F_s(s' \to s_f)}{F_s(s)} & \text{if } s' \geq s \\ 0 & \text{otherwise} \end{cases}$$

Combining this with Prop. 32, and Eq. (45), we obtain for $s' \geq s$:

$$P_T(s' \mid s) = \mathbf{1}_{s' \geq s} \frac{F(s' \to s_f)}{e^{-\mathcal{F}(s)}}$$
$$= \mathbf{1}_{s' \geq s} \frac{e^{-\mathcal{E}(s')}}{e^{-\mathcal{F}(s)}}$$

■

4.4 Conditional GFlowNets

Similar to the way we used a GFlowNet to estimate the flow of a flow network, we can also use a (conditional) GFlowNet in order to estimate a conditional flow network, with given target reward functions. A conditional GFlowNet follows the construction presented in Section 3, with the exception that all quantities to be learned now depend on the conditioning variable $x \in \mathcal{X}$ (e.g., x is an additional input of the neural network).

All parametrizations and losses presented in Sec. 3.2 could in principle be used to train a conditional GFlowNet, regardless of the conditioning set. Below we discuss yet another loss, first presented in Deleu et al. (2022), that could be used to train both GFlowNets and conditional GFlowNets.

Example 8 *Given a family of DAGs G_x and reward functions R_x indexed by $x \in \mathcal{X}$, where each state $s \in G_x$ is terminating (i.e. is a parent of s_f), and following Exs. 2 and 3, we consider a parametrization given by the forward and backward transition probabilities $\mathcal{O}_P^x = \mathcal{O}_2^x \times \mathcal{O}_3^x$, where $\mathcal{O}_2^x$ (resp. $\mathcal{O}_3^x$) is the set of forward (resp. backward) probability functions $\hat{P}_F$ (resp. $\hat{P}_B$) consistent with G_x for every $x \in \mathcal{X}$, and $(\Pi_P^x, \mathcal{H}_P^x)$ defined as in Ex. 3. Each $(\mathcal{O}_P^x, \Pi_P^x, \mathcal{H}_P^x)$ is a flow parametrization of (G_x, R_x), which can be trained with an edge-decomposable flow-matching loss, as proved in Deleu et al. (2022), and defined for every $s \to s' \in \mathbb{A}^{-f}$:*

$$\mathcal{L}_{DB}(\hat{P}_F, \hat{P}_B, s \to s', x) = \left(\log \frac{R_x(s') P_B(s \mid s', x) P_F(s_f \mid s, x)}{R_x(s) P_F(s' \mid s, x) P_F(s_f \mid s', x)} \right)^2$$

4.5 Training Energy-Based Models with a GFlowNet

A GFlowNet can be trained to convert an energy function into an approximate corresponding sampler. Thus, it can be used as an alternative to MCMC sampling (Sec. 3.1). Consider the model $P_\theta(s)$ associated with a given parametrized energy function $\mathcal{E}_\theta(s)$ with parameters θ: $P_\theta(s) = \frac{e^{-\mathcal{E}_\theta(s)}}{Z}$. Sampling from $P_\theta(s)$ could be approximated by sampling from the terminating probability distribution $P_T(s)$ of a GFlowNet trained with target terminal reward $R(s) = e^{-\mathcal{E}_\theta(s)}$ (see Eq. (34)). In practice, $\hat{P}_T$ would be an estimator for the true P_θ because the GFlowNet training objective is not zeroed (insufficient capacity or finite training time). The GFlowNet samples drawn according to $\hat{P}_T$ could then be used to obtain a stochastic gradient estimator for the negative log-likelihood of observed data x with respect to parameters θ of an energy function $\mathcal{E}_\theta$:

$$\frac{\partial - \log P_\theta(x)}{\partial \theta} = \frac{\partial \mathcal{E}_\theta(x)}{\partial \theta} - \sum_s P_\theta(s) \frac{\partial \mathcal{E}_\theta(s)}{\partial \theta}. \tag{48}$$

An approximate stochastic estimator of the second term could thus be obtained by sampling one or more terminating states $s \sim \hat{P}_T(s)$, i.e., from the trained GFlowNet's sampler. Furthermore, if the GFlowNet's loss is 0, i.e. $\hat{P}_T = P_\theta$, the gradient estimator would be unbiased.

One could thus potentially jointly train an energy function $\mathcal{E}_\theta$ and a corresponding GFlowNet by alternating updates of θ using the above equation (with sampling from P_θ replaced by sampling from $\hat{P}_T$) and updates of the GFlowNet using the updated energy function for the target terminal reward.

If we fix $\hat{F}(s{\to}s_f) = R(s)$ by construction (which we can do if the reward function is deterministic), then we can parametrize the energy function with the same neural network that computes the flow, since $\mathcal{E}(s) = -\log R(s) = -\log \hat{F}(s{\to}s_f)$. Hence *the same parameters are used for the energy function and for the GFlowNet*, which is appealing.

The above strategy for learning jointly an energy function and how to sample from it could be generalized to **learning conditional distributions by using a conditional GFlowNet** instead. Let x be an observed random variable and h be a hidden variable, with the GFlowNet generating the pair (x, h) in two sub-trajectories: either first generate x and then generate h given x, or first generate h and then generate x given h. This can be achieved by introducing a 6-valued component u in the state to make sure that both h and x are generated before exiting into s_f, with the following values and constraints:

$$s = s_0 \Rightarrow u = 0 \tag{49}$$

$$s = s_f \Rightarrow u = 5 \tag{50}$$

$$(u_t{\to}u_{t+1}) \in \{0{\to}1, 0{\to}2, 1{\to}1, 2{\to}2, 1{\to}3, 2{\to}4, 3{\to}3, 4{\to}4, 3{\to}5, 4{\to}5\} \tag{51}$$

where $u = 1$ indicates that x is being generated (before h), $u = 2$ indicates that h is being generated (before x), $u = 3$ indicates that h is being generated (conditioned on x), and $u = 4$ indicates that x is being generated (given h). The GFlowNet cannot reach the final state s_f until both x and h have been generated. The conditional GFlowNet can thus approximately sample $P_T(x)$, $P_T(h \mid x)$, $P_T(h)$, $P_T(x \mid h)$ as well as $P_T(x, h)$. If we only want to sample x (or only h), we allow exiting as soon as it is generated (resp. h is generated). See Sec. 5.3 for a more general discussion on how to represent, estimate and sample marginal distributions.

Let us denote by $P_\theta(x, h) \propto e^{-\mathcal{E}_\theta(x,h)}$ the joint distribution over (x, h) associated with the energy function, i.e., with $F(s \to s_f)$. When x is observed but h is not, θ could thus be updated by approximating the marginal log-likelihood gradient

$$\frac{\partial - \log P_\theta(x)}{\partial \theta} = \sum_h P_T(h \mid x) \frac{\partial \mathcal{E}_\theta(x, h)}{\partial \theta} - \sum_{x',h} P_T(x', h) \frac{\partial \mathcal{E}_\theta(x', h)}{\partial \theta} \tag{52}$$

using samples from the estimated terminal sampling probabilities $\hat{P}_T$ of a trained GFlowNet to approximate in a stochastic gradient way the above sums (using one or a batch of samples).

Note how we now have outer loop updates (of the energy function, i.e., the reward function) from actual data, and an inner loop updates (of the GFlowNet) using the energy function as a driving target for the GFlowNet. How many inner loop updates are necessary

for such a scheme to work is an interesting open question but most likely depends on the form of the underlying data generating distribution. If the work on GANs (Goodfellow et al., 2014) is a good analogy, a good strategy may be to interleave updates of the energy function (as minus the log-terminal flow of a GFlowNet) based on a batch of data, and updates of the GFlowNet as a sampler based on both these samples (trajectories can be sampled backwards from a terminating state s using P_B) and forward samples from the tempered training policy π_T defined by the forward transition probabilities of the GFlowNet.

4.6 Active Learning with a GFlowNet

An interesting variant on the above scheme is one where the GFlowNet sampler is used not just to produce negative examples for the energy function but also to actively explore the environment. Jain et al. (2022) use an active learning scheme where the GFlowNet is used to sample candidates x for which we expect the reward $R(x)$ to be generally large (since the GFlowNet approximately samples proportionally to $R(x)$). The challenge is that evaluating the true reward R for any x is computationally expensive and can potentially be noisy (for example, a biological assay to measure the binding energy of a drug to a given target protein). Thus, instead of using the true reward directly, the authors introduce a proxy $\hat{f}$ (which approximates the true reward function f), which is used to train the GFlowNet. This would lead to a setup similar to Sec. 4.5, with an inner loop where a GFlowNet is trained to match the proxy $\hat{f}$, and an outer-loop where the proxy $\hat{f}$ is learned in a supervised fashion using (x, y) pairs, where x is proposed by the GFlowNet, and y is the corresponding *true* reward from the environment (for example, outcome of a biological of chemical assay). It is important to note here that the GFlowNet and the proxy are intricately linked since the coverage of proxy $\hat{f}$ over the domain of x relies on diverse candidates from the GFlowNet. And similarly, since the GFlowNet matches a reward distribution defined by the proxy reward function $\hat{f}$, it also depends on the quality of the true reward function f.

This setup can be further extended by incorporating information about how *novel* a given candidate is, or how much epistemic uncertainty, $u(x, f)$, there is in the prediction of $\hat{f}$. We can use the acquisition function heuristics (like Upper Confidence Bound (UCB) or Expected Improvement (EI)) from Bayesian optimization (Močkus, 1975; Srinivas et al., 2010) to combine the predicted usefulness $\hat{f}(x)$ of configuration x with an estimate of the epistemic uncertainty around that prediction. Using this as the reward can allow the GFlowNet to explore areas where the predicted usefulness is high ($\hat{f}(x)$ is large) and at the same time explore areas where there is more information to be gathered about useful configurations of x. The uncertainty over the predictions of $\hat{f}$ with the appropriate acquisition function can provide more control over the exploratory behaviour of GFlowNets.

As discussed by Bengio et al. (2021) when comparing GFlowNets with return-maximizing reinforcement learning methods, an interesting property of sufficiently trained GFlowNets is that they will sample from all the modes of the reward function, which is particularly desirable in a setting where exploration is required, as in active learning. The experiments in the paper also demonstrate this advantage experimentally in terms of the diversity of the solutions sampled by the GFlowNet compared with PPO, an RL method that had previously been used for generating molecular graphs and that tends to focus on a single mode of the reward function.

4.7 Estimating Entropies, Conditional Entropies and Mutual Information

Definition 34 *Given a reward function R with $0 \leq R(s) < 1 \ \forall s$, we define the* **entropic reward function** R' *associated with R as:*

$$R'(s) = -R(s) \log R(s). \tag{53}$$

In brief, in this section, we show that we can estimate entropies by training two GFlowNets: one that estimates flows as usual for a target terminal reward function $R(s)$, and one that estimates flows for the corresponding entropic reward function. We show below that we obtain an estimator of entropy by looking up the flow in the initial state, and if we do this exercise with conditional flows, we get conditional entropy. Once we have the conditional entropy, we can also estimate the mutual information.

Proposition 35 *Consider a flow network (G, F) such that the terminating flows match a given reward function R, i.e. $\forall s \in \mathcal{S}^f$, $F(s \rightarrow s_f) = R(s)$, with $R(s) < 1$ for all s, and a second flow network (G, F') with the same pointed DAG, but with a flow function for which the terminating flows match the entropic reward function R' (Eq. (53)), then the entropy $H[S]$ associated with the terminating state random variable $S \in \mathcal{S}^f$ with distribution $P_T(S = s) = \frac{R(s)}{Z}$ (Eq. (18)) is*

$$H[S] := -\sum_s P_T(s) \log P_T(s) = \frac{F'(s_0)}{F(s_0)} + \log F(s_0). \tag{54}$$

Proof First apply the definition of $P_T(s)$, then Eq. (11) on both flows:

$$
\begin{aligned}
-\sum_s P_T(s) \log P_T(s) &= -\sum_s \frac{R(s)}{F(s_0)} (\log R(s) - \log F(s_0)) \\
&= \frac{\left(-\sum_s R(s) \log R(s) \right) + \left(\log F(s_0) \sum_s R(s) \right)}{F(s_0)} \\
&= \frac{F'(s_0)}{F(s_0)} + \log F(s_0).
\end{aligned}
$$

Note that we need $R(s) < 1$ to make sure that the rewards $R'(s)$ (and thus the flows) are positive. ∎

Proposition 36 *Given a set $\mathcal{X}$ of conditioning variables, consider a conditional flow network defined by a conditional flow function F, for which the terminating flows match a target reward R family (conditioned on $x \in \mathcal{X}$) that satisfies $R_x(s) < 1$ for all s, and a second conditional flow network defined by a conditional flow function F', for which the terminating flows match the entropic reward functions R'_x (Eq. (53)), then the conditional entropy $H[S \mid x]$ of random terminating states $S \in \mathcal{S}^f$ consistent with condition x is given by*

$$H[S \mid x] = \frac{F'(s_0 \mid x)}{F(s_0 \mid x)} + \log F(s_0 \mid x). \tag{55}$$

In particular, for a state-conditional GFlowNet ($\mathcal{X} = \mathcal{S}$ is the state space of the DAG), we obtain

$$H[S \mid s] = \frac{F'(s \mid s)}{F(s \mid s)} + \log F(s \mid s). \tag{56}$$

More generally, the mutual information $MI(S; X)$ between the random draw of a terminating state $S = s$ according to $P_T(s \mid x)$ and the conditioning random variable X is

$$\mathrm{MI}(S; X) = H[S] - E_X[H[S \mid X]] = \frac{F'(s_0)}{F(s_0)} + \log F(s_0) - E_X\left[\frac{F'(s_0 \mid X)}{F(s_0 \mid X)} + \log F(s_0 \mid X)\right] \tag{57}$$

where $F(s)$ and $F'(s)$ indicate the unconditional flows (trained with no condition x given) while $F(s \mid x)$ and $F'(s \mid x)$ are their conditioned counterparts.

Proof The proof of Eq. (55) follows from the fact that each (G_x, F_x) is a flow network, to which we can apply Prop. 35. Eq. (57) is a direct consequence of the definition of the Mutual Information, Eq. (54) and Eq. (55). ∎

If we have a sampling mechanism for $P(X)$, we can thus approximate the expectation in Eq. (57) by a Monte-Carlo average with draws from $P(X)$.

5. GFlowNets on Sets, Graphs, and to Marginalize Joint Distributions

5.1 Set GFlowNets

We first define an action space for constructing sets and we view the GFlowNet as a means to generate a random set S and to estimate quantities like probabilities, conditional probabilities or marginal probabilities for realizations of this random variable. The elements of those sets are taken from a larger "universe" set $\mathcal{U}$.

Definition 37 *Given a "universe" set $\mathcal{U}$, consider the pointed DAG $G = (\mathcal{S}, \mathbb{A})$, where $\mathcal{S} := 2^{\mathcal{U}} \cup \{s_f\}$ is the set of all subsets of $\mathcal{U}$ with an additional state s_f, $s_0 = \{\}$ is the empty set, and for any two subsets s, s' of $\mathcal{U}$, $s \to s' \in \mathbb{A} \Leftrightarrow \exists a \in \mathcal{U} \setminus s,\ s' = s \cup \{a\}$; meaning that each transition in the DAG corresponds to adding one element of $\mathcal{U}$ to the current subset. Additionally all subsets are connected to s_f, i.e. $\forall s \in \mathcal{S},\ s \to s_f \in \mathbb{A}$. A **set flow network** is a flow network on this graph G, and a **set GFlowNet** is an estimator of such a flow network, as defined in Sec. 3. The target terminal reward function $R : s \mapsto F(s \to s_f)$ satisfies:*

$$Z = \sum_{s \in 2^{\mathcal{U}}} R(s) < \infty. \tag{58}$$

A set flow network defines a terminating probability distribution P_T on states (see Def. 13 and Eq. (44)), with

$$P_T(s) = e^{-\mathcal{E}(s) + \mathcal{F}(s_0)} = \frac{F(s \to s_f)}{F(s_0)} \tag{59}$$

where $\mathcal{E}$ represents the energy $-\log R$. Similarly, Cor. 33 provides us with a formula for conditional probabilities of a given superset s' of a given set s under P_T,

$$P_T(s' \mid s' \supseteq s) := e^{-\mathcal{E}(s') + \mathcal{F}(s)} = \frac{F(s' \to s_f)}{F(s \mid s)} \tag{60}$$

where $\mathcal{F}$ indicates free energy (see Def. 27).

Remember that with a GFlowNet with state and edge flow estimator $\hat{F}$, it is not guaranteed that $\hat{F}(s) = R(s)$ for all states $s \in \mathcal{S} \setminus \{s_f\}$, so we could estimate probabilities with

$$\hat{P}_T(s) = \frac{\hat{F}(s \to s_f)}{\hat{F}(s_0)} \tag{61}$$

or alternatively

$$\hat{P}_T(s) = \frac{R(s)}{\hat{F}(s_0)}. \tag{62}$$

Similarly, we can estimate conditional superset probabilities with Eq. (60) or with

$$\hat{P}_T(s' \mid s' \supseteq s) = \frac{R(s')}{\hat{F}(s \mid s)}, \tag{63}$$

none of which are guaranteed to exactly sum to 1.

We can also compute the marginal probability over all supersets of a given set s, as shown below.

Proposition 38 *Following the notations of Def. 37, let $\mathfrak{S}(s) = \{s' \supseteq s\}$ be the set of all supersets of a set s. The probability of drawing any element from $\mathfrak{S}(s)$ given a set flow network is*

$$P_T(\mathfrak{S}(s)) = \sum_{s' \supseteq s} P_T(s') = \frac{e^{-\mathcal{F}(s)}}{Z} = \frac{F(s \mid s)}{F(s_0)}. \tag{64}$$

Proof We can rewrite the sum as follows, first applying the definition of P_T (Eq. (18)), and Prop. 32:

$$\begin{aligned}
\sum_{s' \supseteq s} P_T(s') &= \sum_{s' \geq s} \frac{F(s' \to s_f)}{F(s_0)} \\
&= \frac{F(s \mid s)}{F(s_0)} \\
&= \frac{e^{-\mathcal{F}(s)}}{Z}
\end{aligned}$$

where we notice that for states that are sets, the order relationship $s \leq s'$ is equivalent to the subset relationship $s \subseteq s'$. ∎

To summarize, a GFlowNet which is trained to match a given energy function (i.e. derived rewards) over sets can be used to represent that distribution, sample from it, estimate the probability of a set under it, estimate the partition function, search for the lowest energy set, sample a conditional distribution over supersets of a given set, estimate that conditional distribution for a given pair of set and superset, compute the marginal probability of a subset (i.e., summing over the probabilities of the supersets), and compute the entropy of the set distribution or of the conditional distribution of supersets of a set. For example, using a GFlowNet to model distributions over sets has been successfully used in

Malik et al. (2023), where the goal is to iteratively select an informative batch of points for efficient active learning. The authors additionally show that the distribution over sets defined by a mutual information-based reward (Sec. 4.7) can be approximated to satisfying levels using a neural network as a function approximator for the GFlowNet policy.

5.2 GFlowNet on Graphs

A graph is a special kind of set in which there are two kinds of elements: nodes and edges, with edges being pairs of node indices. Graphs may also have content attached to nodes and/or edges. The set operations described in the previous section can thus be specialized accordingly. Some actions (i.e. edges in the GFlowNet DAG) could insert a node while other actions could insert an edge. The set of allowable actions can be limited, for example to make sure the graph has a single connected component, or to ensure acyclicity. Like for sets in general, one cannot have an action which adds a node or edge which is already in the set. Deleu et al. (2022) used a GFlowNet over DAGs in order to learn an approximation of the posterior distribution over the graphical structure of a Bayesian Network. While the GFlowNet was learned by minimizing some loss derived from flow-matching conditions as in Sec. 3.2, they showed that the resulting distribution is an accurate approximation of the target posterior. Since graphs are sets, all the GFlowNet operations on sets can be applied on graphs.

5.3 Marginalizing over Missing Variables

The ability of GFlowNets to capture probability distributions over sets can be applied to modeling the joint distribution over random variables, to calculating marginal probabilities over given subsets of variable values, and to sampling or computing probabilities for any conditional (e.g., for a subset of variables given another subset of variables).

Let $X = (X_1, X_2, \ldots, X_n)$ be a composite random variable with n element random variables X_i, $1 \leq i \leq n$, each with possible values $x_i \in \mathcal{X}_i$ (not necessarily numbers). If we are given an energy function or a terminal reward function $R(x)$ to score any instance $X = x$, we can train a particular kind of set GFlowNet for which the set elements are pairs (i, x_i) and at most one element in the set with index i, for all i. The only allowed terminating transitions are when the set has exactly size n and every size-n set s terminates on the next transition.

Note how that GFlowNet can sample an X in any possible order, if P_B allows that order. Given an existing set of (i, x_i) pairs (represented by a state $s = \{(i, x_i)\}_i)$, we can estimate the marginal probability of that subset of variables (implicitly summing over all the missing ones, see Eq. (64)). We can sample the other variables by setting the state at s and continuing to sample from the GFlowNet's policy (the learned forward transition probability function). We can sample from a chosen subset S' of the other variables by constraining that policy to only add elements which are in S'. In addition, we can do all the other things that are feasible on set GFlowNets, such as estimating the partition function, sampling in an order which prefers the early subsequences with the largest marginal probability, searching for the most probable configuration of variables, or estimating the entropy of the distribution.

5.4 Modular Energy Function Decomposition

Let us see how we can apply the graph GFlowNet framework to a special kind of graph: a factor graph (Kschischang et al., 2001) with reusable factors. This will yield a distribution $P_T(g)$ over graphs g, each of which is associated with an energy function value $\mathcal{E}(g)$ (and associated reward $R(g)$). Energy-based models are convenient because they can decompose a joint probability into independent pieces (possibly corresponding to independent mechanisms, Schölkopf et al., 2012; Goyal et al., 2019; Goyal and Bengio, 2020), each corresponding to a *factor* of a factor graph. In our case, we would like a shared set of factors $\mathbb{F}$ to be reusable across many factor graphs g. The factor graph will provide an energy and a probability over a set of random variables $\mathcal{V}$. Let the graph $g = \{(F^i, v^i)\}_i$ be written as a set of pieces (F^i, v^i), where $F^i \in \mathbb{F}$ is the index of a factor with energy function term $\mathcal{E}_{F^i}$, selected from the pool $\mathbb{F}$ of possible factors, and where $v^i = (v_1, v_2, \ldots)$ is a list of realizations of the random variables $V_j \leftarrow v_j$, where $V_j \in \mathcal{V}$ is a node of the factor graph. That list defines the edges of the factor graph connecting variable V_j with the j-th argument of $\mathcal{E}_{F^i}$. Let us denote $\mathcal{E}_{F^i}(v^i)$ the value of this energy function term $\mathcal{E}_{F^i}$ applied to those values v^i, i.e.,

$$\mathcal{E}_{F^i}(v^i) = \mathcal{E}_{F^i}(v_1, v_2, \ldots). \tag{65}$$

The total energy function of such a graph can then be decomposed as follows

$$\mathcal{E}(g) = \sum_i \mathcal{E}_{F^i}(v^i). \tag{66}$$

What is interesting with this construction is that the graph GFlowNet can now sample a graph g, possibly given some conditioning observations x: see Sec. 4.5 on how GFlowNets can be trained jointly with an energy function, including the case where only some random variables are observed. Hence, given some observed variables (not necessarily always the same), the graph GFlowNet can sample a latent factor graph containing and connecting (with energy function terms) both observed and latent random variables, and whose structure defines an energy function over values of the joint observed and latent variables.

Not only can we use the compositional nature of the objects generated by a GFlowNet to decompose the total energy into reusable energy terms corresponding to ideally independent mechanisms, but we can also decompose the GFlowNet itself into modules associated with each mechanism. The action space of this graph GFlowNet is fairly complex, with each action corresponding with the addition of a latent variable V_k or the addition of a graph piece (F^i, v^i). Such an action is taken in the context of the state of the GFlowNet, which is a partially constructed graph (arising from the previous actions). The GFlowNet and its associated energy function parameters are thus decomposed into modules. Each module knows how to compute an energy function $\mathcal{E}_{F_i}$ and how to score and sample competitively (against the other modules) a new graph piece (to insert the corresponding factor in the graph).

Consider some observed variables (a subset of the V_j's with their values v_j), collectively denoted x. Consider a graph g among those compatible with x (i.e. with some nodes corresponding to $V_j = v_j$ for the observed variables) and denote h the specification of the part of g not already provided by x. We can think about latent variable h as the explanation for the observed x. Note how marginalizing over all the possible h, we can compute the

free energy of x. The principles of Sec. 4.5 can be applied to train such an energy-based GFlowNet. It also makes sense to represent a prior over graph structures in the energy function. For example, we may prefer sparse factors (with few arguments), and we may introduce soft or hard constraints having to do with a notion of type that is commonly used in computer programming and in natural language. Each random variable in the graph can have as one of its attributes a type, and each factor energy function argument can expect a type. Energy function terms can be added to construct this prior by favouring graph pieces $(F_i, v^i = (v_k)_k)$ in which the type of variable V_k (of which v_k is a realization) matches the type expected of the k-th argument of F_i. This is very similar to attention mechanisms (Bahdanau et al., 2014; Vaswani et al., 2017), which can be seen to match a query (an expected type) with a key (the type associated with an element). For instance, Pan et al. (2023) propose the Forward-Looking GFlowNet framework that exploits the modularity of the energy function, as in Eq. (66) and obtain a better approximation of the target distribution than those resulting from regular losses.

6. Continuous or Hybrid Actions and States

All of the mathematical developments above have used sums over states or actions, with the idea that these would be elements of a discrete space. However, for the most part one can replace these sums by integrals in case the states or actions are either continuous or hybrid (with some discrete components and some continuous components). Beyond this, we discuss below what the presence of continuous-valued actions and states changes to the GFlowNet framework.

Although there are explicit sums respectively over successors and predecessors which come up in Eq. (40), such sums are also hiding in the detailed balance constraint of Eq. (26). Indeed, these sums are implicit as part of the normalizing constant in the conditional density of the next state or previous state in $P_F(s_{t+1} \mid s_t)$ and $P_B(s_t \mid s_{t+1})$. We consider below ideas to deal with this challenge.

6.1 Integrable Normalization Constants

We first note that if we can handle a continuous state, we can also handle a hybrid state, as follows. Let the state be decomposed into

$$s = (s^i, s^x) \tag{67}$$

where s^i is discrete and s^x is continuous. Then we can decompose any of the transition conditionals as follows:

$$P_F(s_{t+1} \mid s_t) = P(s_{t+1}^x \mid s_{t+1}^i, s_t)P(s_{t+1}^i \mid s_t). \tag{68}$$

We note that this is formally equivalent to decomposing the transition into two transitions, first to perform the discrete choice into the next state, and second to perform the continuous choice into the next state (given the discrete choice). Having continuous-valued inputs to a neural net is no problem. The challenge is to represent continuous densities on the output, with the need to both being able to compute the density of a particular value (say $P(s_{t+1}^x \mid s_{t+1}^i, s_t)$) and to be able to sample from it. Computing categorical probabilities and sampling

from a conditional categorical is standard fare, so we only discuss the continuous conditional. One possibility is to parametrize $s_{t+1}^x \mid s_{t+1}^i, s_t$ with a density for which the normalization constant is a known tractable integral, like the Gaussian. However, that may limit capacity too much, and may prevent a good minimization of the detailed balance or flow-matching loss. One workaround is to augment the discrete part of the state s_i with extra dimensions corresponding to "cluster IDs", i.e., partition the continuous density into a mixture. We know that with enough mixture components, we can arbitrarily well approximate densities from a very large family. Other approaches include modeling the conditional density with an autogressive or normalizing flow model (Rezende and Mohamed, 2015, with a different meaning of the word flow), or, like in denoising diffusion models (Sohl-Dickstein et al., 2015), decomposing sampling of s^x into several resampling steps, transforming its ditribution from a simple one to complex one.

To guarantee that the detailed balance constraint can be exactly satisfied, we could go further and think about parametrizing the edge flow $F((s_i, s_x) \rightarrow (s_i', s_x'))$, and note that this is the natural parametrization if we use the node-based flow-matching loss. For example, keeping with the Gaussian example, we would now have a joint Gaussian energy in the vector (s_x, s_x') for each feasible discrete component indexed by (s_i, s_i'). Note that in practical applications of GFlowNets, not all the transitions that satisfy the order relationship are generally allowed in the GFlowNet's underlying DAG. For example, with set GFlowNets, the only allowed actions add one element to the set (not an arbitrary number of elements). These constraints on the action space mean that the number of legal (s_i, s_i') pairs is manageable and correspond to the number of discrete actions. The overall action is therefore seen as having a discrete part (choosing s_i' given s_i) and a continuous part (choosing s_x' given s_i', s_i and s_x). With such a joint flow formulation, the forward and backward conditional densities can be computed exactly and be compatible with each other.

6.2 GFlowNets in GFlowNets

Another way to implement an edge flow involving continuous variables is to use a lower-level ⟨GFlowNet, energy function⟩ pair to represent its flow, conditional probabilities and sample from them. Remember that such a pair can be trained following the approach discussed in Sec. 4.5. Instead of a joint Gaussian for (s_x, s_x') given (s_i, s_i') we could have a smaller-scale GFlowNet and energy function (representing an edge flow in the outer GFlowNet) to handle a whole family of transitions of a particular type in a larger-scale outer GFlowNet. Imagine that we have a fairly arbitrary energy function for such a transition, with parameters that we will learn. Then we can also train a GFlowNet to sample in either direction (either from s_t to s_{t+1} or from s_{t+1} to s_t) and to evaluate the corresponding normalizing constants (and hence, the corresponding conditional probabilities). The discrete aspect of the state and of its transition may correspond to a family of transitions (e.g., insert a particular type of node in a graph GFlowNet), and a separate ⟨GFlowNet, energy function⟩ module may be specialized and trained to handle such transitions.

While this paper was under review, Lahlou et al. (2023) extended the theory of GFlowNets to more general state spaces, including continuous ones, and experimentally validated that the usual losses can be used to effectively perform inference in continuous domains.

7. Related Work

There are several classes of related literature that concern the problem of generating a diversity of samples, given some energy or reward signal, in particular:

- generative models (in particular deep learning ones),
- RL methods that maximize reward with some form of exploratory behavior or smoothness prior,
- MCMC methods that solve the problem of sampling from $p(x) \propto f(x)$ *in principle*,
- evolutionary methods, that can leverage group diversity through iterations over a population of solutions.

In what follows we discuss these and offer insights into similarities and differences between GFlowNets and these approaches. Note that the literature related to this problem is much larger than we can reference here, and extends to many other subfields of ML, such as GANs (Kumar et al., 2019), VAEs (Kingma and Welling, 2013; Kusner et al., 2017), and normalizing flows (Dinh et al., 2014, 2016; Rezende and Mohamed, 2015). Yet another related type of approach are the Bayesian optimization methods (Močkus, 1975; Srinivas et al., 2010), which have also been used for searching in the space of molecules (Griffiths and Hernández-Lobato, 2017). The main relation with Bayesian optimization methods is that GFlowNets are generative and can thus complement Bayesian optimization methods which scan a tractable list of candidates. When the search space is too large to be able to separately compute a Bayesian optimization acquisition function score on every candidate, using a generative model is appealing. In addition, GFlowNets are used to explore the modes of the distribution rather than to search for the single most dominant mode. This difference is similar to that with classical RL methods, discussed further below.

7.1 Contrast with Generative Models

The main difference between GFlowNets and established deep generative models like VAEs or GANs is that whereas the latter are trained by being provided a finite set of examples sampled from the distribution of interest, a GFlowNet is normally trained by being provided an energy function or a reward function.

This reward function tells us not just about the samples that are likely under the distribution of interest (which we can think of as positive examples) but also about those that are unlikely (which we can think of as negative examples) and also about those in-between (whose reward is not large but is not zero either). If we think of the maximum likelihood training objective in those terms, it is like a reward function that gives a high reward to every training example (seen as a positive example, where the probability should be high) and a zero reward everywhere else. However, other reward functions are possible, as seen in the application of GFlowNets to the discovery of new molecules (Bengio et al., 2021), where the reward is not binary and increases monotonically as a function of the value of a desirable property of the candidate molecule.

Note however that the difference with other generative modeling approaches blurs when we include the learning of the energy function along with the learning of the GFlowNet sampler, as outlined in Sec. 4.5. In that case, the pair comprising the trainable GFlowNet sampler and the trainable energy function achieves a similar objective as a trainable generative

model. Note that GFlowNets have been designed for generating discrete variable-size compositional structures (like sets or graphs), for both latent and observed variables, whereas GANs, VAEs or normalizing flows start from the point of view of modeling real-valued fixed-size vectors using real-valued fixed-size latent variables.

An interesting difference between GFlowNets and most generative model training frameworks (typically some variation on maximum likelihood) is in the very nature of the training objective for GFlowNets, which came about in the context of active learning scenarios. Whereas the GFlowNet training pairs $(s, R(s))$ can come from any distribution over s (any full-support training policy π_T), which does not have to be stationary (and indeed will generally not be, in an active learning setting), the maximum likelihood framework is very sensitive to changes in the distribution of the data it sees. This is connected to the "offline learning" property of the flow matching objective (Sec. 3.3.3, among others).

7.2 Contrast with Regularized Reinforcement Learning

The flow-matching loss of GFlowNets (Bengio et al., 2021) arose from the inspiration of the temporal-difference training (Sutton and Barto, 2018) objectives associated with the Bellman equation. The flow-matching equations are analogous to the Bellman equation in the sense that the training objective is local (in time and states), credit assignment propagates through a bootstrap process and tries to fix the parametrization so that these equations are satisfied, knowing that if they were (everywhere), we would obtain the desired properties. However, these desired properties are different, as elaborated in the next paragraph. The context in which GFlowNets were developed is also different from the typical way of thinking about agents learning in some environment: we can think of the deterministic environments of GFlowNets as involving internal actions typically needed by a cognitive agent that needs to perform some kind of inference through a sequence of steps (predict or sample some things given other things), i.e., through actions internal to the agent and controlling its computation. This is in contrast with the origins of RL, focused on the actions of an agent in an external and unknown stochastic environment. GFlowNets were introduced as a tool for learning an internal policy, similar to the use of attention in modern deep learning, where we know the effect of actions, and the composition of these actions defines an inference machinery for that agent.

Classical RL (Sutton and Barto, 2018) control methods work by maximizing return in Markov Decision Processes (MDPs); their focus is on finding the policy $\pi^* \in \text{argmax}_\pi V^\pi(s) \; \forall s$ maximizing the expected return $V^\pi(s)$, which happens to provably be achieved with a *deterministic* policy (Sutton and Barto, 2018), even in stochastic MDPs. In a deterministic MDP, of interest here, this means that training an RL agent is a search for the most rewarding trajectory, or in the case of terminal-reward-only MDPs (again of interest here), the most rewarding terminating state.

Another perspective, that emerged out of both the probabilistic inference literature (Toussaint and Storkey, 2006) and the bandits literature (Auer et al., 2002), is concerned with finding policies of the form $\pi(a|s) \propto f(s, a)$. It turns out that maximizing both return and *entropy* (Ziebart et al., 2008) of policies in a control setting yield policies such that

$$p(\tau) = \left[p(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t) \right] \exp \left(\eta \sum_{t=0}^{T-1} R(s_t, a_t) \right) \tag{69}$$

where $\tau = (s_0, a_0, s_1, a_1, \dots, s_T)$ and η can be seen as a temperature parameter. This result can also be found under the control-as-inference framework (Haarnoja et al., 2017; Levine, 2018). In deterministic MDPs with terminal rewards and no discounting of future rewards, this simplifies to $p(\tau) \propto \exp(\eta\rho(\tau))$, where ρ is the return.

In recent literature, this entropy maximization (MaxEnt) is often interpreted as a regularization scheme (Nachum et al., 2017), entropy being used either as an intrinsic reward signal or as an explicit regularization objective to be maximized. Another way to understand this scheme is to imagine ourselves in an adversarial bandit setting (Auer et al., 2002) where each arm corresponds to a unique trajectory, drawn with probability $\propto \exp(\rho(\tau))$.

An important distinction to make between MaxEnt RL and GFlowNets is that, in the general case they do not find the same result. A GFlowNet learns a policy such that $P_T(s) \propto R(s)$, whereas MaxEnt RL (with appropriately chosen temperature and R) learns a policy such that $P_T(s) \propto n(s)R(s)$, where $n(s)$ is the number of paths in the DAG of all trajectories that lead to s (a proof is provided in Bengio et al., 2021). An equivalence only exists if the DAG minus s_f is a tree rooted at s_0, which has been found to be useful (Buesing et al., 2019). What this overweighting by a factor $n(s)$ means practically is that states corresponding to longer sequences (which typically will have exponentially more paths to them) will tend to be sampled much more often (typically exponentially more often) than states corresponding to shorter sequences. Clearly, this breaks the objective of sampling terminating states in proportion to their reward and provides a strong motivation for considering GFlowNets instead.

Another perspective on maximizing entropy in RL is that one can also maximize entropy on the states' *stationary distribution* d^π (Ziebart et al., 2008), rather than the policy. In fact, one can show that the objective of training a policy such that $P_T(s) \propto R(s)$ is equivalent to training a policy that maximizes $r(s, a) = \log R(s, a) - \log d^\pi(s, a)$. Unfortunately, computing stationary distributions, although possible (Nachum et al., 2019; Wen et al., 2020), is not always tractable nor precise enough for purposes of reward regularization.

7.3 Contrast with Monte-Carlo Markov Chain methods

MCMC has a long and rich history (Metropolis et al., 1953; Hastings, 1970; Andrieu et al., 2003), and is particularly relevant to the present work, since it is also a principled class of methods towards sampling from $P_T(s) \propto R(s)$. MCMC-based methods have already found some amount of success with learned deep neural networks used to drive sampling (Grathwohl et al., 2021; Dai et al., 2020; Xie et al., 2021; Nash and Durkan, 2019; Seff et al., 2019).

An important drawback of MCMC is its reliance on iterative sampling (forming the Markov chain, one configuration at a time, each of which is like a terminating state of a GFlowNet): a new state configuration is obtained at each step of the chain by making a small stochastic change to the configuration in the previous step. Although these methods guarantee that asymptotically (in the length of the chain) we obtain samples drawn from the correct distribution, there is an important set of distributions for which finite chains are unlikely to provide enough diversity of the modes of the distribution.

This is known as the mode-mixing problem (Jasra et al., 2005; Bengio et al., 2013; Pompe et al., 2020): the chances of going from one mode to a neighboring one may become

exponentially small (and thus require exponentially long chains) if the modes are separated by a long sequence of low-probability configurations. This can be alleviated by burning more computation (sampling longer chains) but becomes exponentially unsustainable with increased mode separation. The issue can also be reduced by introducing random sampling (e.g., drawing multiple chains) and simulated annealing (Andrieu et al., 2003) to facilitate jumping between modes. However, this becomes less effective in high dimensions and when the modes occupy a tiny volume (which can become an exponentially small fraction of the total space as its dimension increases) since random sampling is unlikely to land in the neighborhood of a mode.

In contrast, GFlowNets belong to the family of **amortized** sampling methods (which includes VAEs, Kingma and Welling, 2013), where we train a machine learning system to produce samples: we have exchanged the complexity of sampling through long chains for the complexity of training the sampler. The potential advantage of such amortized samplers is when the distribution of interest has generalizable structure: when it is possible to guess reasonably well where high-probability samples can be found, based on the knowledge of a set of known high-probability samples (the training set). This is what makes deep generative models work in the first place and thus suggests that in such high-dimensional settings where modes occupy tiny volumes (as per the manifold hypothesis, Cayton, 2005; Narayanan and Mitter, 2010; Rifai et al., 2011), one can capitalize on the already observed $(x, R(x))$ pairs (where x is an already visited configuration and $R(x)$ its reward) to "jump" from known modes to yet unvisited ones, even if these are far from the ones already visited.

How well this will work then depends on the ability to generalize of the learner, i.e., on the strength and appropriateness of its inductive biases, as usual in machine learning. In the case where there is no structure at all (and thus no possibility to generalize when learning about the distribution), there is no reason to expect that amortized ML methods will fare better than MCMC. But if there is structure, then the exponential cost of mixing between modes could go away. There is plenty of evidence that ML methods can do a good job in such high-dimensional spaces (like the space of natural images) and this suggests that GFlowNets and other amortized sampling methods would be worth considering where ML generally works well. Molecular graph generation experiments (Bengio et al., 2021) comparing GFlowNets and MCMC methods appear to confirm this.

Another factor to consider (independent of the mode mixing issue) is the amortization of the computational costs: GFlowNets pay a large price upfront to train the network and then a small price (sampling once from P_T) to generate each new sample. Instead, MCMC has no upfront cost but pays a lot for each independent sample. Hence, if we want to only sample once, MCMC may be more efficient, whereas if we want to generate a lot of samples, amortized methods may be advantageous. One can imagine settings where GFlowNets and MCMC could be combined to achieve some of the advantages of both approaches.

Evolutionary Methods Evolutionary methods work similarly to MCMC methods, via an iterative process of stochastic local search, and populations of candidates are found that maximize one or many objectives (Brown et al., 2004; Salimans et al., 2017; Jensen, 2019; Swersky et al., 2020). From such a perspective, they have similar advantages and disadvantages. One practical advantage of these methods is that natural diversity is easily obtainable via group metrics and subpopulation selection (Mouret and Doncieux, 2012).

This is not something that is explicitly tackled by GFlowNet, which instead relies on i.i.d. sampling and giving non-zero probability to suboptimal samples as a diversity mechanism.

Sequential Monte-Carlo Sequential Monte Carlo (SMC, Naesseth et al., 2019; Arulampalam et al., 2002) methods are a class of methods aimed at solving inference problems. Similar to GFlowNets, SMC samplers are trained to sample from a distribution given by its unnormalized probability density, and require forward and backward kernels, as in GFlowNets. Unlike GFlowNets though, they require specifying intermediate targets $\gamma_t(z_t)$. In addition, with GFlowNets, the reward normally only comes at the end of the sequence, unlike the per-time-step likelihoods used to reweight particles in SMC. GFlowNets can be applied in settings which do not fit the typical particle filter setting, such as those whose intermediate states do not correspond to valid elements of the sample space. The trajectories do not necessarily represent a sequence of latent variables associated with corresponding observations, as in filtering tasks. The GFlowNet trajectory distributions are only defined by the terminal state reward function.

8. Conclusions and Open Questions

This paper extends and deepens the mathematical framework and mathematical properties of GFlowNets (Bengio et al., 2021). It connects the notion of flow in GFlowNets with that of measure over trajectories and introduces a novel training objective (the detailed balance loss) which makes it possible to choose a parametrization separating the backward policy P_B which controls preferences over the order in which things are done from the constraints imposed by the target reward function.

An important contribution of this paper is the mathematical framework for marginalization or free energy estimation using GFlowNets. It relies on the simple idea of conditioning the GFlowNet so as to push the ability to estimate a partition function already introduced by Bengio et al. (2021) to a much more general setting. This makes it possible in principle to estimate intractable sums of rewards over the terminating states reachable by an arbitrary state, opening the door to marginalization over supergraphs of graphs, supersets of sets, and supersets of (variable,value) pairs. In turn, this provides formulae for estimating entropies, conditional entropies and mutual information.

Many open questions obviously remain, from the extension to continuous actions and states to hierarchical versions of GFlowNets with abstract actions and integrating the energy function in the GFlowNet parametrization itself, enabling an interesting form of modularization and knowledge decomposition. Importantly, many of the mathematical formulations presented in this paper will require empirical validation to ascertain their usefulness, improve these ideas, turn them into impactful algorithms and explore a potentially very broad range of interesting applications, from replacing MCMC or being combined with MCMC in some settings, to probabilistic reasoning to further applications in active learning for scientific discovery.

References

C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1):5–43, 2003.

M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002. doi: 10.1109/78.978374.

P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 32(1):48–77, 2002.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ICLR'2015, arXiv:1409.0473*, 2014.

E. Bengio, M. Jain, M. Korablyov, D. Precup, and Y. Bengio. Flow network based generative models for non-iterative diverse candidate generation. *NeurIPS'2021, arXiv:2106.04399*, 2021.

Y. Bengio, G. Mesnil, Y. Dauphin, and S. Rifai. Better mixing via deep representations. In *International conference on machine learning*, pages 552–560. PMLR, 2013.

N. Brown, B. McKay, F. Gilardoni, and J. Gasteiger. A graph-based genetic algorithm and its application to the multiobjective evolution of median molecules. *Journal of chemical information and computer sciences*, 44(3):1079–1087, 2004.

L. Buesing, N. Heess, and T. Weber. Approximate inference in discrete distributions with monte carlo tree search and value functions, 2019.

L. Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 12(1-17):1, 2005.

H. Dai, R. Singh, B. Dai, C. Sutton, and D. Schuurmans. Learning discrete energy-based models via auxiliary-variable local exploration. In *Neural Information Processing Systems (NeurIPS)*, 2020.

T. Deleu, A. Góis, C. Emezue, M. Rankawat, S. Lacoste-Julien, S. Bauer, and Y. Bengio. Bayesian structure learning with generative flow networks. In *Uncertainty in Artificial Intelligence*, pages 518–528. PMLR, 2022.

L. Dinh, D. Krueger, and Y. Bengio. Nice: Non-linear independent components estimation. *ICLR'2015 Workshop, arXiv:1410.8516*, 2014.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real NVP. *ICLR'2017, arXiv:1605.08803*, 2016.

D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

A. Goyal and Y. Bengio. Inductive biases for deep learning of higher-level cognition. *arXiv*, abs/2011.15091, 2020. https://arxiv.org/abs/2011.15091.

A. Goyal, A. Lamb, J. Hoffmann, S. Sodhani, S. Levine, Y. Bengio, and B. Schölkopf. Recurrent independent mechanisms. *ICLR'2021, arXiv:1909.10893*, 2019.

W. Grathwohl, K. Swersky, M. Hashemi, D. Duvenaud, and C. J. Maddison. Oops i took a gradient: Scalable sampling for discrete distributions, 2021.

R.-R. Griffiths and J. M. Hernández-Lobato. Constrained bayesian optimization for automatic chemical design. *arXiv preprint arXiv:1709.05501*, 2017.

T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361. PMLR, 2017.

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 1970.

E. Hu, N. Malkin, M. Jain, K. Everett, A. Graikos, and Y. Bengio. Gflownet-em for learning compositional latent variable models. *arvix*, 2023.

M. Jain, E. Bengio, A. Hernandez-Garcia, J. Rector-Brooks, B. F. P. Dossou, C. Ekbote, J. Fu, T. Zhang, M. Kilgour, D. Zhang, L. Simine, P. Das, and Y. Bengio. Biological sequence design with gflownets. *International Conference on Machine Learning (ICML)*, 2022.

M. Jain, S. C. Raparthy, A. Hernandez-Garcia, J. Rector-Brooks, Y. Bengio, S. Miret, and E. Bengio. Multi-objective gflownets. *arXiv preprint arXiv:2210.12765*, 2023.

A. Jasra, C. C. Holmes, and D. A. Stephens. Markov chain monte carlo methods and the label switching problem in bayesian mixture modeling. *Statistical Science*, pages 50–67, 2005.

J. H. Jensen. A graph-based genetic algorithm and generative model/monte carlo tree search for the exploration of chemical space. *Chemical science*, 10(12):3567–3572, 2019.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

R. Kumar, S. Ozair, A. Goyal, A. Courville, and Y. Bengio. Maximum entropy generators for energy-based models, 2019.

M. J. Kusner, B. Paige, and J. M. Hernández-Lobato. Grammar variational autoencoder. In *International Conference on Machine Learning*, pages 1945–1954. PMLR, 2017.

S. Lahlou, T. Deleu, P. Lemos, D. Zhang, A. Volokhova, A. Hernández-García, L. N. Ezzine, Y. Bengio, and N. Malkin. A theory of continuous generative flow networks. *International Conference on Machine Learning (ICML)*, 2023.

S. Lange, T. Gabel, and M. Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

S. Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

S. A. Malik, S. Lahlou, A. Jesson, M. Jain, N. Malkin, T. Deleu, Y. Bengio, and Y. Gal. Batchgfn: Generative flow networks for batch active learning. *arXiv preprint arXiv: 2306.15058*, 2023.

N. Malkin, M. Jain, E. Bengio, C. Sun, and Y. Bengio. Trajectory balance: Improved credit assignment in gflownets. *arXiv preprint arXiv:2201.13259*, 2022.

N. Malkin, S. Lahlou, T. Deleu, X. Ji, E. Hu, K. Everett, D. Zhang, and Y. Bengio. GFlowNets and variational inference. *International Conference on Learning Representations (ICLR)*, 2023.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6): 1087–1092, 1953.

J. Močkus. On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer, 1975.

J.-B. Mouret and S. Doncieux. Encouraging Behavioral Diversity in Evolutionary Robotics: An Empirical Study. *Evolutionary Computation*, 20(1):91–133, 03 2012. ISSN 1063-6560. doi: 10.1162/EVCO_a_00048. URL https://doi.org/10.1162/EVCO_a_00048.

O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. Bridging the gap between value and policy based reinforcement learning. *arXiv preprint arXiv:1702.08892*, 2017.

O. Nachum, Y. Chow, B. Dai, and L. Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. *arXiv preprint arXiv:1906.04733*, 2019.

C. A. Naesseth, F. Lindsten, T. B. Schön, et al. Elements of sequential monte carlo. *Foundations and Trends® in Machine Learning*, 12(3):307–392, 2019.

H. Narayanan and S. Mitter. Sample complexity of testing the manifold hypothesis. In *NIPS'2010*, pages 1786–1794, 2010.

C. Nash and C. Durkan. Autoregressive energy machines. In *International Conference on Machine Learning*, pages 1735–1744. PMLR, 2019.

L. Pan, N. Malkin, D. Zhang, and Y. Bengio. Better training of gflownets with local credit and incomplete trajectories. *arXiv preprint arXiv: 2302.01687*, 2023.

E. Pompe, C. Holmes, and K. Łatuszyński. A framework for adaptive mcmc targeting multimodal distributions. *The Annals of Statistics*, 48(5):2930–2952, 2020.

D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

M. Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European conference on machine learning*, pages 317–328. Springer, 2005.

S. Rifai, Y. N. Dauphin, P. Vincent, Y. Bengio, and X. Muller. The manifold tangent classifier. *Advances in neural information processing systems*, 24:2294–2302, 2011.

T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

B. Schölkopf, D. Janzing, J. Peters, E. Sgouritsa, K. Zhang, and J. Mooij. On causal and anticausal learning. In *ICML'2012*, pages 1255–1262, 2012.

A. Seff, W. Zhou, F. Damani, A. Doyle, and R. P. Adams. Discrete object generation with reversible inductive construction. *arXiv preprint arXiv:1907.08268*, 2019.

J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning (ICML)*, 2010.

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

K. Swersky, Y. Rubanova, D. Dohan, and K. Murphy. Amortized bayesian optimization over discrete spaces. In *Conference on Uncertainty in Artificial Intelligence*, pages 769–778. PMLR, 2020.

M. Toussaint and A. Storkey. Probabilistic inference for solving discrete and continuous state markov decision processes. In *Proceedings of the 23rd international conference on Machine learning*, pages 945–952, 2006.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

J. Wen, B. Dai, L. Li, and D. Schuurmans. Batch stationary distribution estimation. *arXiv preprint arXiv:2003.00722*, 2020.

Y. Xie, C. Shi, H. Zhou, Y. Yang, W. Zhang, Y. Yu, and L. Li. {MARS}: Markov molecular sampling for multi-objective drug discovery. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=kHSu4ebxFXY`.

D. Zhang, N. Malkin, Z. Liu, A. Volokhova, A. Courville, and Y. Bengio. Generative flow networks for discrete probabilistic modeling. *International Conference on Machine Learning (ICML)*, 2022.

D. W. Zhang, C. Rainone, M. Peschl, and R. Bondesan. Robust scheduling with gflownets. *International Conference on Learning Representations (ICLR)*, 2023.

B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

H. Zimmermann, F. Lindsten, J.-W. van de Meent, and C. A. Naesseth. A variational perspective on generative flow networks. *arXiv preprint 2210.07992*, 2022.