

# GFlowNet Foundations

Yoshua Bengio<sup>1,2,5</sup>, Tristan Deleu<sup>1,2</sup>, Edward Hu<sup>6,1</sup>,  
Salem Lahlou<sup>1,2</sup>, Mo Tiwari<sup>4</sup>, and Emmanuel Bengio<sup>1,3</sup>

<sup>1</sup>Mila

<sup>2</sup>University of Montreal

<sup>3</sup>McGill University

<sup>4</sup>Stanford University

<sup>5</sup>CIFAR, IVADO

<sup>6</sup>Microsoft Azure AI

September 2021

## 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. They 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, sampling from a Pareto frontier, connections to reward-maximizing policies, and extensions to stochastic environments, continuous actions and modular energy functions.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Measures over Markovian Flows</b>	<b>5</b>
2.1	Trajectories and Flows . . . . .	5
2.2	Probability Measure over Flows . . . . .	7
2.3	Markovian Flows . . . . .	9
2.4	Flow Matching Conditions . . . . .	11
<b>3</b>	<b>GFlowNets: Learning a Flow</b>	<b>18</b>
3.1	Introducing Time Stamps to Allow Cycles . . . . .	18
3.2	Estimating Transition Probabilities from Terminal Flows . . . . .	19
3.3	GFlowNets as an Alternative to MCMC Sampling . . . . .	20
3.4	Flow Matching and Detailed Balance Losses . . . . .	20
3.5	Stochastic Rewards . . . . .	24
3.6	GFlowNets can be Trained Offline . . . . .	24
3.7	Direct Credit Assignment in GFlowNets . . . . .	24
3.8	Exploiting Data as Known Terminating States . . . . .	30
<b>4</b>	<b>Conditional Flows and Free Energies</b>	<b>31</b>
4.1	Conditioning a GFlowNet . . . . .	32
4.2	Estimating Free Energies . . . . .	35
4.3	Training Energy-Based Models with a GFlowNet . . . . .	38
4.4	Active Learning with a GFlowNet . . . . .	39
4.5	Estimating Entropies, Conditional Entropies and Mutual Information . . . . .	40
<b>5</b>	<b>Policies in Deterministic and Stochastic Environments</b>	<b>42</b>
5.1	Known Deterministic Environments . . . . .	43
5.2	Backwards Transitions can be Chosen Freely . . . . .	44
5.3	Unknown Deterministic Environments . . . . .	45
5.4	Stochastic Environments . . . . .	45
<b>6</b>	<b>Expected Reward and Reward-Maximizing Policy</b>	<b>47</b>
6.1	Preference for High-Reward Early Trajectory . . . . .	49
<b>7</b>	<b>Intermediate Rewards and Trajectory Returns</b>	<b>49</b>
<b>8</b>	<b>Multi-Flows, Distributional GFlowNets, Unsupervised GFlowNets and Pareto GFlowNets</b>	<b>51</b>
8.1	Defining a reward function a posteriori . . . . .	53
8.2	Pareto GFlowNets . . . . .	54

<b>9</b>	<b>GFlowNets on Sets, Graphs, and to Marginalize Joint Distributions</b>	<b>55</b>
9.1	Set GFlowNets . . . . .	55
9.2	GFlowNet on Graphs . . . . .	57
9.3	Marginalizing over Missing Variables . . . . .	57
9.4	Modular Energy Function Decomposition . . . . .	58
<b>10</b>	<b>Continuous or Hybrid Actions and States</b>	<b>59</b>
10.1	Integrable Normalization Constants . . . . .	59
10.2	GFlowNets in GFlowNets . . . . .	60
<b>11</b>	<b>Related Work</b>	<b>61</b>
11.1	Contrast with Generative Models . . . . .	61
11.2	Contrast with Regularized Reinforcement Learning . . . . .	62
11.3	Contrast with Monte-Carlo Markov Chain methods . . . . .	64
<b>12</b>	<b>Conclusions and Open Questions</b>	<b>65</b>

# 1 Introduction

This paper builds upon the Generative Flow Networks (GFlowNets) introduced by Bengio et al. (2021), providing 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. GFlowNets have properties which make them well-suited to model and sample from distributions over sets and graphs, estimate free energies and marginal distributions in general, and be used to learn an energy function from data as a learned, amortized alternative to Monte-Carlo Markov chains (MCMC).

The key property of GFlowNets is that they learn a policy which samples composite objects  $s$  through several steps such that the probability  $P_T(s)$  of sampling an object  $s$  is approximately proportional to the value  $R(s)$  of a given reward function applied to that object. Whereas one typically trains a generative model from a dataset of positive examples, a GFlowNet is trained to match a given energy function and convert it into a sampler, which we view as a generative policy because the composite object  $s$  is constructed through a sequence of steps. This is similar to what MCMC methods achieve but does not require a lengthy stochastic search in the space of such objects and avoids the mode-mixing intractability challenge of MCMC methods (Jasra et al., 2005; Bengio et al., 2013; Pompe et al., 2020). GFlowNets exchange that intractability 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 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.

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<sup>1</sup> 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 be generalized to estimate multiple flows corresponding to modeling a rich outcome (rather than a scalar reward function), similarly to distributional reinforcement learning (Bellemare et al., 2017).

We refer the reader to Bengio et al. (2021) and Sec. 11 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 (i) can be trained in an offline manner with trajectories sampled from a distribution different from the one represented by the GFlowNet and (ii) 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

---

<sup>1</sup>In machine learning, a free energy is the logarithm of an unnormalized marginal probability, a generally intractable sum of exponentiated negative energies.

both interesting and diverse.

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 formulations enabling the calculation of marginal probabilities (or free energies) for subsets of variables, more generally for subsets of larger sets, or subgraphs, their application to estimating entropy and mutual information, and the introduction of an unsupervised form of GFlowNet (the reward function is not needed while training, only observations of outcomes) enabling sampling from a Pareto frontier, for example. Although basic GFlowNets are more similar to bandits (in that a reward is only provided at the end of a sequence of actions), they can be extended to take into account intermediate rewards and thus a notion of return, and sample according to these returns. The original formulation of GFlowNet is also limited to discrete and deterministic environments, while this paper suggests how these two limitations could be lifted. 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.

## 2 Measures over Markovian Flows

### 2.1 Trajectories and Flows

**Definition 1.** A *directed graph* is a tuple  $(\mathcal{S}, \mathbb{A})$ , where  $\mathcal{S}$  is a set of states, and  $\mathbb{A}$  a subset of  $\mathcal{S} \times \mathcal{S}$  representing directed edges. Elements of  $\mathbb{A}$  are denoted  $s \rightarrow s'$  and called **edges** or **transitions**.

A **trajectory** in such a graph is a sequence  $\tau = (s_1, \dots, s_n)$  of elements of  $\mathcal{S}$  such that every transition  $s_t \rightarrow s_{t+1} \in \mathbb{A}$ . We denote  $s \in \tau$  to mean that  $s$  is in the trajectory  $\tau$ , i.e.,  $\exists t \in \{1, \dots, n\} \ s_t = s$ , and similarly  $s \rightarrow s' \in \tau$  to mean that  $\exists t \in \{1, \dots, n\} \ s_t = s, s_{t+1} = s'$ .

A **directed acyclic graph** (DAG) is a directed graph in which there is no trajectory  $\tau = (s_1, \dots, s_n)$  satisfying  $s_n = s_1$ , besides trajectories composed of one state only.

**Definition 2.** Given a DAG, we define a **partial order**, denoted by  $<$ , such

that for any pair of states  $s, s' \in \mathcal{S}$ ,  $s < s'$  if there exists a trajectory in the DAG starting in  $s$  and ending in  $s'$ . If there is no order relation between  $s$  and  $s'$ , we write  $s \leq s'$ .

We only allow at most one edge to directly connect two states. We introduce the notion of odd and even states in Sec. 5 to deal with cases where the same transition can be triggered by different “actions.”

**Definition 3.** Hereinafter, we consider only DAGs in which we can define two special states: the **source state** or **initial state**  $s_0$  and the **sink state** or **final state**  $s_f$ , with  $\forall s, s_0 < s$  and  $\forall s, s < s_f$ . We define a **complete trajectory**  $\tau$  as starting in  $s_0$  and ending in  $s_f$ :  $\tau = (s_0, s_1, \dots, s_f)$ . We denote by  $\mathcal{T}$  the **set of all complete trajectories** associated with such a given DAG.

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.** We call a transition  $s \rightarrow s_f$  into the final state a **terminating transition** and  $F(s \rightarrow s_f)$  a **terminating flow**. The flow through an edge  $s \rightarrow s'$  is called an **edge flow**. In a trajectory  $(s_0, s_1, \dots, s_n, s_f)$ ,  $s_n$  is called the **terminating state** of the trajectory.

**Definition 5.** We define a **trajectory flow**  $F : \mathcal{T} \mapsto \mathbb{R}^+$  as any nonnegative function defined on the set of complete trajectories  $\mathcal{T}$ .

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 trajectory  $\tau$  contains the number of particles sharing the same path  $\tau$ .

**Definition 6.** The **flow through a state** (or **state flow**)  $F : \mathcal{S} \mapsto \mathbb{R}$  is the sum of the flows of the complete trajectories passing through that state:

$$F(s) := \sum_{\tau \in \mathcal{T}} 1_{s \in \tau} F(\tau) \quad (1)$$

More generally, we can define any constraint on the complete trajectories, for example going through two specific states, or going through a specific transition:

**Definition 7.** Given a trajectory flow  $F : 2^{\mathcal{T}} \mapsto \mathbb{R}$ , we define the flow  $F$  as a measure on  $\mathcal{T}$ , defined for every subset  $A \subseteq \mathcal{T}$  as the sum of the flows of the complete trajectories within some event, i.e., a set  $A \subseteq \mathcal{T}$  of trajectories compatible with that event:

$$F(A) := \sum_{\tau \in A} F(\tau). \quad (2)$$

We abuse notation for the argument  $A$  of  $F$  in ways that should be clear, as follows. A special kind of event is the singleton trajectory  $\tau$  and we just write its flow or measure  $F(\tau)$ . Another special type of event is the set of trajectories containing a particular state (with flow denoted  $F(s)$ ) or a particular transition (with edge flow denoted  $F(s \rightarrow s')$ ). Note that we define  $F(s \rightarrow s') = 0$  if it is not the case that  $s \rightarrow s' \in \mathbb{A}$ .

**Definition 8.** *The **flow over a joint event**  $A$  and  $B$  is the flow associated with the intersection of these events, denoted*

$$F(A \cap B) := \sum_{\tau \in A \cap B} F(\tau) \quad (3)$$

*When the events are adjacent transitions, such as  $F(s \rightarrow s' \cap s' \rightarrow s'')$ , we abbreviate it to  $F(s \rightarrow s' \rightarrow s'')$ .*

## 2.2 Probability Measure over Flows

**Definition 9.** *The **total flow**  $Z$  is the sum of the flows of all the complete trajectories:*

$$Z := \sum_{\tau \in \mathcal{T}} F(\tau). \quad (4)$$

We use the letter  $Z$ , often used to denote the partition function in probabilistic models and statistical mechanics, because it is a normalizing constant which can turn a measure  $F$  over sets of trajectories into a probability measure  $P$  defined below:

**Proposition 1.** *The flow through the initial state equals the flow through the final state equals the total flow  $Z$ .*

*Proof.* Since  $\forall \tau \in \mathcal{T}$ ,  $s_0 \in \tau$ , applying Eq. 1 to  $s_0$  yields

$$F(s_0) = \sum_{\tau \in \mathcal{T}} F(\tau) = Z. \quad (5)$$

Similarly, since  $\forall \tau \in \mathcal{T}$ ,  $s_f \in \tau$ , we obtain

$$F(s_f) = \sum_{\tau \in \mathcal{T}} F(\tau) = Z \quad (6)$$

and combining both statements we obtain the result.  $\square$

Intuitively, Prop. 1 justifies our use of the term "flow" by analogy with a stream of particles flowing from the initial state to the final states.

**Definition 10.** *We can associate with the trajectory flow  $F$  a **probability measure**  $P$  over the mutually exclusive outcomes that are the complete trajectories  $\tau \in \mathcal{T}$ , via*

$$P(\tau) := \frac{F(\tau)}{\sum_{\tau \in \mathcal{T}} F(\tau)} = \frac{F(\tau)}{Z}. \quad (7)$$

In the same way that we have extended the semantics of the flow function  $F$  to accept any event (implicitly or explicitly a set of trajectories) as argument, we can extend the semantics of the probability function  $P$  to be in line with the general usage in probability theory.

**Definition 11.** *For any event  $A \subseteq \mathcal{T}$ , we define the probability of that event as*

$$P(A) := \frac{F(A)}{Z}. \quad (8)$$

Intuitively,  $P$  defines a (normalized) probability measure over trajectories such that the probability of any trajectory  $\tau$  is proportional to  $F(\tau)$ .

**Proposition 2.** *The probability of a trajectory passing through initial state  $s_0$  is 1:*

$$P(s_0) = 1. \quad (9)$$

*Proof.* First apply Eq. 8, then Eq. 5 and cancel  $Z$ .  $\square$

From the above probability measure, one can define any conditional probability over flow events, as usual:

**Definition 12.** *The **conditional probability of an event  $A$  given another event  $B$**  is*

$$P(A|B) := \frac{P(A, B)}{P(B)} \quad (10)$$

$$= \frac{F(A \cap B)}{F(B)}. \quad (11)$$

The second line comes from combining Def. 8 and Def. 11.

**Definition 13.** *The **transition probability**  $P(s_t \rightarrow s_{t+1} | s_t)$  is a special case of conditional probability, denoted  $P_F(s_{t+1} | s_t)$  and defined as*

$$P_F(s' | s) := P(s \rightarrow s' | s) = \frac{F(s \rightarrow s')}{F(s)}. \quad (12)$$

*Similarly, we define the **backwards transition probability** and introduce the notation*

$$P_B(s | s') := P(s \rightarrow s' | s') = \frac{F(s \rightarrow s')}{F(s')}. \quad (13)$$

We use the explicit subscript  $_B$  to avoid ambiguity when the arguments of  $P_B$  are not indexed by their trajectory position.

**Notation:** In this paper, when we write events (regarding the measure  $F$  or its associated probability measure  $P$ ) involving states indexed by  $t$  as in  $s_t, s_{t+1}, s_{t+2}, \dots$ , we mean that we consider only trajectories  $\tau$  where those states are consecutive in a trajectory, according to the order prescribed by the integer temporal index  $t$ .



### 2.3 Markovian Flows

**Definition 14.** A flow with probability measure  $P$  is called a **Markovian flow** (or simply **Markovian**) if, for any state  $s$ , outgoing edge  $s \rightarrow s'$ , and for any trajectory  $\tau = (s_1, \dots, s)$  ending in  $s$ :

$$P(s \rightarrow s' | \tau) = P(s \rightarrow s' | s) = P_F(s' | s). \quad (14)$$

Note that the Markovian property does not hold for all of the flows as defined in the previous sections. 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.

**Proposition 3.** For Markovian flows and  $\tau = (s_0, s_1, \dots, s_n)$  a complete trajectory (i.e.,  $s_n = s_f$ ), we obtain that

$$P(\tau) = \prod_{t=1}^n P_F(s_t | s_{t-1}) \quad (15)$$

and

$$F(\tau) = Z \prod_{t=1}^n P_F(s_t | s_{t-1}) = \frac{\prod_{t=1}^n F(s_{t-1} \rightarrow s_t)}{\prod_{t=1}^{n-1} F(s_t)} \quad (16)$$

as well as

$$F(\tau) = Z \prod_{t=1}^n P_B(s_{t-1} | s_t) \quad (17)$$

Therefore

$$P(\tau) = \prod_{t=1}^n P_B(s_{t-1} | s_t) \quad (18)$$

*Proof.* Eq. 15 is obtained from the usual laws of probability, the Markovian conditional independence and Eq. 9.

$$\begin{aligned}
P(\tau) &= P(s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_n) \\
&= P(s_0 \rightarrow s_1) \prod_{t=1}^{n-1} P(s_t \rightarrow s_{t+1} | s_0 \rightarrow \dots \rightarrow s_t) \\
&= P(s_0 \rightarrow s_1) \prod_{t=1}^{n-1} P(s_t \rightarrow s_{t+1} | s_t) \\
&= P(s_0) P_F(s_1 | s_0) \prod_{t=1}^{n-1} P_F(s_{t+1} | s_t) \\
&= \prod_{t=1}^n P_F(s_t | s_{t-1}).
\end{aligned}$$

Eq. 16 is obtained from the above by plugging the definition of conditional probability in terms of flow ratios (Eq. 11):

$$\begin{aligned}
F(\tau) &= Z P(\tau) \\
&= F(s_0) \prod_{t=1}^n P_F(s_t | s_{t-1}) \\
&= F(s_0) \prod_{t=1}^n \frac{F(s_{t-1} \rightarrow s_t)}{F(s_{t-1})} \\
&= \frac{\prod_{t=1}^n F(s_{t-1} \rightarrow s_t)}{\prod_{t=1}^{n-1} F(s_t)}.
\end{aligned}$$

The third equation is obtained by rewriting the last one and using the definition of the backwards transition probability:

$$\begin{aligned}
F(\tau) &= \frac{\prod_{t=1}^n F(s_{t-1} \rightarrow s_t)}{\prod_{t=1}^{n-1} F(s_t)} \\
&= F(s_f) \prod_{t=1}^n \frac{F(s_{t-1} \rightarrow s_t)}{F(s_t)} \\
&= Z \prod_{t=1}^n P_B(s_{t-1} | s_t)
\end{aligned}$$

The fourth equation follows directly by dividing both sides by  $Z$ . □

**Corollary 1.** *A Markovian flow is completely and uniquely specified by the combination of the total flow and the forward transition probabilities  $P_F(s_t | s_{t-1})$  or, alternatively, by the combination of the terminating flows  $F(s \rightarrow s_f)$  and the backwards transition probabilities  $P_B(s_{t-1} | s_t)$  for all  $s_t < s_f$  or, alternatively, by the total flow and all the backwards transition probabilities.*

*Proof.* This is a direct consequence of Prop. 3, as shown below. First, the specification of  $F(\tau)$ , and thus the measure and probability or conditional probability of every event on the flow is fully specified by the total flow and the forward transition probabilities, using Eq. 15:

$$F(\tau) = ZP(\tau) = Z \prod_{t=1}^n P_F(s_t | s_{t-1}) = F(s_0) \prod_{t=1}^n P_F(s_t | s_{t-1}).$$

Second, the specification of  $F(\tau)$  is alternatively fully specified by defining the terminating flows  $F(s_{n-1} \rightarrow s_f)$  and the backwards transition probabilities  $P_B(s_{t-1} | s_t)$ , as shown by Eq. 18. Note how in both cases we can independently choose each of the two factors. Note also that specifying all the terminal flows  $F(s \rightarrow s_f)$  is equivalent to specifying the total flow  $F(s_f)$  and the terminal backwards transition probabilities  $P_B(s | s_f) = P(s \rightarrow s_f | s_f)$ .  $\square$

**Corollary 2.** *Consider a flow specified by a forward transition probability function  $P_F(s_{t+1} | s_t)$  and a total flow  $Z$ , such that the flow of any (possibly incomplete) trajectory  $\tau = (s_1, \dots, s_n)$  satisfies  $F(\tau) = F(s_1) \prod_{t=1}^{n-1} P_F(s_{t+1} | s_t)$  (i.e., trajectories can be drawn by drawing their initial state  $s_1$  and then iteratively sampling successive states according to  $P_F(s_{t+1} | s_t)$ ), then the flow is Markovian.*

*Proof.* To show that the flow is Markovian, we consider a state  $s$  and an outgoing edge  $s \rightarrow s'$ , along with a trajectory  $\tau = (s_1, \dots, s_n)$  ending in  $s$  (i.e.,  $s_n = s$ ). We need to prove that Eq. 14 holds.

Let  $\tau' = (s_1, \dots, s_n, s_{n+1})$ , with  $s_{n+1} = s'$ . By definition of conditional probability, we have:

$$\begin{aligned} P(\tau') &= P(s_1 \rightarrow \dots \rightarrow s_n, s_n \rightarrow s_{n+1}) \\ &= P(s_n \rightarrow s_{n+1} | \tau) P(\tau) = P(s \rightarrow s' | \tau) P(\tau) \end{aligned}$$

Which means that

$$\frac{F(s_1)}{Z} \prod_{t=1}^n P_F(s_{t+1} | s_t) = \frac{F(s_1)}{Z} P(s \rightarrow s' | \tau) \prod_{t=1}^{n-1} P_F(s_{t+1} | s_t)$$

Cancelling out the  $n - 1$  terms of the product, along with  $\frac{F(s_1)}{Z}$ , leads to  $P(s \rightarrow s' | \tau) = P_F(s' | s)$ .  $\square$

## 2.4 Flow Matching Conditions

**Definition 15.** *The **parent set** of a state  $s$ , which we denote  $Par(s)$ , contains all of the direct parents of  $s$  in the flow DAG, i.e.,  $Par(s) = \{s' \in \mathcal{S} : s' \rightarrow s \in \mathbb{A}\}$ ; similarly, the **child set**  $Child(s)$  contains all of the direct children of  $s$  in the flow DAG, i.e.,  $Child(s) = \{s' \in \mathcal{S} : s \rightarrow s' \in \mathbb{A}\}$ .*

**Proposition 4.** Consider a non-negative function  $\hat{F}$  taking as input either a state  $s$  or a transition  $s \rightarrow s'$ . Let  $\hat{F}$  define the associated forward transition probabilities estimator

$$\hat{P}_F(s_{t+1}|s_t) = \hat{P}(s_t \rightarrow s_{t+1}|s_t) := \frac{\hat{F}(s_t \rightarrow s_{t+1})}{\hat{F}(s_t)} \quad (19)$$

and backwards transition probabilities estimator

$$\hat{P}_B(s_t|s_{t+1}) = \hat{P}(s_t \rightarrow s_{t+1}|s_{t+1}) := \frac{\hat{F}(s_t \rightarrow s_{t+1})}{\hat{F}(s_{t+1})}. \quad (20)$$

Then  $\hat{F}$  corresponds to a flow if and only if the incoming and outgoing flows of each non-source and non-sink state are matched, i.e.,

$$\begin{aligned} \forall s' > s_0, \quad \hat{F}(s') &= \sum_{s \in \text{Par}(s')} \hat{F}(s \rightarrow s') \\ \forall s' < s_f, \quad \hat{F}(s') &= \sum_{s'' \in \text{Child}(s')} \hat{F}(s' \rightarrow s'') \end{aligned} \quad (21)$$

More specifically,  $\hat{F}$  uniquely defines a Markovian flow  $F$  matching  $\hat{F}$  on states and transitions and such that

$$F(\tau) = \frac{\prod_{t=1}^n \hat{F}(s_{t-1} \rightarrow s_t)}{\prod_{t=1}^{n-1} \hat{F}(s_t)} \quad (22)$$

more generally. Furthermore, if the flow-matching constraints are satisfied then the transition probability estimators are correctly normalized (summing to 1).

*Proof.* We first show necessity, i.e., we show every flow  $F$  satisfies the above flow-matching condition. First note that for a given state  $s$ , the transitions into it are mutually exclusive events, and so are the transitions out of it, i.e.,

$$\sum_{s \in \text{Par}(s')} 1_{s \rightarrow s' \in \tau} = 1_{s' \in \tau}.$$

Therefore

$$F(s') = \sum_{\tau} F(\tau) 1_{s' \in \tau} \quad (23)$$

$$= \sum_{s \in \text{Par}(s')} \sum_{\tau} F(\tau) 1_{s \rightarrow s' \in \tau} \quad (24)$$

$$= \sum_{s \in \text{Par}(s')} F(s \rightarrow s'). \quad (25)$$

Exactly the same approach can be used to prove the second equality of Eq. 21. Up to now we have only shown the condition in Eq. 21 is necessary to obtain a

flow.

To show sufficiency, we start with an  $\hat{F}$  which satisfies the flow matching conditions and show that it uniquely defines a Markovian flow  $F$ , as per Def. 7, i.e., that  $\hat{F}(s)$  and  $\hat{F}(s \rightarrow s')$  match  $F(s)$  and  $F(s \rightarrow s')$  respectively, and is defined on trajectories by Eq. 22 (as suggested by Eq. 16) for any complete trajectory  $\tau = (s_0, \dots, s_n)$ , with  $s_n = s_f$ .

1— We extend the trajectory flow function  $F$  to incomplete trajectories of at least 2 states, either starting from  $s_0$  or ending in  $s_n = s_f$  as:

$$\begin{aligned}\tilde{F}(s_0 \rightarrow \dots \rightarrow s_T) &= \frac{\prod_{t=1}^T \hat{F}(s_{t-1} \rightarrow s_t)}{\prod_{t=1}^{T-1} \hat{F}(s_t)} \\ \tilde{F}(s_T \rightarrow \dots \rightarrow s_n) &= \frac{\prod_{t=T+1}^n \hat{F}(s_{t-1} \rightarrow s_t)}{\prod_{t=T+1}^{n-1} \hat{F}(s_t)}\end{aligned}$$

It is straightforward that  $\tilde{F}$  coincides with flow  $F$  on the set of complete trajectories  $\mathcal{T}$ .

2— For the given DAG, we define the notion of maximum depth of a state  $s > s_0$  as the longest length of a trajectory (i.e., number of states composing it) from  $s_0$  to  $s$ . Naturally, the lowest possible value for this maximum depth is 2, and there exists at least one state for which the maximum depth is indeed equal to 2. By strong induction on the maximum depth, we will show that  $\forall s > s_0 \quad \sum_{\tau \text{ ending in } s} \tilde{F}(\tau) = \hat{F}(s)$ , where  $\{\tau \text{ ending in } s\}$  denotes the set of (possibly incomplete) trajectories starting in  $s_0$  and ending in  $s$ .

**Base case:** We need to prove the property for states of maximum depth equal to 2 (the lowest possible value):

Let  $s$  be such a state. By definition of the maximum depth, this means that  $s$  has one incoming edge only:  $s_0 \rightarrow s$ , and there is only one trajectory ending in  $s$ , that is  $(s_0, s)$ . Hence  $\sum_{\tau \text{ ending in } s} \tilde{F}(\tau) = \tilde{F}(s_0 \rightarrow s) = \hat{F}(s_0 \rightarrow s) = \hat{F}(s)$ , where the second equality stems from the definition of  $\tilde{F}$ , and the third follows from Eq. 21.

**Induction step:** Let  $d \geq 2$ , and assume that the property  $\sum_{\tau \text{ ending in } s} \tilde{F}(\tau) = \hat{F}(s)$  holds for every state  $s$  of any maximum depth in  $\{2, \dots, d\}$ . We need to prove the property for states of maximum depth  $d + 1$ :

Let  $s'$  be such a state. We can write the following:

$$\sum_{\tau \text{ ending in } s'} \tilde{F}(\tau) = \sum_{s \in \text{Par}(s')} \sum_{\tilde{\tau} \text{ ending in } s} \tilde{F}(\tilde{\tau}) \frac{\hat{F}(s \rightarrow s')}{\hat{F}(s)}$$

Because the maximum depth of  $s'$  is  $d + 1$ , then the maximum depth for any  $s \in \text{Par}(s')$  is at most  $d$ , which means that we can apply to each  $s \in \text{Par}(s')$  the induction step hypothesis, to obtain that  $\sum_{\tilde{\tau} \text{ ending in } s} \tilde{F}(\tilde{\tau}) = \hat{F}(s)$ . It

follows that  $\sum_{\tau \text{ ending in } s'} \tilde{F}(\tau) = \sum_{s \in \text{Par}(s')} \hat{F}(s \rightarrow s') = \hat{F}(s')$ , where the last equality follows from Eq. 21.

3– Using a similar notion of distance to the sink state, we can prove by strong induction again that  $\forall s < s_f \quad \sum_{\tau \text{ starting in } s} \tilde{F}(\tau) = \hat{F}(s)$ , where  $\{\tau \text{ starting in } s\}$  denotes the set of (possibly incomplete) trajectories starting in  $s$  and ending in  $s_f$ . The actual proof of this third step is omitted given that it requires the exact same reasoning as that of the second step.

4– We show that for any  $s \rightarrow s' \in \mathbb{A}$ , we indeed get  $\hat{F}(s \rightarrow s') = F(s \rightarrow s')$ . Assuming that  $s > s_0$  and  $s < s_f$ , we get:

$$\begin{aligned} F(s \rightarrow s') &= \sum_{\{\tau \in \mathcal{T}, s \rightarrow s' \in \tau\}} F(\tau) \\ &= \sum_{\{\tilde{\tau} \text{ ending in } s\}} \sum_{\{\tilde{\tau}' \text{ starting in } s'\}} \tilde{F}(\tilde{\tau}) \frac{\hat{F}(s \rightarrow s')}{\hat{F}(s)\hat{F}(s')} \tilde{F}(\tilde{\tau}') \\ &= \hat{F}(s \rightarrow s'), \end{aligned}$$

where the last equality is a direct application of the second and third steps above.

If  $s = s_0$  or  $s' = s_f$ , a similar reasoning can be applied to get that  $\hat{F}(s \rightarrow s') = F(s \rightarrow s')$ .

5– Finally, we show that for any  $s' \in \mathcal{S}$ , we indeed get  $\hat{F}(s') = F(s')$ . Assuming  $s' > s_0$ :

$$\begin{aligned} F(s) &= \sum_{s' \in \text{Child}(s)} F(s \rightarrow s') \\ &= \sum_{s' \in \text{Child}(s)} \hat{F}(s \rightarrow s') \\ &= \hat{F}(s), \end{aligned}$$

where the first equality is due to the necessity of the flow matching conditions as proved above (i.e.,  $F$  satisfies Eq. 31), the second equality comes from the fourth step above, and the last equality comes from the assumption that  $\hat{F}$  satisfies Eq. 31.

If  $s = s_0$ , then the second equality of Eq. 31 can be used to obtain the same result.

This finishes the proof of sufficiency.

Summing these over either  $s_t$  or  $s_{t+1}$  shows that the  $\hat{P}$  and  $\hat{P}_B$  conditional probabilities sum to 1.  $\square$

Note how Eq. 21 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, studied below, is when we are given all the terminal flows  $F(s \rightarrow s_f)$  and we would like to deduce a state flow function  $F(s)$  and a forward transition probability function  $P(s \rightarrow s')$  for the rest of the flow network:

**Proposition 5.** *Consider a non-negative function  $\hat{F}$  taking either a state  $s$  or a transition  $s \rightarrow s'$  as input and corresponding to a flow with probability measure  $\hat{P}$  and forward and backwards probabilities defined as in Eq. 19-Eq. 20. Then  $\hat{F}$  is Markovian if and only if, for every trajectory  $\tau = (s_{t_1}, \dots, s_{t_2})$ ,  $\hat{P}(\tau) = \hat{P}(s_{t_1}) \prod_{t=t_1}^{t_2-1} \hat{P}_F(s_{t+1}|s_t)$  (i.e., trajectories can be drawn by drawing  $s_{t_1}$  and iteratively sampling a next state  $s_{t+1}$  from the current state  $s_t$  using  $\hat{P}_F(s_{t+1}|s_t)$  for transition probabilities), or equivalently if  $\hat{P}(\tau) = \hat{P}(s_{t_2}) \prod_{t=t_1}^{t_2-1} \hat{P}_B(s_t|s_{t+1})$  (i.e., trajectories can be drawn by starting from  $s_{t_2}$  and iteratively sampling a previous state  $s_t$  from the current state  $s_{t+1}$  using  $\hat{P}_B(s_t|s_{t+1})$  at each step).*

*Proof.* Using the same proof as the one used for Eq. 15 in Prop. 3, it is straightforward that given a Markovian flow with probability measure  $P$ , then for any (possibly incomplete) trajectory  $\tau = (s_{t_1}, \dots, s_{t_2})$ ,  $\hat{P}(\tau) = \hat{P}(s_{t_1}) \prod_{t=t_1}^{t_2-1} \hat{P}_F(s_{t+1}|s_t)$ . Combining this property with Corollary 2 yields the first equivalence (with the forward probabilities).

To prove necessity of the second equivalence, we will use proof elements similar to those of Prop. 3. We assume the flow is Markovian:

$$\begin{aligned}
\hat{P}(\tau) &= \hat{P}(s_{t_1} \rightarrow s_{t_1+1} \rightarrow \dots \rightarrow s_{t_2}) \\
&= \hat{P}(s_{t_1} \rightarrow s_{t_1+1}) \prod_{t=t_1+1}^{t_2-1} \hat{P}(s_t \rightarrow s_{t+1} | s_{t_1} \rightarrow \dots \rightarrow s_t) \\
&= \hat{P}(s_{t_1} \rightarrow s_{t_1+1}) \prod_{t=t_1+1}^{t_2-1} \hat{P}_F(s_{t+1}|s_t) \quad \text{using Def. 14} \\
&= \hat{P}(s_{t_1}) \hat{P}_F(s_{t_1+1}|s_{t_1}) \prod_{t=t_1+1}^{t_2-1} \hat{P}_F(s_{t+1}|s_t) = \hat{P}(s_{t_1}) \prod_{t=t_1}^{t_2-1} \hat{P}_F(s_{t+1}|s_t) \\
&= \hat{P}(s_{t_1}) \prod_{t=t_1}^{t_2-1} \left( \hat{P}_B(s_t|s_{t+1}) \frac{\hat{F}(s_{t+1})}{\hat{F}(s_t)} \right) \quad \text{using Eq. 19-Eq. 20} \\
&= \hat{P}(s_{t_1}) \frac{\hat{F}(s_{t_2})}{\hat{F}(s_{t_1})} \prod_{t=t_1}^{t_2-1} \hat{P}_B(s_t|s_{t+1}) \\
&= \hat{P}(s_{t_2}) \prod_{t=t_1}^{t_2-1} \hat{P}_B(s_t|s_{t+1})
\end{aligned}$$

For sufficiency, then similarly to Corollary 2, let  $s \rightarrow s'$  be an edge, and  $\tau = (s_1, \dots, s_T)$  be a trajectory ending in  $s_T = s$ ; we need to prove that  $\hat{P}(s \rightarrow s' | \tau) = \hat{P}_F(s' | s)$ . For this reason, we consider  $\tau' = (s_1, \dots, s_T, s_{T+1})$ , with  $s_{T+1} = s'$ . We start by using the values of  $\hat{P}(\tau')$  and  $\hat{P}(\tau)$  in the decomposition  $\hat{P}(\tau') = \hat{P}(s \rightarrow s' | \tau) \hat{P}(\tau)$ :

$$\begin{aligned} \hat{P}(s_{T+1}) \prod_{t=1}^T \hat{P}_B(s_t | s_{t+1}) &= \hat{P}(s \rightarrow s' | \tau) \hat{P}(s_T) \prod_{t=1}^{T-1} \hat{P}_B(s_t | s_{t+1}) \\ \hat{P}(s_{T+1}) \hat{P}_B(s_T | s_{T+1}) &= \hat{P}(s_T) \hat{P}(s \rightarrow s' | \tau) \\ \frac{\hat{P}(s')}{\hat{P}(s)} \hat{P}_B(s | s') &= \hat{P}(s \rightarrow s' | \tau), \end{aligned}$$

which, according to Eq. 19-Eq. 20, leads to  $\hat{P}(s \rightarrow s' | \tau) = \hat{P}_F(s' | s)$ , which is the Markov condition.  $\square$

**Definition 16.** A forward transition probability function  $\hat{P}_F(s' | s)$  (i.e., a function mapping edges  $s \rightarrow s' \in \mathbb{A}$  to non-negative numbers  $\hat{P}_F(s' | s)$  with the condition  $\forall s \sum_{s' \in \text{Child}(s)} \hat{P}_F(s' | s) = 1$ ) and a backward transition probability function  $\hat{P}_B(s | s')$  (i.e., a function mapping edges  $s \rightarrow s' \in \mathbb{A}$  to non-negative numbers  $\hat{P}_B(s | s')$  with the condition  $\forall s' \sum_{s \in \text{Par}(s')} \hat{P}_B(s | s') = 1$ ) are **compatible** if there exists an edge flow function  $\hat{F} : \mathbb{A} \rightarrow [0, \infty)$  such that

$$\hat{P}_F(s' | s) = \frac{\hat{F}(s \rightarrow s')}{\sum_{s' \in \text{Child}(s)} \hat{F}(s \rightarrow s')} \quad (26)$$

$$\hat{P}_B(s | s') = \frac{\hat{F}(s \rightarrow s')}{\sum_{s \in \text{Par}(s')} \hat{F}(s \rightarrow s')}. \quad (27)$$

An alternative way to obtain the flow matching condition of Prop. 4 is as follows: unlike the condition in Prop. 4, it 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 17.** Consider a non-negative function  $\hat{F}$  over states, a forward transition probability function  $\hat{P}_F(s' | s)$  and a backwards transition probability function  $\hat{P}_B(s | s')$  over transitions  $s \rightarrow s'$ . The **detailed balance** condition over these functions is defined as follows:

$$\forall s \rightarrow s' \in \mathbb{A} \quad \hat{F}(s) \hat{P}_F(s' | s) = \hat{F}(s') \hat{P}_B(s | s'). \quad (28)$$

**Proposition 6.** Consider a non-negative function  $\hat{F}$  over states, a forward transition probability function  $\hat{P}_F(s' | s)$  and a backwards transition probability function  $\hat{P}_B(s | s')$  over transition  $s \rightarrow s'$ . Then,  $\hat{F}$ ,  $\hat{P}_B$ , and  $\hat{P}_F$  jointly correspond to a flow if and only if the detailed balance condition holds. Furthermore, when this condition is satisfied, the forward and backward transition probability functions  $\hat{P}_F$  and  $\hat{P}_B$  are compatible.



*Proof.* We prove the sufficiency of the condition by first defining the edge flow

$$\hat{F}(s \rightarrow s') := \hat{F}(s) \hat{P}_F(s'|s). \quad (29)$$

We then sum both sides of Eq. 28 over  $s$ , yielding

$$\sum_{s \in \text{Par}(s')} \hat{F}(s) \hat{P}_F(s'|s) = \hat{F}(s') \sum_{s \in \text{Par}(s')} \hat{P}_B(s|s') = \hat{F}(s') \quad (30)$$

where we used the fact that  $\hat{P}_B$  is a normalized probability distribution. Combining this with Eq. 29, we get

$$\hat{F}(s') = \sum_{s \in \text{Par}(s')} \hat{F}(s \rightarrow s') \quad (31)$$

which is the first equality of the flow-matching condition (Eq. 21) of Prop. 4. We can obtain the second equality by first using the normalization of  $\hat{P}$ ,

$$\begin{aligned} \hat{F}(s') &= \hat{F}(s') \sum_{s'' \in \text{Child}(s')} \hat{P}_F(s''|s') \\ &= \sum_{s'' \in \text{Child}(s')} \hat{F}(s') \hat{P}_F(s''|s') \\ &= \sum_{s'' \in \text{Child}(s')} \hat{F}(s' \rightarrow s'') \end{aligned} \quad (32)$$

and then using our definition of the edge flow (Eq. 29). This proves sufficiency, following Prop. 4, i.e., the detailed balance condition implies  $\hat{F}$  is a flow.

For the other direction, if  $\hat{F}$  is a flow, we can combine Eq. 12 and Eq. 13 in Def. 13 and obtain the detailed balance equation trivially, proving its necessity.

Having already proven that  $\hat{F}$  is a flow if and only if the detailed balance equation holds, we can also show that  $\hat{P}_F$  and  $\hat{P}_B$  are compatible (Def. 16) as follows. First we combine Eq. 29 and Eq. 32 (with relabeling of variables) to obtain

$$\hat{P}_F(s'|s) = \frac{\hat{F}(s \rightarrow s')}{\sum_{s' \in \text{Child}(s)} \hat{F}(s \rightarrow s')}$$

which corresponds to Eq. 26 of Def. 16. To obtain Eq. 27, we first isolate  $\hat{P}_B$  in Eq. 28, yielding

$$\hat{P}_B(s|s') = \frac{\hat{F}(s)}{\hat{F}(s')} \hat{P}_F(s'|s) = \frac{\hat{F}(s \rightarrow s')}{\hat{F}(s')}$$

and we can then replace the denominator using Eq. 31, getting Eq. 27 of Def. 16, as desired.  $\square$

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

$$\hat{P}_B(s_t|s_{t+1}) = \frac{\hat{P}_F(s_{t+1}|s_t)\hat{F}(s_t)}{\hat{F}(s_{t+1})} \quad (33)$$

for all  $s_t < s_{t+1}$  pairs. However, because we also have the constraint

$$\sum_{s_t \in \text{Par}(s_{t+1})} \hat{P}_B(s_t|s_{t+1}) = 1,$$

Eq. 33 can only be satisfied if the flows are consistent with the forward transition:

$$\sum_{s_t \in \text{Par}(s_{t+1})} \hat{P}_F(s_{t+1}|s_t)\hat{F}(s_t) = \hat{F}(s_{t+1}).$$

### 3 GFlowNets: Learning a Flow

With the theoretical preliminaries established in Sections 1 and 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  $P(s \rightarrow s'|s)$  that best match these desiderata using estimators  $\hat{F}(s)$  and  $\hat{P}(s \rightarrow s'|s)$  which may not correspond to a proper flow. Such learning machines are called Generative Flow Networks (or GFlowNets for short).

**Definition 18.** A **GFlowNet** is a pair  $(\hat{F}(s), \hat{P}_F(s_{t+1}|s_t))$  where  $\hat{F}(s)$  is a state flow function and  $\hat{P}_F(s_{t+1}|s_t)$  is a transition distribution from which one can draw trajectories  $\tau$  by iteratively sampling each state given the previous one, starting at initial state  $s_0$  and then with  $s_{t+1} \sim \hat{P}_F(s_{t+1}|s_t)$  for  $t = 0, 1, \dots$  until sink state  $s_{n+1} = s_f$  is reached for some  $n$ .

#### 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  $\mathcal{S}' = \mathcal{S} \times \mathbb{N}^0$ , where  $\mathbb{N}^0 = \{0, 1, 2, \dots\}$ , and  $(t, s_t) \mapsto s'_t$  maps an underlying state  $s_t$  in the  $t$ -th position of the trajectory into the corresponding augmented state  $s'_t$ . With this augmented state space, we automatically avoid cycles. Furthermore, we may choose the backwards transition probabilities  $P_B(s'_t|s'_{t+1} = (t+1, s_{t+1}))$  to create a preference for shorter paths towards  $s_{t+1}$ , as discussed in Sec. 5.2. Note that we can further generalize this setup by replacing  $\mathbb{N}^0$  with any totally ordered indexing set; the augmented state space will still have an associated DAG. The ordering  $\leq$  in the original state-space is lifted to the augmented state-space:  $(t, s_t) < (t', s'_t)$  iff  $t < t'$  (in the integers) and  $s_t < s'_t$  (in the underlying state space ordering).

### 3.2 Estimating Transition Probabilities from Terminal Flows

In the setting of Bengio et al. (2021) we are given terminal flows that correspond to a terminal reward function  $R$  that is a **deterministic function** of the state:

$$R(s) := F(s \rightarrow s_f). \quad (34)$$

Note that we can extend the framework to handle random rewards in various ways (see Def. 39). The above equation induces the (generally intractable) partition function or total flow:

**Corollary 3.** *With the terminal reward function  $R$  defined as in Eq. 34, we obtain*

$$Z = F(s_0) = F(s_f) = \sum_{s \in \text{Par}(s_f)} R(s). \quad (35)$$

*Proof.* Apply the flow-matching condition (first line of Eq. 21) to  $s' = s_f$  and we obtain

$$F(s_f) = \sum_{s \in \text{Par}(s_f)} F(s \rightarrow s_f) = \sum_{s \in \text{Par}(s_f)} R(s).$$

Applying Eq. 5 and Eq. 6, we obtain Eq. 35 as desired.  $\square$

Note how an estimate of  $Z$  (via an estimator  $\hat{F}(s_0)$ , for example), combined with an estimator of the state flow  $\hat{F}(s)$  provides an estimator of the state visitation probability,

$$\hat{P}(s) := \frac{\hat{F}(s)}{\hat{F}(s_0)}. \quad (36)$$

**Definition 19.** *Given a target terminal reward function  $R$ , its associated **energy function**  $\mathcal{E}$  is:*

$$\mathcal{E}(s) := -\log R(s) \quad (37)$$

*Alternatively,*

$$e^{-\mathcal{E}(s)} := R(s) \quad (38)$$

The energy function or reward function can be used to specify a target probability distribution for terminating states:

**Definition 20.** *A GFlowNet’s **terminating state probability**  $P_T(s)$  is the probability over terminating states  $s$  under the sampling probability  $P$  of the GFlowNet of terminating a trajectory with the transition  $s \rightarrow s_f$ :*

$$P_T(s) = P(s \rightarrow s_f). \quad (39)$$

Given a terminal reward function  $R(s)$ , we attempt to find a forward transition probability function  $P(s_t \rightarrow s_{t+1} | s_t)$  (also denoted  $P_F(s_{t+1} | s_t)$ ) that assigns the desired target flows  $R(s)$  to the terminal events  $s \rightarrow s_f$ . This corresponds to the (generally intractable) problem of sampling from the terminating state distribution  $P_T(s)$  such that  $P_T(s) = \frac{R(s)}{Z}$ , given the reward function  $R$  or, equivalently, given the energy function  $\mathcal{E}$ .

### 3.3 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 learner that generalizes could guess the presence of modes it has not discovered 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).

### 3.4 Flow Matching and Detailed Balance Losses

To train a GFlowNet, we need to construct a training procedure that implicitly enforces our constraints and preferences. In this section, we transform the flow-matching or detailed balance conditions into usable loss functions. More specifically, we will generate sample training trajectories from a training policy  $\pi_T$  and construct loss functions that penalize the GFlowNet’s estimators departing from Eq. 21 or from Eq. 28.

**Definition 21.** A training distribution  $\pi_T$  is said to have **full support** (implicitly, over some given state space and its order relation  $<$ ) if every trajectory  $\tau$  consistent with the order relation  $<$  and the DAG has  $\pi_T(\tau) > 0$ .

In general, we can write training objectives as an expectation  $\mathcal{L}$  (which we will typically minimize by stochastic gradient descent, sampling trajectories  $\tau$

from  $\pi_T$ )

$$\mathcal{L} = E_{(s_0, s_1, \dots, s_n, s_f) \sim \pi_T} \left[ \sum_{t=0}^n L(s_t, s_{t+1}) \right] \quad (40)$$

where  $L(s_t, s_{t+1})$  is a per-state and/or per-transition loss and  $s_{n+1} = s_f$ . For example, the training loss proposed by Bengio et al. (2021) is

$$L_{FM}(s_t, s_{t+1}) = L_{FM}(s_t) = \left( \log \left( \frac{\delta + \sum_{s \in \text{Par}(s_t)} \hat{F}(s \rightarrow s_t)}{\delta + \sum_{s' \in \text{Child}(s_t)} \hat{F}(s_t \rightarrow s')} \right) \right)^2 \quad (41)$$

where  $\delta \geq 0$  is a hyper-parameter which allows to reduce the importance given to small flows (smaller than  $\delta$ ). The authors propose using the square of the log-ratio to ensure matching flows in states with large flows are not receiving too much gradient compared to states with small flows. With this approach, the primary object of learning is the edge flow  $\hat{F}(s \rightarrow s')$ , from which one can derive the state flow via Eq. 21 by either summing the incoming or outgoing flows.

Note that we define  $\hat{F}(s \rightarrow s_f) = R(s)$  to obtain the correct flow on the terminal transitions, so the only thing we really need is the flow-matching loss, which will propagate that constraint into the rest of the flow network. We assume this reward matching constraint throughout this work unless specified otherwise. If  $R$  isn't available at test time or is too expensive to evaluate, we can evaluate  $\hat{F}(s \rightarrow s_f)$  in at least two ways. If the flow is matched at state  $s$ , i.e.,  $L_{FM}(s) = 0$ , we have  $\hat{F}(s \rightarrow s_f) = \sum_{s' \in \text{Par}(s)} F(s' \rightarrow s) - \sum_{s'' \in \text{Child}(s); s'' \neq s_f} F(s \rightarrow s'')$ .

Alternatively, we can learn  $\hat{F}(s \rightarrow s_f)$  directly using a reward-matching loss:

$$\mathcal{L}_R = E_{\pi_T} [L_R(s)], \quad (42)$$

where  $s$  is the terminating state associated with a trajectory  $\tau \sim \pi_T$ . For example, Bengio et al. (2021) used the following:

$$L_R(s) = \left( \log \left( \frac{\delta + R(s)}{\delta + \hat{F}(s \rightarrow s_f)} \right) \right)^2. \quad (43)$$

An interesting alternative to Eq. 43 is the following **MSE reward-matching loss**:

$$L_R(s) = \left( R(s) - \hat{F}(s \rightarrow s_f) \right)^2, \quad (44)$$

which more heavily penalizes large deviations from the given reward configurations and makes  $\hat{F}(s \rightarrow s_f)$  estimate  $E[R(s)|s]$  in case the reward  $R$  is not deterministic. Similarly, minimizing Eq. 43 with  $\delta \rightarrow 0$  makes  $\log \hat{F}(s \rightarrow s_f)$  estimate  $E[\log R(s)|s]$ .

**Proposition 7.** *If a GFlowNet yields a flow, then it is a Markovian flow. As a consequence, minimizing a flow-matching loss gives rise to a Markovian flow.*

*Proof.* This is a consequence of how we define the trajectory probability  $P(\tau)$  in a GFlowNet, i.e., by drawing a sequence of states sequentially by iteratively sampling from  $P_F(s_{t+1}|s_t)$ . Applying Corollary 2 gives the result and the definition of flow-matching loss gives the last statement.  $\square$

**Definition 22.** A loss  $L$  is called **flow-matching**, if minimizing its expectation in the space of all GFlowNets over a training distribution  $\pi_T$  with full support yields a Markovian flow under the constraint that  $\hat{F}(s \rightarrow s_f) = R(s)$  for all  $s$ .

**Proposition 8.**  $L_{FM}$  (Eq. 41) is a flow-matching loss.

*Proof.*  $L_{FM}(s_t)$  cannot be negative and has a value of 0 achieved if and only if

$$\sum_{s \in \text{Par}(s_t)} \hat{F}(s \rightarrow s_t) = \sum_{s' \in \text{Child}(s_t)} \hat{F}(s_t \rightarrow s').$$

Hence its expected value has a global minimum of 0 when the flow is matched on all the states.  $\square$

Note that since terminal transitions are all to  $s_f$ , the function which estimates the flow  $F(s \rightarrow s')$  could start by checking if  $s' == s_f$ , and if so return  $R(s)$ , and otherwise return the output of the neural net estimating the edge flow. Hence, we do not actually need  $L_R$  if  $R$  is a deterministic function of  $s$ . We may still use a loss term like  $L_R$ , either because  $R$  is noisy (and a log-domain regression loss allows to estimate the conditional expected value of the energy  $-\log R$ ) or because  $L_R$  may provide additional training data for the flow predictor, thus regularizing it so it is consistent across both terminal and non-terminal transitions.

An alternative to Eq. 41 is a loss based on detailed balance, which avoids summing over all predecessors and all successors of a state, but still requires normalization of probabilities over successors or predecessors:

$$L_{DB}(s_t, s_{t+1}) = \left( \log \left( \frac{\delta + \hat{F}(s_t) \hat{P}_F(s_{t+1}|s_t)}{\delta + \hat{F}(s_{t+1}) \hat{P}_B(s_t|s_{t+1})} \right) \right)^2 \quad (45)$$

where the GFlowNet is parametrized by three functions: the estimated state flow  $\hat{F}(s)$ , the estimated forward transition probability function  $\hat{P}_F(s_{t+1}|s_t)$  and the estimated backward transition probability function  $\hat{P}_B(s_t|s_{t+1})$ . This assumes that the learner is free to choose both forward and backward transitions, with the exception of the backward transition from the sink state to a terminating state, where we implicitly set  $\hat{F}(s_f) \hat{P}_B(s|s_f) := R(s)$  since  $\hat{P}_B(s|s_f)$  is generally intractable to model because of the exponential number of possible terminating states  $s$ . Unlike in the case of the flow-matching loss, where we have the choice of either using  $R(s)$  or a learned  $\hat{F}(s \rightarrow s_f)$  during sampling, we always use the learned forward policy  $\hat{P}_F$  to determine termination.

In Sec. 5 we study the case of a learning agent acting in a stochastic environment where the transitions depend both on the actions of a learner as well as a stochastic environment.

**Proposition 9.**  $L_{DB}$  is a flow-matching loss.

*Proof.*  $L_{DB}(s_t, s_{t+1})$  attains its minimal value of 0 when

$$\hat{F}(s_t)\hat{P}_F(s_{t+1}|s_t) = \hat{F}(s_{t+1})\hat{P}_B(s_t|s_{t+1}),$$

i.e., the detailed balance condition is achieved. When its expected value achieves its global minimum of 0, it thus means that detailed balance is achieved on all transitions. By Prop. 6, this implies that the flows are matched.  $\square$

**Definition 23.** We say that a GFlowNet is **trained to completion** if it satisfies the flow-matching conditions and that terminal transition flows match the given terminal reward function  $R$ .

**Notation:** We use  $P$  and  $F$  for probability and flow when a GFlowNet is trained to completion, and  $\hat{P}$  and  $\hat{F}$  when it may be that it is not trained to completion, i.e., it is not guaranteed that the flows are matched properly.

**Proposition 10.** If a global minimum of the expected flow-matching loss over a full-support training distribution is obtained, then a GFlowNet is trained to completion.

*Proof.* The global minimum is when the flows are matched in expectation over the training distribution. Because the training distribution has full support, it means that the flow matching is achieved on all states and all transitions respectively. By construction, the terminal transition flows match the given reward function  $R$ .  $\square$

**Proposition 11.** If a GFlowNet is trained to completion, then its initial flow is the partition function of the terminal reward function, and it samples terminating states with probability proportional to the terminal reward function:

$$F(s_0) = \sum_s R(s) \tag{46}$$

$$P_T(s) = P(s \rightarrow s_f) = \frac{R(s)}{Z}. \tag{47}$$

*Proof.* If a GFlowNet is trained to completion, it means that it satisfies the flow-matching conditions and its terminal transition flows match the reward function. We can thus apply Corollary 3 and obtain Eq. 46. Applying the definition of  $P_T$  (Eq. 39) and substituting  $P(s \rightarrow s_f) = \frac{F(s \rightarrow s_f)}{Z}$  and Eq. 34, we obtain Eq. 47.  $\square$

Crucially, Prop. 11 implies that we can sample *efficiently* from a trained GFlowNet.

### 3.5 Stochastic Rewards

We also consider the setting in which the reward is stochastic rather than being a deterministic function of the state. With a reward-matching loss like that of Eq. 44, the effective target for the terminal flow  $F(s \rightarrow s_f)$  is the expected reward  $E_R[R(s)]$  since this is the value that minimizes the loss in expectation over  $R(s)$ , given  $s$ . With a reward-matching loss like that of Eq. 43, the effective target for the log of the terminal flow  $\log F(s \rightarrow s_f)$  is the expected value of the log-reward,  $E_R[\log R(s)]$ .

This demonstrates how GFlowNets can also be generalized to match stochastic rewards when using a reward-matching loss.

### 3.6 GFlowNets can be Trained Offline

The above results (Prop. 10-Prop. 11) show that we do not need to train a GFlowNet using samples from its own trajectory distribution  $\hat{P}(\tau = (s_0, s_1, \dots, s_n)) = \prod_t \hat{P}(s_{t+1}|s_t)$ . Those training trajectories can be drawn from any training distribution  $\pi_T$  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  $\pi_T$ , 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  $\pi_T$  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  $\pi_T$ ? 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  $\pi$  it might not have sufficient support and thus miss modes of  $R$ , i.e., regions where  $R$  is substantially greater than 0. 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,  $\pi_T$  could be the policy of a 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  $\pi$ , to make sure those are not forgotten, even temporarily.

### 3.7 Direct Credit Assignment in GFlowNets

Similarly to temporal-difference methods, which are based on minimizing the mismatch with respect to the Bellman equations, the flow-matching and detailed-balance losses will take many updates (and sampling many trajectories) to prop-



agate a reward mismatch on a terminating state into the transition probabilities inside the flow network. This would be particularly acute for longer trajectories and prompts the question of alternative more direct training objectives. Given a training trajectory  $\tau$ , are there more direct ways of assigning credit to the earlier transitions in the trajectory?

We can view the process of sampling a trajectory with a GFlowNet as analogous to sampling a sequence of states in a stochastic recurrent neural network. What makes things complicated is that such a neural network (i) does not directly output a prediction to be matched by some target, and (ii) that the state may be discrete (or a combination of discrete and continuous components).

Regarding (i), we recall that the flow in a transition  $s \rightarrow s'$  (involved in the training objective) is an intractable sum over all the possible trajectories leading to  $s$ . However, we may be able to obtain a stochastic gradient that, in average over such trajectories, estimates the desired quantity. For this, we exploit the properties of flows to obtain a stochastic gradient estimator, derived through the next three propositions.

**Notation for derivatives through the flow-matching constraint:** below we sometimes need to distinguish total derivatives (noted  $\frac{dy}{dx}$ ) that take into account the indirect effects due to the flow matching constraint from other derivatives (noted  $\frac{\partial y}{\partial x}$ ) and capturing only direct gradients. Either notation can be used when the constraint does not change the result. Computing indirect gradients through implicit dependencies is an active area of research and commonly utilizes the Implicit Function Theorem, e.g., for implicit layers in fixed-point iteration layers and Deep Equilibrium Models (Bai et al., 2019).

**Proposition 12.** *Consider the effect of a slight change in the log of the flow at  $s < s'$  under the flow-matching constraint: it yields a change in the log of the flow at  $s'$ , following the conditional probability  $P(s|s')$ :*

$$\frac{d \log F(s')}{d \log F(s)} = P(s|s') \quad (48)$$

where  $P$  is the distribution on events over trajectories.

*Proof.* We are going to consider a partition of the complete trajectories going through  $s'$  into those that also go through  $s$  (set  $\mathcal{T}_{s'}^s$ ) and those that don't (set  $\mathcal{T}_{s'}^{-s}$ ). By definition we can write:

$$P(s') = P(\mathcal{T}_{s'}^s) + P(\mathcal{T}_{s'}^{-s})$$

Because  $\mathcal{T}_{s'}^s = \{\tau \in \mathcal{T} : s \in \tau, s' \in \tau\}$ , we can write  $P(\mathcal{T}_{s'}^s) = P(s)P(s'|s)$ . Hence:

$$F(s') = F(s)P(s'|s) + F(\mathcal{T}_{s'}^{-s}).$$

Additionally, because the flow in  $s$  does not influence trajectories in  $\mathcal{T}_{s'}^{-s}$ , then

$\frac{dF(\mathcal{T}_{s'}^{-s})}{dF(s)} = 0$ ; which leads to:

$$\begin{aligned}\frac{dF(s')}{dF(s)} &= P(s'|s) \\ \frac{dF(s')}{d \log F(s)} &= F(s)P(s'|s) \\ &= F(s \cap s') \\ &= P(s|s')F(s') \\ \frac{d \log F(s')}{d \log F(s)} &= P(s|s').\end{aligned}$$

□

**Proposition 13.** *Consider the effect of a slight change in the log of the flow on the edge  $s \rightarrow s'$ : we obtain a change in the log of the flow at  $s'$  following the backward conditional probability  $P_B(s|s')$ :*

$$\frac{d \log F(s')}{d \log F(s \rightarrow s')} = P_B(s|s') \quad (49)$$

where  $P$  is the distribution on events over trajectories.

*Proof.* We first use the chain rule and properties of the derivatives of the log, and then we use from the flow matching constraint that  $F(s') = \sum_{s \in \text{Par}(s')} F(s \rightarrow s') \Rightarrow \frac{dF(s')}{dF(s \rightarrow s')} = 1$ :

$$\begin{aligned}\frac{d \log F(s')}{d \log F(s \rightarrow s')} &= \frac{1}{F(s')} \frac{dF(s')}{dF(s \rightarrow s')} \frac{dF(s \rightarrow s')}{d \log F(s \rightarrow s')} \\ &= \frac{F(s \rightarrow s')}{F(s')} \frac{dF(s')}{dF(s \rightarrow s')} \\ &= \frac{F(s \rightarrow s')}{F(s')} \\ &= P_B(s|s').\end{aligned}$$

□

Intuitively, Eq. 48 and Eq. 49 tell us how a perturbation in the flow at one place would result in a change elsewhere in order to maintain the flow match everywhere, using the flow-matching conditions to propagate infinitesimal changes in flow backwards and forwards. Hence, they only become true as we approach the limit of matched flows, and in practice (with an imperfectly trained GFlowNet) the corresponding expressions will be biased. However, we can exploit them to estimate long-range equilibrium gradients and obtain an estimator of credit assignment across a long trajectory in Prop. 14 below.

In the GFlowNet setting, suppose we parametrize the edge flow estimator  $F_\theta(s \rightarrow s')$  via parameters  $\theta$ . In order to understand the effect of a change in  $\theta$

on our loss function  $\mathcal{L}$ , we must compute the *total* derivative  $\frac{d\mathcal{L}}{d\theta}$  by summing the *direct* and *indirect* gradients. In our context, the direct gradient  $\frac{\partial L}{\partial \theta}$  is due to the explicit change in loss from changing  $\theta$  (not taking the flow-matching constraint into account) and the indirect gradient includes the induced changes in the flow due to the constraint. With this, we are in a position to formalize unbiased estimators for the total derivative  $\frac{d\mathcal{L}}{d\theta}$  in Prop. 14:

**Proposition 14.** *Let  $\mathcal{L}$  be a flow-matching loss computed along a trajectory  $\tau = (s_0, s_1, \dots, s_f)$  sampled according to the GFlowNet trajectory distribution  $P(\tau)$ . Let  $\mathcal{L} = \sum_i L(s_i)$  decompose the total loss into per-state losses  $L(s_i)$  along the trajectory. Let  $\theta$  parametrize the edge flow estimator  $F_\theta(s \rightarrow s')$ . Then, in the limit of the flows becoming matched,*

$$G_1 := \sum_i \frac{\partial L(s_i)}{\partial \theta} + \frac{\partial L(s_i)}{\partial \log F(s_i)} \sum_{t=1}^{i-1} \frac{\partial \log F_\theta(s_{t-1} \rightarrow s_t)}{\partial \theta} \quad (50)$$

*is an unbiased estimator of the total derivative  $\frac{d\mathcal{L}}{d\theta}$ , as is*

$$G_2 := \sum_i \frac{\partial L(s_i)}{\partial \theta} + \frac{\partial L(s_i)}{\partial \log F(s_i)} \sum_{t=1}^{i-1} \sum_{s \in \text{Par}(s_t)} P_B(s|s_t) \frac{\partial \log F_\theta(s \rightarrow s_t)}{\partial \theta} \quad (51)$$

*as is the convex combination*

$$G = \lambda G_1 + (1 - \lambda) G_2 \quad (52)$$

*for any  $0 \leq \lambda \leq 1$ . Intuitively,  $\frac{\partial L(s_i)}{\partial \theta}$  indicates the gradient directly through the occurrences of  $F_\theta$  in  $L(s_i)$ , and  $F(s_i)$  indicates either the forward or backward flow sum present in  $L(s_i)$  to obtain the flow through  $s_i$ .*

*Proof.* We will use the partial derivative notation when not considering the indirect influence due to the matching flow constraint and the total derivative notation when considering it, to apply Prop. 12. Consider the expected value under the GFlowNet’s trajectory distribution of the  $G$  in Eq. 50, use the previous

proposition (Eq. 48) and the chain rule:

$$\begin{aligned}
E[G_1] &= E \left[ \sum_i \frac{\partial L(s_i)}{\partial \theta} + \sum_{t=1}^{i-1} \frac{\partial L(s_i)}{\partial \log F(s_i)} \frac{\partial \log F_\theta(s_{t-1} \rightarrow s_t)}{\partial \theta} \right] \\
&= \sum_{s_i} P(s_i) \left( \frac{\partial L(s_i)}{\partial \theta} + \sum_{s \rightarrow s' < s_i} P(s \rightarrow s' | s_i) \frac{\partial L(s_i)}{\partial \log F(s_i)} \frac{\partial \log F_\theta(s \rightarrow s')}{\partial \theta} \right) \\
&= \sum_{s_i} P(s_i) \left( \frac{\partial L(s_i)}{\partial \theta} \right. \\
&\quad \left. + \sum_{s \rightarrow s' < s_i} P_B(s | s') P(s' | s_i) \left( \frac{\partial L(s_i)}{\partial \log F(s_i)} \frac{\partial \log F_\theta(s \rightarrow s')}{\partial \theta} \right) \right) \\
&= \sum_{s_i} P(s_i) \left( \frac{\partial L(s_i)}{\partial \theta} \right. \\
&\quad \left. + \sum_{s \rightarrow s' < s_i} \frac{\partial L(s_i)}{\partial \log F(s_i)} \frac{d \log F(s_i)}{d \log F(s')} \frac{d \log F(s')}{d \log F_\theta(s \rightarrow s')} \frac{\partial \log F_\theta(s \rightarrow s')}{\partial \theta} \right) \\
&= \sum_{s_i} P(s_i) \left( \underbrace{\frac{\partial L(s_i)}{\partial \theta}}_{\text{direct gradients}} + \underbrace{\sum_{s \rightarrow s' < s_i} \frac{\partial L(s_i)}{d \log F_\theta(s \rightarrow s')} \frac{\partial \log F_\theta(s \rightarrow s')}{\partial \theta}}_{\text{indirect gradients}} \right) \\
&= E \left[ \frac{d\mathcal{L}}{d\theta} \right] \\
&= \frac{dE[\mathcal{L}]}{d\theta}
\end{aligned}$$

The above demonstration shows that  $G$  in Eq. 52 is asymptotically (as the flows become matched) unbiased when  $\lambda = 1$  because we recover the  $G_1$  of Eq. 50. The same proof technique can then be used for  $G_2$  which uses transitions  $s \rightarrow s_t$  sampled from  $P_B(s | s_t)$  instead of the trajectory transitions  $s_{t-1} \rightarrow s_t$  and we obtain that the estimator  $G$  in Eq. 52 is asymptotically unbiased when  $\lambda = 0$ :

$$\begin{aligned}
E[G_2] &= E \left[ \sum_i \sum_{t=1}^{i-1} \frac{\partial L(s_i)}{\partial \theta} + \frac{\partial L(s_i)}{\partial \log F(s_i)} \sum_{s \in \text{Par}(s')} P_B(s|s') \frac{\partial \log F_\theta(s \rightarrow s_t)}{\partial \theta} \right] \\
&= \sum_{s_i} \sum_{s' < s_i} P(s', s_i) \left( \frac{\partial L(s_i)}{\partial \theta} + \sum_{s \in \text{Par}(s')} P_B(s|s') \frac{\partial L(s_i)}{\partial \log F(s_i)} \frac{\partial \log F_\theta(s \rightarrow s')}{\partial \theta} \right) \\
&= \sum_{s_i} P(s_i) \left( \frac{\partial L(s_i)}{\partial \theta} + \sum_{s' < s_i} P(s'|s_i) \sum_{s \in \text{Par}(s')} P_B(s|s') \frac{\partial L(s_i)}{\partial \log F(s_i)} \frac{\partial \log F_\theta(s \rightarrow s')}{\partial \theta} \right) \\
&= \sum_{s_i} P(s_i) \left( \frac{\partial L(s_i)}{\partial \theta} + \sum_{s \rightarrow s' < s_i} \left( \frac{\partial L(s_i)}{\partial \log F(s_i)} \frac{d \log F(s_i)}{d \log F(s')} \frac{d \log F(s')}{d \log F_\theta(s \rightarrow s')} \frac{\partial \log F_\theta(s \rightarrow s')}{\partial \theta} \right) \right) \\
&= \sum_{s_i} P(s_i) \left( \underbrace{\frac{\partial L(s_i)}{\partial \theta}}_{\text{direct gradients}} + \underbrace{\sum_{s \rightarrow s' < s_i} \left( \frac{\partial L(s_i)}{d \log F_\theta(s \rightarrow s')} \frac{\partial \log F_\theta(s \rightarrow s')}{\partial \theta} \right)}_{\text{indirect gradients}} \right) \\
&= \frac{dE[\mathcal{L}]}{d\theta}
\end{aligned}$$

where the last identity follows the fourth line in the proof for  $G_1$ . Finally a convex combination of two unbiased estimators is unbiased, so we obtain that  $G$  in Eq. 52 is asymptotically unbiased for any  $0 \leq \lambda \leq 1$ .  $\square$

This surprising result says that something very close to policy gradient actually provides an asymptotically (i.e., when flows are matched) unbiased gradient on the parameters of the edge flow, in expectation<sup>2</sup>. Note that it only works exactly in an online setting, i.e., when the trajectory is sampled according to the learner's current policy. Otherwise, the gradient estimator may be biased (it would be biased anyways in practice because the flows are never perfectly matched). However, if instead of sampling trajectories  $\tau$  from the GFlowNet

---

<sup>2</sup>the connection becomes clearer when you imagine minus the loss  $L$  to be the reward itself, and we see that we immediately get a training signal at earlier times in the sequence with  $G$ , similarly to policy gradient. There are also differences because the above proposition relies on staying close to the learning fixed point where the flows are matched.

transition probabilities  $P_F(s_{t+1}|s_t)$  we sample them from a training distribution  $\hat{P}$  with transition probabilities  $\hat{P}(s_{t+1}|s_t)$  we can calculate the importance weights (by the ratio  $P(\tau)/\hat{P}(\tau)$ ) and correct the estimator accordingly. Since the training distribution  $\hat{P}$  should be broader and have a full support, the importance ratio cannot explode but there could still be the usual numerical problems with the variance of such importance-weighted estimators.

We now consider the setting in which the sampling policy is only slightly different from  $\hat{P}$ , which is typically the case because we want the sampling policy to be broader and more exploratory, and because we may be using delayed data, e.g., with a replay buffer. This slight difference may induce a bias but it might still be advantageous to use the above gradient estimator. Note how it does not come in conflict with the gradient of the flow matching loss (which is the first term in  $G$ ). The expected advantage of using  $G$  is that it may initially speed up training by directly providing updates to earlier transitions of a complete trajectory. However, analogous to the trade-off between temporal-difference methods and policy-gradient methods, this may come at the price of higher variance.

This estimator is unbiased when the flows are matched and when the trajectory is sampled according to the GFlowNet’s distribution, but it also makes a lot of intuitive sense: if the estimated flow at  $s_i$  is too small (in the eye of  $L_i$ ) one can clearly push that flow up by increasing the probability of a transition on a path leading to  $s_i$ . Even if we consider a slightly different trajectory sampling distribution, so long as it leads to  $s_i$  we would expect that increasing its probability would increase the probability of ending up in  $s_i$  (see Prop. 12).

If the state has a continuous component, we could also increase the probability of ending up in  $s_i$  by choosing more often a more probable path to  $s_i$ . This could be calculated by backpropagating through the state transitions (with some form of backpropagation through time). However, if the transitions are not fully known or are not differentiable, this approach may be more challenging, and is related to similar questions raised with credit assignment in reinforcement learning with continuous states.

Finally, keep in mind that the more direct credit assignment terms in  $G$  have to be combined with the local terms  $\frac{\partial L_i}{\partial \theta}$  which make sure that the flows becomes better matched, since flow-matching is a necessary condition for  $G$  to be unbiased.

### 3.8 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  $\tau$  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  $\tau$  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 flows is that if the detailed balance or flow matching conditions are satisfied, we can recover the normalizing constant  $Z$  from the initial state flow  $F(s_0)$  (Corollary 3).  $Z$  also gives us the partition function associated with a given terminal reward function  $R$  specifying the terminal transition flows.

What about internal states  $s$  with  $s_0 < s < s_f$ ? Can we compute the sum of rewards for all the terminating states  $s'$  reachable from  $s$ ? If we had something like a normalizing constant for only the terminal rewards achievable from  $s$ , we would be able to obtain a form of marginalization given state  $s$ . Naturally, one could ask: does  $F(s)$  give us that kind of marginalization over only the downstream terminal transitions which can follow  $s$ ? Consider the terminal flow  $R(s')$  through an edge  $s' \rightarrow s_f$  with  $s < s'$ . The flow  $F(s)$  gives us the sum of the flows through  $s$  but unfortunately does not fully include the flow through  $s' \rightarrow s_f$  which contains flow from paths not going through  $s$ . To see this, consider a state  $s''$  with  $s'' < s'$  (i.e., there are trajectories through  $s''$  and  $s'$ ) but  $s'' \not\leq s$  (i.e., there are no trajectories through both  $s$  and  $s''$ ). Hence, some of the flow through the edge  $s' \rightarrow s_f$  comes from  $s$  and some from  $s''$ . These flows do not overlap and are added up (potentially with others) to form  $F(s' \rightarrow s_f)$ ; see Fig. 1.

In Section 9.3, 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 24.** *The free energy  $\mathcal{F}(s)$  of state  $s$  is:*

$$e^{-\mathcal{F}(s)} := \sum_{s': s' \geq s} R(s') = \sum_{s': s' \geq s} e^{-\mathcal{E}(s')}. \quad (53)$$

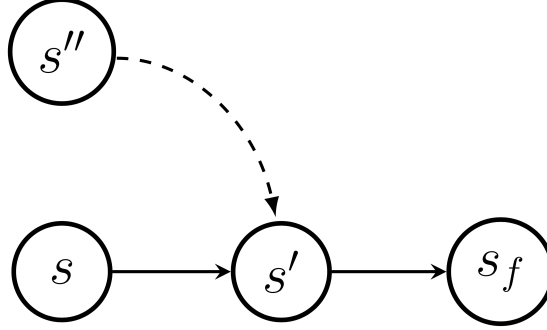


Figure 1: Some of the flow through  $s'$  comes from  $s$ , while other parts of that flow come from other sources like  $s''$  (dashed line). Hence the flow through  $s$  generally does not account all of the flow through the terminating edges accessible from  $s$ , making  $F(s)$  not correctly estimating the sum of the flows in the terminal transitions downstream of  $s$ , if the GFlowNet’s DAG allows multiple paths into the same internal node.

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 Conditioning a GFlowNet

As motivated above, besides  $F(s)$ , there is another quantity that is useful to estimate, namely the sum of the flows through  $s$  if all of the flows through any terminal edge  $s' \rightarrow s_f$  with  $s \leq s'$  were diverted towards  $s$ , which provides free energies. This quantity would not count the flows through terminal edges from states  $s''$  such that  $s \not\leq s''$ , and we will show below how it enables the computation of marginalized probabilities or free energies; see Fig. 2.

How we propose to achieve this is to train our GFlowNet estimators with an additional argument  $x$  which represents a conditioning variable. In general, the conditioning variable can represent any conditioning information, either external to the GFlowNet (but influencing the rewards) or internal (e.g.,  $x$  can be a flow event, like passing through a particular state). We will show below how such a conditional GFlowNet provides what we need. If a GFlowNet can represent a flow measure  $\hat{F}(s_t)$  and a flow transition distribution  $\hat{P}_F(s_{t+1}|s_t)$  or  $\hat{P}_B(s_t|s_{t+1})$ , using an additional input  $x \in \mathcal{X}$  (and training with a sufficiently diverse set of values of  $x$ ) would make all the outputs,  $\hat{F}(s_t|x)$ ,  $\hat{P}_F(s_{t+1}|s_t, x)$  and  $\hat{P}_B(s_t|s_{t+1}, x)$  compute the corresponding estimators conditioned on  $x$ .

We observe that in the notation  $P(s_{t+1}|s_t, x)$ , the arguments  $s_t$  and  $x$  are treated on unequal footing:  $s_t$  indicates the variable that determines which flows are included in the computation of the probability (e.g., either  $s_t \rightarrow s_{t+1}$  is an



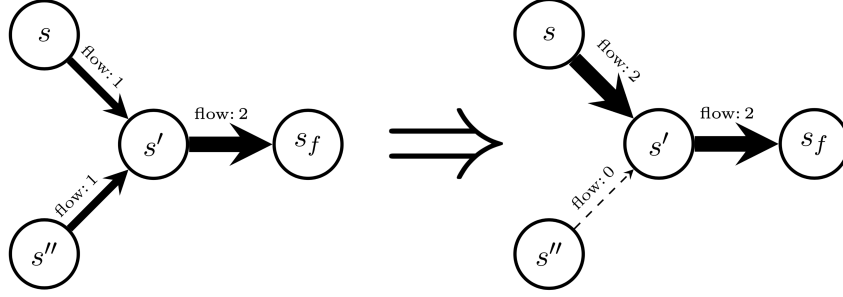


Figure 2: For a given state  $s$ , consider the original flows (left) and diverting the flows to create a new set of flows (right). We do this by diverting to  $s$  the flow to  $s'$  from any  $s'' \leq s$  such that  $s'' < s'$  and  $s' > s$  is a terminating state.

edge or  $s_{t+1} \rightarrow s_t$  is an edge, depending on whether  $P$  is a forward or backward probability) whereas  $x$  represents the conditioning variable that restricts the set of flows under consideration. This is in contrast with the usual notation in probability.

We note that conditioning on  $x$  may change the effective initial state, final state, or terminal reward function:

**Definition 25.** We denote by  $s_0|x$  and  $s_f|x$  the initial and final states in the conditions imposed by the value of conditioning variable  $x$ , or  $s_0$  and  $s_f$  for short if they do not depend on  $x$ . We use  $R(s|x)$  to denote the terminal reward function in condition  $x$ , or simply  $R(s)$  if it does not depend on  $x$ .

We will find that the results proven thus far for unconditioned GFlowNets will also apply to each member of the family of conditional flows indexed by the value of  $x$ . We begin with an analogue of Prop. 4 for the conditional flow case:

**Proposition 15.** Consider a non-negative function  $\hat{F}$  taking as input a conditioning variable  $x$  and either a state  $s$  or a transition  $s \rightarrow s'$ . Let  $\hat{F}$  define the associated forward transition probabilities estimator

$$\hat{P}_F(s_{t+1}|s_t, x) = \hat{P}(s_t \rightarrow s_{t+1}|s_t, x) := \frac{\hat{F}(s_t \rightarrow s_{t+1}|x)}{\hat{F}(s_t|x)} \quad (54)$$

and backwards transition probabilities estimator

$$\hat{P}_B(s_t|s_{t+1}, x) = \hat{P}(s_t \rightarrow s_{t+1}|s_{t+1}, x) := \frac{\hat{F}(s_t \rightarrow s_{t+1}|x)}{\hat{F}(s_{t+1}|x)}. \quad (55)$$

Then  $\hat{F}$  corresponds to a flow if and only if the incoming and outgoing flows of

each non-source and non-sink state are matched, i.e.,

$$\begin{aligned} \forall s' > s_0 \text{ s.t. } s' \text{ is consistent with } x, \quad \hat{F}(s'|x) &= \sum_{s \in \text{Par}(s')} \hat{F}(s \rightarrow s'|x) \\ \forall s' < s_f \text{ s.t. } s' \text{ is consistent with } x, \quad \hat{F}(s'|x) &= \sum_{s'' \in \text{Child}(s')} \hat{F}(s' \rightarrow s''|x) \end{aligned} \quad (56)$$

Furthermore, if the flow-matching constraints are satisfied then the transition probability estimators are correctly normalized (summing to 1).

The proof of Prop. 15 is very similar to that of Prop. 4 (except that everything is conditioned on  $x$ ) and is omitted for brevity. With the analogous flow-matching conditions for conditional flows, we can define analogous terms for conditional GFlowNets and prove results similar to those for unconditional GFlowNets:

**Definition 26.** Given a set of conditioning information  $\mathcal{X}$ , a **conditional GFlowNet** defines an estimator for a conditional state flow function,  $\hat{F}(s|x)$ , and an estimator for a transition distribution,  $\hat{P}(s_{t+1}|s_t, x)$ , from which one can draw conditional trajectories  $\tau$ , for all  $x \in \mathcal{X}$ , by iteratively sampling each state given the previous one, starting at initial state  $s_0|x$ .

**Definition 27.** A **full support training distribution**  $\pi_T(\tau, x)$  for a **conditional GFlowNet** samples both conditions  $x \sim \pi_T(x)$  ( $x \in \mathcal{X}$ ) with full support in  $\mathcal{X}$  and trajectories  $\tau \sim \pi_T(\tau|x)$  with full support with respect to order  $<$ .

**Definition 28.** A **flow-matching** conditional loss  $L$  is such that minimizing its expectation over a training distribution  $\pi_T(\tau, x)$  with full support in the space of all conditional GFlowNets yields a Markovian flow for all  $s$  and  $x \in \mathcal{X}$ .

**Definition 29.** A conditional GFlowNet is **trained to completion** if it satisfies the flow-matching conditions and matches the given terminal reward function  $R(s|x)$  for every  $x \in \mathcal{X}$ .

**Proposition 16.** If one obtains a global minimum of the expected value over a full-support training distribution of a flow-matching conditional loss, then the conditional GFlowNet is trained to completion.

*Proof.* By Def. 28, minimizing the expectation of a flow-matching conditional loss over a training distribution  $\pi_T(\tau, x)$  with full support yields a Markovian flow satisfying  $F(s \rightarrow s_f|x) = R(s|x)$ . By Prop. 4, a Markovian flow necessarily satisfies the flow-matching conditions, meaning that the GFlowNet has been trained to completion by Def. 29.  $\square$

**Proposition 17.** If a conditional GFlowNet is trained to completion, then its initial flow is the partition function of the conditional terminal reward function, and it samples terminating states with probability proportional to the terminal

reward function:

$$F(s_0|x) = \sum_{s|x} R(s|x) := Z(x) \quad (57)$$

$$P_T(s|x) = \frac{R(s|x)}{Z(x)} \quad (58)$$

where the sum over  $s|x$  runs over all the states compatible with condition  $x$ , while  $\hat{P}(s \rightarrow s_f|x)$  indicates the probability of sampling a trajectory from  $\hat{P}$  conditioned on  $x$  and ending in edge  $s \rightarrow s_f|x$ .

*Proof.* By definition,  $F(s_0|x)$  is the sum of all the flows  $\tau$  compatible with condition  $x$ , denoted as  $\tau|x$ :

$$F(s_0|x) := \sum_{\tau|x} F(\tau). \quad (59)$$

On the other hand, if the conditional GFlowNet has been trained to completion, then by Def. 29 the reward-matching conditions are satisfied:  $F(s \rightarrow s_f|x) = R(s|x)$ . We can make the flow  $F(s \rightarrow s_f)$  more explicit by considering all the trajectories  $\tau|x$  compatible with  $x$  such that  $s \rightarrow s_f \in \tau$

$$\begin{aligned} \sum_{s|x} R(s|x) &= \sum_{s|x} F(s \rightarrow s_f|x) \\ &= \sum_{s|x} \left[ \sum_{\tau|x} 1_{s \rightarrow s_f \in \tau} F(\tau) \right] = \sum_{\tau|x} \underbrace{\left[ \sum_{s|x} 1_{s \rightarrow s_f \in \tau} \right]}_{=1} F(\tau) \\ &= \sum_{\tau|x} F(\tau) = F(s_0|x). \end{aligned}$$

This shows that the initial flow is the partition function  $Z(x)$  of the terminal reward function, conditioned on  $x$ .

Moreover, by definition of  $P_T$  (see Def. 20), and using the partition function above, we have

$$P_T(s|x) := P(s \rightarrow s_f|x) = \frac{F(s \rightarrow s_f|x)}{F(s_0|x)} = \frac{R(s|x)}{Z(x)}. \quad (60)$$

□

## 4.2 Estimating Free Energies

Let us consider a special case of a conditional GFlowNet that will allow the network to estimate free energies  $\mathcal{F}(s)$  (Eq. 53). For this purpose, we propose to train a conditional GFlowNet with the conditioning input  $x$  being an earlier state  $s$  in the trajectory.

**Definition 30.** A **state-conditional GFlowNet** is a special kind of conditional GFlowNet for which the conditioning information  $\mathcal{X}$  is the set of all states in the GFlowNet and that  $(s_0|x = s)$  is  $s$  for all  $s \in \mathcal{X}$ .

Intuitively, the main impact of choosing  $x = s$  is a change in the initial state, i.e., the only trajectories sampled using the GFlowNet compatible with  $x = s$  are those following from state  $s$ , so we will make the state-conditional GFlowNet use  $s$  as the initial state.

In our case,  $R(s'|x = s) = R(s')1_{s' \geq s}$  and  $(s_f|x = s)$  equals  $s_f$ , i.e., the choice of condition  $s$  does not change the terminal reward function for states  $s'$  reachable from  $s$  (but it zeroes the others) nor the notion of final state.

We will use the detailed balance training loss and parametrization via  $\hat{F}(s'|x = s)$ ,  $\hat{P}_F(s' \rightarrow s''|s', x = s)$  and  $\hat{P}_B(s' \rightarrow s''|s'', x = s)$ , respectively taking two states (with  $s < s'$ ), three states (with  $s < s' \rightarrow s''$ ) and three states (with  $s < s' \rightarrow s''$ ) as input. Both  $\hat{F}(s'|x = s)\hat{P}_F(s''|s', x = s)$  and  $\hat{F}(s'|x = s)\hat{P}_B(s''|s', x = s)$  are 0 if  $s \not\prec s'$ . Note that unlike in the unconditional case (Def. 13),  $\hat{P}_B(s' \rightarrow s''|s'', x = s)$  is a probability distribution only over the states  $s'$  that belong to a path from  $s$  to  $s''$ .

The training objective for such a state-conditional GFlowNet could be

$$\mathcal{L} = E_{(s_0, s_1, \dots, s_n, s_f) \sim \pi_T} \left[ \sum_{t=0}^n E_{0 \leq t' \leq t} [L(s_t, s_{t+1}|x = s_{t'})] \right] \quad (61)$$

where we introduced an additional (conditioning) argument to the loss function, obtaining a conditional loss  $L(s_t, s_{t+1}|x = s_{t'})$  and where  $0 \leq t' \leq t$  is sampled in a way that covers all  $t' \leq t$ , e.g., uniformly (or alternatively the full double sum over  $t'$  and  $t$  can be run to train over all possible conditioning states rather than a particular one). The per-triplet loss could be defined as follows, using the detailed balance loss in Eq. 45, but with the extra conditioning argument:

$$L(s', s''|x = s) = \left( \log \left( \frac{\delta + \hat{F}(s'|x = s)\hat{P}_F(s''|s', x = s)}{\delta + \hat{F}(s''|x = s)\hat{P}_B(s'|s'', x = s)} \right) \right)^2 \quad (62)$$

where  $s' \rightarrow s'' \in \mathbb{A}$  and  $\hat{F}(s_f)\hat{P}_B(s''|s_f, x = s) := R(s''|x = s)$ .

**Proposition 18.**  $L$  as defined in Eq. 62 is a flow-matching conditional loss.

*Proof.* First, we recall that for a state-conditioned GFlowNet, the set of conditioning information  $\mathcal{X}$  is the set of all states. To show that Eq. 62 is a flow-matching conditional loss, we need to show that when minimized in expectation over all transitions and conditioning variables,  $\hat{F}(\cdot|x = s)$  yields a Markovian flow for all  $s \in \mathcal{X}$ . When  $L$  is minimized, we obtain  $\hat{F}(s'|x = s)\hat{P}_F(s''|s', x = s) = \hat{F}(s''|x = s)\hat{P}_B(s'|s'', x = s)$  for all  $(s, s', s'')$ -triplets that satisfy  $s < s'$  and  $s' \rightarrow s'' \in \mathbb{A}$ . Moreover, this identity trivially holds for  $s \not\prec s'$  since both sides are 0 by definition; therefore, the detailed balance condition holds for all  $s \in \mathcal{X}$ . By Prop. 6 and Prop. 7, we obtain that  $\hat{F}(\cdot|x = s)$  is a Markovian flow for all  $s \in \mathcal{X}$ .  $\square$

**Definition 31.** We call  $F(s|x = s)$  the **conditional state self-flow**. Intuitively,  $F(s|x = s)$  represents the flow through  $s$  when only the trajectories through  $s$  are allowed and yield the desired flows  $R(s')$  through all the  $s' \geq s$ .

**Proposition 19.** If the conditional training loss is flow-matching and the state-conditional GFlowNet is trained to completion then we obtain the free energy  $\mathcal{F}(s)$  (Eq. 53) from the conditional state self-flow as follows:

$$e^{-\mathcal{F}(s)} = F(s|x = s) \quad (63)$$

$$= \sum_{s' \geq s} R(s') \quad (64)$$

$$= \sum_{s' \geq s} F(s' \rightarrow s_f) \quad (65)$$

*Proof.* We apply Prop. 17 (Eq. 57) to the special case of  $x = s$  and obtain Eq. 64 since the condition  $x = s$  limits the trajectories to those reachable from  $s$ . Taking  $s' = s$  and combining with the definition of free energy (Eq. 53), we obtain Eq. 63.  $\square$

**Definition 32.** Let us define the **conditional terminating probability distribution** as follows:

$$\begin{aligned} P_T(s|A) &:= \frac{1_{s \in A} P(s \rightarrow s_f)}{\sum_{s' \in A} P(s' \rightarrow s_f)} \\ &= \frac{\sum_{\tau \in A} 1_{s \rightarrow s_f \in \tau} P(\tau)}{\sum_{\tau \in A} P(\tau)} \end{aligned} \quad (66)$$

where  $A$  is any condition on the trajectories leading to state  $s$  and  $s \in A$  indicates that  $s$  is compatible with  $A$ .  $P_T$  only has the terminating events as possible mutually exclusive outcomes, and conditioning must be understood in this way. Consistent with this, we obtain the special case

$$P_T(s) = P_T(s|\mathcal{T}) = P(s \rightarrow s_f) = \frac{R(s)}{F(s_0)} = e^{-\mathcal{E}(s) + \mathcal{F}(s_0)} \quad (67)$$

where  $\mathcal{T}$  is the set of all possible trajectories under order  $<$ .

Note that  $P_T(s'|s < s')$  is different from  $P(s'|s < s')$  in general because the latter counts all the flows through  $s'$  whereas the former only counts the flow through  $s' \rightarrow s_f$ . Conditioning  $P_T$  on  $A$  also has a different meaning: it restricts the set of outcomes to those compatible with  $A$  rather than dealing with sets of trajectories.

**Proposition 20.** A state-conditional GFlowNet trained to completion can compute conditional probability  $P_T(s'|x = s)$  of terminating the trajectory with state  $s'$  given earlier state  $s < s'$  with

$$P_T(s'|x = s) = 1_{s' \geq s} \frac{F(s' \rightarrow s_f)}{F(s|x = s)} = \frac{R(s')}{\sum_{s'' \geq s} R(s'')} \quad (68)$$

$$= 1_{s' \geq s} e^{-\mathcal{E}(s') + \mathcal{F}(s)}. \quad (69)$$

*Proof.* If  $s'$  is not reachable from  $s$ , then clearly the conditional probability is zero. Otherwise, by definition of  $P_T$  (Eq. 66) and substituting  $F$ 's for  $P$ 's

$$P_T(s'|x=s) = \frac{F(s' \rightarrow s_f)/Z}{\sum_{s'' \geq s} F(s'' \rightarrow s_f)/Z}$$

which gives Eq. 68 by cancelling the  $Z$ 's and using Eq. 64. To obtain Eq. 69 from Eq. 68, we apply Eq. 37 to the numerator and Eq. 63 to the denominator.  $\square$

### 4.3 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. 3.3). Consider the model  $P_\theta(s)$  associated with a given parametrized energy function  $\mathcal{E}_\theta(s)$  with parameters  $\theta$ :  $P_\theta(s) = \frac{e^{-\mathcal{E}_\theta(s)}}{Z}$ . Sampling from  $P_\theta(s)$  could be approximated by sampling from the terminating probability distribution  $P_T(s)$  of a GFlowNet trained to completion with target terminal reward  $R(s) = e^{-\mathcal{E}_\theta(s)}$  (see Eq. 39). In practice,  $\hat{P}_T$  would be an estimator for the true  $P_\theta$  because the GFlowNet training objective is not zeroed (insufficient capacity and 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  $\theta$  of an energy function  $\mathcal{E}_\theta$ :

$$\frac{\partial -\log P_\theta(x)}{\partial \theta} = \frac{\partial \mathcal{E}_\theta(x)}{\partial \theta} - \sum_s P_\theta(s) \frac{\partial \mathcal{E}_\theta(s)}{\partial \theta}. \quad (70)$$

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 were trained to completion, 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  $\theta$  using the above equation (with sampling from  $P_\theta$  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  $\hat{F}(s \rightarrow s_f) = R(s)$  by construction (which we can do if the reward function is deterministic, and in which case there is no need for a reward matching loss), then we can parametrize the energy function with the same neural network that computes the flow, since  $\mathcal{E}(s) = -\log R(s) = -\log \hat{F}(s \rightarrow s_f)$ . 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, with the following values and constraints:

$$s = s_0 \Rightarrow u = 0 \quad (71)$$

$$s = s_f \Rightarrow u = 5 \quad (72)$$

$$(u_t \rightarrow u_{t+1}) \in \{0 \rightarrow 1, 0 \rightarrow 2, 1 \rightarrow 1, 2 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 4, 3 \rightarrow 3, 4 \rightarrow 4, 3 \rightarrow 5, 4 \rightarrow 5\} \quad (73)$$

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  $P_T(x)$ ,  $P_T(h|x)$ ,  $P_T(h)$ ,  $P_T(x|h)$  as well as  $P_T(x, h)$ . If we only want to sample  $x$  (or only  $h$ ), we can exit as soon as it is generated (resp.  $h$  is generated). See Sec. 9.3 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. When  $x$  is observed but  $h$  is not,  $\theta$  could thus be updated by approximating the marginal log-likelihood gradient

$$\frac{\partial -\log P_\theta(x)}{\partial \theta} = \sum_h P_T(h|x) \frac{\partial \mathcal{E}_\theta(x, h)}{\partial \theta} - \sum_{x', h} P_T(x', h) \frac{\partial \mathcal{E}_\theta(x', h)}{\partial \theta} \quad (74)$$

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 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  $\pi_T$ .

#### 4.4 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. Bengio et al. (2021) 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 protein to a given target). Thus, instead of using the true reward directly, the authors introduce a proxy  $\hat{f}$  (which approximates the true reward), which is used to train the GFlowNet. This would lead to a setup similar to Sec. 4.3, with an inner loop where 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 GFlowNet, and  $y$  is the corresponding *true* reward from the environment (for example, outcome of a biological or 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 GFlowNet. And similarly, since the GFlowNet matches a reward distribution defined by  $\hat{f}$ , it also depends on the quality of  $f$ .

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), Expected Improvement (EI)) from Bayesian optimization 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 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.

## 4.5 Estimating Entropies, Conditional Entropies and Mutual Information

**Definition 33.** *Given a reward function  $R$  with  $0 \leq R(s) < 1 \forall s$ , we define an entropic reward function  $R'$  associated with  $R$  is*

$$R'(s) = -R(s) \log R(s). \quad (75)$$

In brief, in this section, we find 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 21.** *If  $R(s) < 1$  for all  $s$  and a GFlowNet with flows  $F$  is trained to completion to achieve target terminal rewards  $R(s)$  while another with flows  $F'$  is trained to completion with associated entropic terminal rewards  $R'(s)$  (Eq. 75), then the entropy  $H[S]$  associated with the terminating state random variable  $S$  with distribution  $P_T(S = s) = \frac{R(s)}{Z}$  (Eq. 39) is*

$$H[S] := - \sum_s P_T(s) \log P_T(s) = \frac{F'(s_0)}{F(s_0)} + \log F(s_0). \quad (76)$$

*Proof.* First apply the definition of  $P_T(s)$ , then Eq. 35 on both flows:

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

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

**Proposition 22.** *If  $R(s) < 1$  for all  $s$  and a conditional GFlowNet with flows  $F$  is trained to completion with target terminal rewards  $R(s)$  while another with flows  $F'$  is trained to completion with associated entropic terminal rewards  $R'(s)$  (Eq. 75), then the conditional entropy  $H[S|x]$  of random terminating states  $S$  consistent with condition  $x$  is given by*

$$H[S|x] = \frac{F'(s_0|x)}{F(s_0|x)} + \log F(s_0|x). \quad (77)$$

where  $F(s_0|x)$  and  $F'(s_0|x)$  are the conditional flows of the GFlowNets respectively trained with target terminal rewards  $R$  and  $R'$  respectively. In particular, for a state-conditional GFlowNet ( $x = s$  being a GFlowNet state), we obtain

$$H[S|s] = \frac{F'(s|s)}{F(s|s)} + \log F(s|s). \quad (78)$$

*Proof.* We apply the same proof as for Prop. 21 but conditioning everything on  $x$ , thus obtaining Eq. 77. When  $x$  is an event in the trajectory, this restricts the set of trajectories being considered to those consistent with that event. In particular, with  $x = s$ , every (remaining) trajectory goes through  $s$  (which thus plays a role similar to the initial state), and  $F(s_0|s) = F(s|s)$ , which gives us Eq. 78.  $\square$

**Corollary 4.** *Let the reward function  $0 \leq R(s) < 1$  for all  $s$ , and a conditional GFlowNet with flows  $F$  (both conditional on  $x$  and unconditional) be trained to completion with target terminal rewards  $R(s)$  while another with flows  $F'$  is similarly trained to completion with associated entropic terminal rewards  $R'(s)$  (Eq. 75). The mutual information  $MI(S; X)$  between the random draw of a terminating state  $S = s$  according to  $P_T(s|x)$  and the conditioning random variable  $X$  is*

$$MI(S; X) = H[S] - E_X[H[S|X]] = \frac{F'(s_0)}{F(s_0)} + \log F(s_0) - E_X \left[ \frac{F'(s_0|X)}{F(s_0|X)} + \log F(s_0|X) \right] \quad (79)$$

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

*Proof.* We apply the definition of mutual information in the first equality above, plugging Eq. 76 and Eq. 77 to obtain the second equality above.  $\square$

If we have a sampling mechanism for  $P(X)$ , we can thus approximate the expectation in Eq. 79 by a Monte-Carlo average with draws from  $P(X)$ .

## 5 Policies in Deterministic and Stochastic Environments

Until now, we have focused on a deterministic environment where state changes can perfectly be calculated from a given action. This makes sense when the actions are cognitive actions, internal to an agent, e.g., sequentially constructing a candidate solution to a problem (an explanation, a plan, an inferred guess, etc). What about the scenario where the actions are external and affect the real world? The outcome are likely to be only imperfectly predictable. To address this scenario, we will now extend the GFlowNet framework to learn a policy  $\pi$  for an agent in an environment that could be deterministic or stochastic.

**Definition 34.** A policy  $\pi : \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$  is a probability distribution  $\pi(a|s)$  over actions  $a \in \mathcal{A}$  for each state  $s$ . To denote the fact that the action space may be restricted based on  $s$ , we write  $\mathcal{A}(s)$  for the valid actions in state  $s$ .

To denote the introduction of actions in the GFlowNet framework, we will decompose transitions in two steps: first an action  $a_t$  is sampled according to a policy  $\pi$  from state  $s_t$ , and then the environment transforms this (in a possibly stochastic way) into a new state  $s_{t+1}$ .

**Definition 35.** We generalize the notion of state as follows: **even states** are of the form  $s \in \mathcal{S}$  while **odd states** are of the form  $(s, a) \in \mathcal{S} \times \mathcal{A}$ . The policy  $\pi$  governs the transition from an even state to a compatible next odd state with  $a \in \mathcal{A}(s)$ , while the **environment**  $P(s_t \rightarrow s_{t+1} | s_t, a_t)$  governs the transition from an odd state to the next even state.

As a result of the above definition, the even-to-even transition is summarized by

$$P_F(s_{t+1}|s_t) = \sum_{a_t} P(s_t \rightarrow s_{t+1} | s_t, a_t) \pi(a_t | s_t). \quad (80)$$

Note that the detailed balance condition, which involves a backward transition  $P_B$ , will also be decomposed in two parts: (1) for inverting the even-to-odd transition,

$$P_B(s_t | (s_t, a_t)) = 1 \quad (81)$$

by definition, and (2) for inverting the odd-to-even transition, we have to actually represent (and learn)

$$P_B((s_t, a_t) | s_{t+1}). \quad (82)$$

This conditional distribution incorporates the preference we may have over different paths leading to the same state while consistent with the environment  $P(s_t \rightarrow s_{t+1} | s_t, a_t)$ . The normalization constraint on  $\hat{P}_B$  can guarantee flow-matching via detailed balance, as argued around Eq. 33.

## 5.1 Known Deterministic Environments

A deterministic environment is perfectly controllable: we can choose the action that leads to the most desired next state, among the valid actions from the previous state. In the case where the environment is deterministic, we can directly apply the results of Sec. 3, as follows. At each time step  $t$  and from state  $s_t$ , the agent picks an allowed action  $a_t \in \mathcal{A}(s_t)$  according to a policy  $\pi(a_t | s_t)$ . The set of allowed actions should coincide with those actions for which  $\pi(a_t | s_t) > 0$ . Since the environment is deterministic and known, there is a deterministic function  $T : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$  which gives us the next state  $s_{t+1} = T(s_t, a_t)$ . In that case, we can ignore the even/odd state distinction and identify the learnable policy  $\pi$  with the learnable transition probability function  $P_F(s_{t+1} | s_t)$  of GFlowNets, as follows.

**Proposition 23.** *In a deterministic environment and a GFlowNet agent with policy  $\pi(a_t | s_t)$  and state transitions given by  $s_{t+1} = T(s_t, a_t)$ , the transition probability  $P_F(s_{t+1} | s_t)$  is given by*

$$P_F(s_{t+1} | s_t) = \sum_{a: T(s_t, a) = s_{t+1}} \pi(a | s_t). \quad (83)$$

Hence if only one action  $a_t$  can transition from  $s_t$  to  $s_{t+1} = T(s_t, a_t)$ , then

$$P_F(s_{t+1} | s_t) = \pi(a_t | s_t). \quad (84)$$

*Proof.* The result is obtained by marginalizing over  $a$ :

$$\begin{aligned} P_F(s_{t+1}|s_t) &= \sum_a P(s_t \rightarrow s_{t+1}, a_t | s_t) \\ &= \sum_a P(s_t \rightarrow s_{t+1} | s_t, a_t) \pi(a | s_t) \\ &= \sum_{a: s_{t+1}=T(s_t, a)} \pi(a | s_t) \end{aligned}$$

with  $P(s_t \rightarrow s_{t+1} | s_t, a_t) = 1_{s_{t+1}=T(s_t, a)}$ .

The case with a single possible action to obtain the transition is obtained because the sum contains only one term.  $\square$

**Proposition 24.** *In a deterministic environment with  $s_{t+1} = T(s_t, a_t)$ , a backwards transition probability function can be derived from a backwards policy  $\pi_B$ ,*

$$P_B(s_t | s_{t+1}) = \sum_{a: T(s_t, a) = s_{t+1}} \pi_B(a | s_{t+1}) \quad (85)$$

and in the case where a single action  $a_t$  explains each transition  $s_{t+1} = T(s_t, a_t)$ ,

$$P_B(s_t | s_{t+1}) = \pi_B(a_t | s_{t+1}) \quad (86)$$

*Proof.* The proof goes along exactly the same lines as for Prop. 23.  $\square$

## 5.2 Backwards Transitions can be Chosen Freely

Consider the setting in which we are given a reward function  $R$  to be matched, the environment is deterministic and known, which is not uncommon when the GFlowNet actions are internal (cognitive actions), used to produce a higher-level composed decision, such as to generate molecules in the GFlowNet paper (Bengio et al., 2021). In that case, Corollary 1 tells us that in order to fully determine the policy and the state or state-action flows, it is not sufficient in general to specify only the reward function  $R$ ; it is also necessary to specify the backwards transition probabilities on the edges other than the terminal ones (the latter being given by  $R$ ).

What this means is that the reward function does 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  $P_B(s | s_f)$  which is a function of  $R(s)$  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  $s$ .

**Definition 36.** *We say the backward transition probabilities  $P_B$  can be **chosen freely** if it is subject only to the constraints below and no others:*

1.  $P_B(s|s_f) = \frac{R(s)}{Z}$  (the terminal rewards),
2.  $P_B(s|s') = 0$  if  $s \rightarrow s' \notin \mathcal{A}$  (the constraints imposed by the DAG induced by the partial ordering  $\leq$ ),
3. any constraints implied by (1) and (2).

### 5.3 Unknown Deterministic Environments

If the environment is deterministic but unknown, we have to learn the transition function  $T$  and we should also learn its inverse  $T^{-1}$  which recovers the previous state given the next state and the action:

$$T^{-1} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S} \quad \text{s.t.} \quad T^{-1}(T(s, a), a) = s. \quad (87)$$

Unfortunately, if the state and action spaces are discrete and in high dimension, learning  $T$  and  $T^{-1}$  in a way that generalizes to unseen transitions<sup>3</sup> may be difficult and might be more easily achievable via a continuous relaxation. The methods for stochastic environments could be used for this purpose.

### 5.4 Stochastic Environments

The setting of stochastic environments is less straightforward but more general. We will decompose the transition as per Eq. 80 but not assume that  $P(s_t \rightarrow s_{t+1} | s_t, a_t)$  is a dirac. The first thing to note is that we can still obtain a Markovian flow, but that we are not guaranteed to find a policy which matches the desired terminal reward function.

**Proposition 25.** *In a stochastic environment with environment transitions  $P(s_t \rightarrow s_{t+1} | s_t, a_t)$ , any policy  $\pi(a|s)$  can yield a Markovian flow and it may not be possible to perfectly achieve desired flows  $\hat{F}(s \rightarrow s_f) = R(s)$ .*

*Proof.* We obtain a flow by satisfying the flow-matching or detailed balance equations for both even and odd steps, which can always be done for the following reason. From the even states, we can define an edge flow

$$\hat{F}(s_t \rightarrow (s_t, a_t)) = \hat{F}(s_t) \pi(a_t | s_t)$$

and the backwards transition is  $\pi_B(s_t | (s_t, a_t)) = 1$ . This leads to the intermediate state flow

$$\hat{F}((s_t, a_t)) = \hat{F}(s_t \rightarrow (s_t, a_t))$$

since there is only one edge into  $(s_t, a_t)$ , the one starting at  $s_t$  and taking action  $a_t$ . From the odd states, we have the edge flow

$$\begin{aligned} \hat{F}((s_t, a_t) \rightarrow s_{t+1}) &= \hat{F}((s_t, a_t)) P(s_t \rightarrow s_{t+1} | s_t, a_t) \\ &= \hat{F}(s_t) \pi(a_t | s_t) P(s_t \rightarrow s_{t+1} | s_t, a_t) \end{aligned}$$

---

<sup>3</sup>Seen transitions can just be recorded in a table, but in a combinatorial state-space, they will form an exponentially tiny fraction of the ones to be encountered in the future.

with  $P(s_t \rightarrow s_{t+1} | s_t, a_t)$  representing the environment, and we obtain the even state flow with the usual formula (Eq. 21)

$$\hat{F}(s_{t+1}) = \sum_{(s_t, a_t)} \hat{F}((s_t, a_t) \rightarrow s_{t+1}) = \sum_{(s_t, a_t)} \hat{F}(s_t) \pi(a_t | s_t) P(s_t \rightarrow s_{t+1} | s_t, a_t).$$

If unknown, the environment transitions  $P(s_t \rightarrow s_{t+1} | s_t, a_t)$  can be estimated in the usual supervised way by observing the triplets  $(s_t, a_t, s_{t+1})$  and estimating transition probabilities that approximately maximize the empirical log-likelihood of these observations. However, whereas in a stationary environment the transitions  $P(s_t \rightarrow s_{t+1} | s_t, a_t)$  do not depend on the policy, the backwards transitions  $\hat{P}_B((s_t, a_t) | s_{t+1})$  depend on the forward environment transition and on the state flow, i.e., on the policy. With enough training time and capacity, the forward and backward transitions can be made compatible, but as usual with GFlowNets, in a realistic settings the flow matching equations will not be perfectly achieved.

If there is enough capacity and training time (training to completion), we thus obtain a flow. Then, defining the transition probabilities by the sequential sampling of transitions from the even and odd steps above, we obtain a Markovian flow (Prop. 5).

To show that the desired terminal flows are not necessarily achievable, it is sufficient to identify a counter-example. Consider a terminal reward  $R(s) > 0$  while the environment transitions into  $s$  have zero probability. In that case, no matter how we choose our policy, we cannot put the desired flow into state  $s$ .  $\square$

Keep also in mind that in practice, even in a completely controllable environment, we will not be guaranteed to find a flow that matches the target terminal reward function simply because of finite capacity and finite training time for the GFlowNet.

Whereas with a deterministic environment for the GFlowNet, one can freely choose  $P_B$  for non-terminal edges, it is not so for stochastic environments, as argued below. On even-to-odd transitions,  $P_B(s | s_t, a_t) = 1_{s=s_t}$  by construction. On odd-to-even transitions  $(s_t, a_t) \rightarrow s_{t+1}$  the problem is that the forward transition is not a free parameter (it corresponds to the environment's  $P(s_t \rightarrow s_{t+1} | s_t, a_t)$ ).

**Counterexample 1.** *With a GFlowNet with a fixed stochastic environment, it may not be possible for  $P_B(s_t, a_t | s_{t+1})$  to be chosen freely while also matching the flows and the terminal rewards.*

*For a counterexample, consider the setting in Fig. 3. Suppose  $R(s''') \neq 0$  and the environment-provided transition  $T(s''' | s'', a'') = 0$ . Then, in order to match the terminal reward  $R(s''')$ , we must require that  $P_B(s', a' | s''') \neq 0$ , which means that  $P_B$  cannot be chosen freely.*

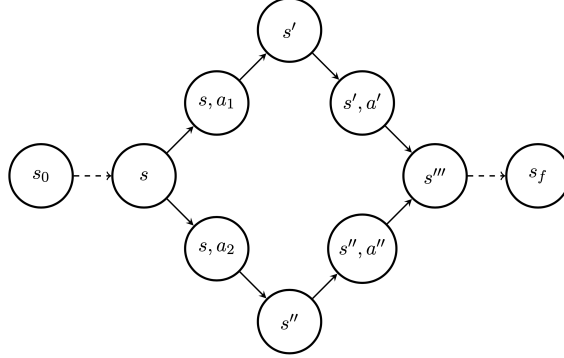


Figure 3: Consider a simple counter-example with only two paths from  $s$  to  $s'''$ , with a given  $R(s''') > 0$ . One path goes through  $s'$  and the other through  $s''$ . From  $s'$  the only feasible action  $a'$  leads to  $s'''$  and similarly from  $s''$  with action  $a''$ . However, it may be that the environment probability  $P(s'''|(s'', a'')) = 0$ , constraining  $P_B((s'', a'')|s''') = 0$ . Therefore it may not be possible to choose the backward transitions freely while matching the flows and terminal rewards.

## 6 Expected Reward and Reward-Maximizing Policy

We have already introduced the probability distribution  $P_T(s)$  and conditional probabilities  $P_T(s'|s \leq s')$  over terminating states (the states visited just before exiting into  $s_f$ ). More generally, one can consider any distribution  $P_\pi(s)$  over terminating states arising from some arbitrary choice of GFlowNet policy  $\pi$  and compute the expected reward under this distribution:

**Definition 37.** *The **expected reward** after visiting state  $s$  of a flow with terminal reward function  $R$ , under some distribution over terminating states  $P$ , is*

$$V_{P_\pi}(s) := E_{P_\pi(s)}[R(S)|S \geq s] = \sum_{s' \geq s} R(s')P_\pi(s'|s \leq s'). \quad (88)$$

**Proposition 26.** *When the probability distribution over terminating states is  $P_T$  given by the flow (see Def. 20), the expected reward under  $P_T$  is*

$$V_{P_T}(s) = \frac{\sum_{s' \geq s} R(s')^2}{\sum_{s' \geq s} R(s')}. \quad (89)$$

*Proof.* We apply the definition of conditional  $P_T$  (Eq. 68) to Eq. 88 and obtain the result.  $\square$

While we have a simple expression of the expected reward under  $P_T$ , the expected reward is defined more broadly for the distribution  $P_\pi$  arising from any

policy  $\pi$ . In particular for any policy  $\pi(a|s)$ , we can also define the expected reward  $V_{P_\pi}$  under the distribution  $P_\pi$  over terminating states induced by  $\pi$ . Expected rewards play a role similar to the state and state-action value functions in reinforcement learning, and as a consequence they also satisfy an equivalent of the policy improvement theorem when intermediate rewards are 0 and the discount factor  $\gamma = 1$ :

**Proposition 27.** *Let  $P_\pi$  be a distribution over terminating states arising from a policy  $\pi$ , and  $\bar{\pi}$  a greedy policy under the expected reward  $V_{P_\pi}$ , i.e.,*

$$\begin{aligned} \bar{\pi}(a|s) &= 0 \quad \text{unless} \\ V_{P_\pi}((s, a)) &\geq V_{P_\pi}((s, a')) \quad \forall a'. \end{aligned} \tag{90}$$

Then for all  $s$

$$V_{P_{\bar{\pi}}}(s) \geq V_{P_\pi}(s). \tag{91}$$

That is, the expected reward under the probability induced by  $\bar{\pi}$  is no worse than the one induced by  $\pi$ .

*Proof.* Let us denote by  $\bar{\pi}(s)$  the action deterministically chosen by greedy policy  $\bar{\pi}$  from  $s$ , and  $s_n$  the stochastically sampled terminating state. Then:

$$\begin{aligned} V_{P_\pi}(s) &\leq V_{P_\pi}((s, \bar{\pi}(s))) \\ &= E_{\bar{\pi}}[V_{P_\pi}(s_{t+1})|s_t = s] \\ &\leq E_{\bar{\pi}}[V_{P_\pi}(s_{t+1}, \bar{\pi}(s_{t+1}))|s_t = s] \\ &= E_{\bar{\pi}}[E_{\bar{\pi}}[V_{P_\pi}(s_{t+2})|s_t = s, s_t = s]] \\ &= E_{\bar{\pi}}[V_{P_\pi}(s_{t+2})|s_t = s] \\ &\dots \\ &\leq E_{\bar{\pi}}[V_{P_\pi}(s_n)|s_t = s] \\ &= E_{\bar{\pi}}[R(s_n)|s_t = s] \\ &= V_{P_{\bar{\pi}}}(s) \end{aligned}$$

where we have used the fact that, for all  $s'$ ,  $V_{P_\pi}(s') \leq V_{P_\pi}(s', \bar{\pi}(s'))$  since  $\bar{\pi}$  is a greedy policy. □

An immediate consequence is the following:

**Corollary 5.** *There exists a policy  $\pi^*(a|s)$  that maximizes the expected reward for all states  $s$ , namely the greedy policy of Prop. 27 associated with the GFlowNet's policy  $\pi$  yielding terminal distribution  $P_T(s) = R(s)/Z$ .*

How do we estimate the expected reward under  $P_T$ ? We just need to train another flow (or set of heads for a GFlowNet, see Sec. 8) with  $R^2$  as the reward function.



**Proposition 28.** *Consider two flows  $F$  and  $F'$ , one matching terminal reward function  $R$  and the other matching terminal reward function  $R^2$ . Then the expected reward under  $P_T$  (the distribution over terminating states defined by the flow  $F$ ) is*

$$V_{P_T}(s) = \frac{F'(s|s)}{F(s|s)}. \quad (92)$$

*Proof.* We start from Eq. 89 of the above corollary and notice that the numerator is the self-flow for  $F'$  while the denominator is the self-flow for  $F$  (see Eq. 64).  $\square$

## 6.1 Preference for High-Reward Early Trajectory

We have seen in Sec. 5.2 that by imposing a particular preference on  $P_B$ , one can make the GFlowNet sampling mechanism prefer to construct states in some orders more than others, e.g., one could prefer to start with states with larger expected reward (over their potential continuations), using Def. 37. It suffices to define  $P_B(s_t, a_t | s_{t+1})$  so that it puts more probability mass on state-action pairs  $(s_t, a_t)$  with larger  $V((s_t, a_t))$ .

## 7 Intermediate Rewards and Trajectory Returns

Up to now and in the GFlowNet paper (Bengio et al., 2021), we have considered terminal rewards as events happening only once per trajectory, at its end. Consider instead an agent experiencing a complete trajectory  $\tau$  and declare its **return** to be the sum of some intermediate environment rewards associated with all the transitions into the sink node from each of the visited states.

**Definition 38.** *The **trajectory return**  $\rho(\tau)$  associated with a partial trajectory  $\tau = (s_i, s_{i+1} \dots, s_n, s_f)$  is defined as*

$$\rho(s_i, s_{i+1}, \dots, s_n, s_f) = \sum_{t=i}^n R(s_t) = \sum_{t=i}^n F(s_t \rightarrow s_f) \quad (93)$$

$$\rho(s_f) = 0 \quad (94)$$

and the **expected future return**  $\bar{\rho}(s_t)$  associated with a state  $s_t$  is defined as

$$\begin{aligned} \bar{\rho}(s_t) &= E[\rho(s_t, s_{t+1}, s_{t+2}, \dots, s_n, s_f) | s_t] \\ &= \sum_{s_{t+1}, \dots, s_n} P(s_{t+1}, \dots, s_n, s_f | s_t) \rho(s_t, s_{t+1}, \dots, s_f) \end{aligned} \quad (95)$$

where the expectation is defined under the flow’s probability measure over trajectories (conditioned on the trajectory going through  $s_t$ ).

**Proposition 29.** *The expected future return  $\bar{\rho}(s)$  achievable from trajectories starting at  $s$  satisfies the following recursion:*

$$\bar{\rho}(s) = R(s) + \sum_{s' \in \text{Child}(s)} P_F(s' | s) \bar{\rho}(s'). \quad (96)$$

*Proof.* From the definition of return (Def. 38), we obtain the following:

$$\begin{aligned}
\bar{\rho}(s_t) &= \sum_{s_{t+1}, s_{t+2}, \dots} P(s_{t+1}, \dots, s_f | s_t) (R(s_t) + \rho(s_{t+1}, s_{t+2}, \dots, s_n)) \\
&= R(s_t) + \sum_{s_{t+1}} P_F(s_{t+1} | s_t) \sum_{s_{t+2}, \dots, s_n} P(s_{t+2}, \dots, s_n | s_{t+1}) \rho(s_{t+1}, \dots, s_n) \\
&= R(s_t) + \sum_{s_{t+1}} P_F(s_{t+1} | s_t) E[\rho(s_{t+1}, s_{t+2}, \dots) | s_{t+1}] \\
&= R(s_t) + \sum_{s_{t+1}} P_F(s_{t+1} | s_t) \bar{\rho}(s_{t+1}).
\end{aligned}$$

□

One reason why the above recursion is interesting is that it corresponds to the Bellman equation (Sutton and Barto, 2018) for the value function (which is the expected downstream return) in the case of no discounting (with an episodic setting). It is interesting to compare it with one of the equations we obtain for the state flow (Eq. 21):

$$\begin{aligned}
F(s_t) &= \sum_{s_{t+1}} F(s_{t+1}) P_B(s_t | s_{t+1}) \\
&= R(s_t) + \sum_{s_{t+1} \neq s_f} P_B(s_t | s_{t+1}) F(s_{t+1})
\end{aligned}$$

The two recursions are different: one uses the forward transition to propagate values backward, while the other uses the backward transitions to propagate flows.

**Definition 39.** Let us denote  $r(s)$  the possibly stochastic **environment reward**, provided when an agent visits state  $s$  (and generally distinct from the GFlowNet terminal reward at  $s$ ), and consider the environment reward accumulated in the partial trajectory  $\tau = (s_0, s_1, \dots, s_n)$  leading to state  $s_n = s$ . Let us call the GFlowNet state **return-augmented** if the state  $s$  includes the **accumulated reward**  $\nu(s)$ , i.e., there exists a function  $\nu(s) = \sum_{t=0}^n r(s_t)$ . Let us call a GFlowNet with a return-augmented state a **return-augmented GFlowNet**.

We want to keep track of accumulated intermediate rewards in the GFlowNet state to compute the terminal reward from the state, and thus train GFlowNets to sample in proportion to the accumulated reward:

**Proposition 30.** Suppose  $G$  is a return-augmented GFlowNet with target terminal reward function  $R(s)$  equal to the accumulated environment reward  $\nu(s)$ . Furthermore, suppose  $G$  is trained to completion. Then sampling from  $G$  produces accumulated reward  $\nu$  with probability proportional to  $\nu$ .

*Proof.* Since  $G$  is a GFlowNet trained to completion with terminal reward function  $R(s) = \nu(s)$ , we know that sampling from  $G$  samples terminal accumulated rewards  $\nu(s)$  with probability proportional to  $\nu$  by Prop. 11.  $\square$

Note that such a GFlowNet can only be trained offline, i.e., using trajectories which have terminated and for which we have observed the return.

Note also that if we did not augment the state with the accumulated reward, then the GFlowNet terminal reward would not be a function of the state. Having a return-augmented state also makes it possible to handle stochastic environment rewards in the GFlowNet framework (but see also the use of the MSE reward-matching loss, Eq. 44).

## 8 Multi-Flows, Distributional GFlowNets, Unsupervised GFlowNets and Pareto GFlowNets

Consider an environment with stochastic rewards. As with Def. 39, we could augment the state to include the random accumulated rewards, thus (by Prop. 30) making the GFlowNet sample trajectories with returns  $\rho$  occurring with probability proportional to  $\rho$ . However, similarly to Distributional RL (Bellemare et al., 2017), it could be interesting to generalize GFlowNets to capture not just the expected value of achievable terminal rewards but also other statistics of its distribution. More generally, we can think of this like a family of GFlowNets, each of which models in its flow a particular future environmental outcome of interest. With the particle analogy of GFlowNets, it would be as if the particles had a colour or label (just like the frequency of each photon in a group of photons travelling together in a beam of light) and that we separately account for the flows associated with all the possible label types.

If the number of outcomes (the number of possible labels) is small, this could be implemented with different output heads of the GFlowNet (e.g., one output for the flow associated with each label). When a trajectory associated with a particular label outcome is observed, the corresponding heads get gradients. A more powerful and general implementation puts the outcome event as an input of the GFlowNet, thus amounting to training a conditional GFlowNet (see Sec. 4.1), and formalized below.

**Definition 40.** *Let us define the **outcome**  $y = f(s)$  as a known function  $f(s)$  of the state  $s$ . An outcome can be whether an environment reward takes a particular value, or it can be a vector of important features of  $s$  which are sufficient to determine many possible environment reward functions (in particular,  $f$  can be the identity function). Let us call the conditional GFlowNet taking  $y$  as conditioning input, with conditional flows  $F(A|y)$  for events  $A$  over the trajectories consistent with reward function*

$$R_y(s) = 1_{f(s)=y} \tag{97}$$

*an **outcome-conditioned GFlowNet** with outcome function  $f$ .*

We will limit ourselves here to a discrete set of outcomes for simplicity but expect that this approach can be generalized to a continuous set. Note how, if the outcome-conditioned GFlowNet is trained to completion, it makes it possible to only sample terminating states  $s$  yielding the chosen outcome  $y$ . In principle, this allows sampling objects guaranteed to have a high reward (under the given reward function). In practice, a GFlowNet will never be perfectly trained to completion, and we should think of such an outcome-conditioned GFlowNet similarly to a goal-conditioned policy in RL (Ghosh et al., 2018) or the reward-conditioned upside-down RL (Schmidhuber, 2019). An interesting question for future work is to extend these outcome-conditioned GFlowNets to the case of stochastic rewards or stochastic environments.

**Definition 41.** *A **distributional GFlowNet** is an outcome-conditioned GFlowNet taking as conditioning input the value of the environment return associated with complete trajectories. This can be achieved by making the GFlowNet return-augmented, so that the return can be read from the terminating state of the trajectory.*

Training an outcome-conditioned GFlowNet can only be done offline because the conditioning input (e.g., the final return) may only be known after the trajectory has been sampled. A reasonable contrastive training procedure could thus proceed as follows:

1. Sample a trajectory  $\tau^+$  according to an unconditional training policy  $\pi_T$ .
2. Obtain the outcome  $y^+ = f(s^+)$  from the terminating state  $s^+$  (occurring just before the sink state  $s_f$  in  $\tau^+$ ).
3. Update the conditional GFlowNet with  $\tau^+$  and target terminating reward  $R(s^+|y^+) = 1_{f(s^+)=y^+} = 1$ .
4. Sample a trajectory  $\tau^-$  according to the conditional GFlowNet policy with condition  $y^+$ .
5. Obtain the actual outcome  $y^- = f(s^-)$  for the terminating state  $s^-$  (occurring just before the sink state  $s_f$ ) in  $\tau^-$ . If the GFlowNet was perfectly trained, we should have  $y^+ = y^-$  but otherwise, especially if the number of possible outcomes is large, this becomes unlikely.
6. Update the conditional GFlowNet using a flow-matching loss with trajectory  $\tau^-$  and target terminating reward  $R(s^-|y) = 1_{y^-=y^+}$  (likely to be 0 if there are many possible values for  $y$ ).

An interesting question for future work is to consider a smoother reward function instead of the sharp but sparse reward  $R(s|y) = 1_{f(s)=y}$  as conditional reward, in order to make training easier.

## 8.1 Defining a reward function a posteriori

The reward function may not be known a priori or it may be known only up to some unknown constants (e.g., defining a Pareto front) or we may wish to generalize GFlowNets so they can be trained or pre-trained in a more unsupervised way, with the specific generative task only specified afterwards. The following important proposition allows us to do just that, and convert an outcome-conditioned GFlowNet into one that samples according to a given reward function, without having to retrain the network.

**Proposition 31.** *Consider an outcome-conditioned GFlowNet trained to completion with respect to the possible outcomes  $y = f(s)$  over terminating states  $s$  and a terminal reward function  $R(s) = r(f(s))$  given a posteriori (possibly after training the GFlowNet) as a function  $r(y)$  of the outcome  $y = f(s)$ . Then a GFlowNet with flow  $F_{r \circ f}(A)$  over events  $A$  which matches target terminal reward function  $R = r \circ f$  can be obtained from the flow  $F(A|y)$  of the outcome-conditioned GFlowNet via*

$$F_{r \circ f}(A) = \sum_y r(y) F(A|y). \quad (98)$$

For example, the GFlowNet policy  $\pi_{r \circ f}(a|s)$  for terminal reward  $R = r \circ f$  can be obtained from

$$\pi_{r \circ f}(a|s) = \frac{\sum_y r(y) F((s, a)|y)}{\sum_y r(y) F(s|y)} \quad (99)$$

or

$$\pi_{r \circ f}(a|s) = \frac{\sum_y r(y) F(s|y) \pi(a|s, y)}{\sum_y r(y) F(s|y)} \quad (100)$$

where  $F(s|y)$ ,  $F((s, a)|y)$  and  $\pi(a|s, y)$  are the outcome-conditioned state flow, state-action flow and action policy respectively.

*Proof.* We first clarify what  $F_{r \circ f}(A)$  means:

$$F_{r \circ f}(A) = \sum_s r(f(s)) P(A|s \rightarrow s_f) = \sum_s r(f(s)) P_B(A|s \rightarrow s_f) \quad (101)$$

where  $P(A|s \rightarrow s_f)$  is the probability of event  $A$  (e.g., a particular state or transition) among the trajectories that end in terminal transition  $s \rightarrow s_f$ , and it can be determined entirely by  $P_B$  (considering all the backward paths starting at  $s$  and going back to the particular state or transition  $A$ ), as shown by Corollary 1. This means that  $P(A|s \rightarrow s_f)$  does not depend of the choice of reward function, so Eq. 101 is valid for any  $r$ .

Clearly, we see that Eq. 98 works in the extreme case where  $r(y') = 1_{y=y'}$  is an indicator function at some specific  $y$  value since the sum in Eq. 98 reduces to  $F(A|y)$  which corresponds to reward function  $R_y$  as per Eq. 97. This yields

$$F_{1_y}(A) = F(A|y) = \sum_s 1_{f(s)=y} P(A|s \rightarrow s_f) \quad (102)$$

where  $1_y$  denotes the function that, given  $s$ , returns  $1_{y=s}$ .

To complete the proof let us start with the right-hand side of Eq. 98 and insert the above definition of  $F(A|y)$  (Eq. 102), then swap the sums and use the indicator function to cancel the sum over  $y$ , and finally apply the definition of  $F_{r \circ f}(A)$  in Eq. 101:

$$\begin{aligned}
\sum_y r(y)F(A|y) &= \sum_y r(y) \sum_s 1_{f(s)=y} P(A|s \rightarrow s_f) \\
&= \sum_s \sum_y r(y) 1_{f(s)=y} P(A|s \rightarrow s_f) \\
&= \sum_s r(f(s)) P(A|s \rightarrow s_f) \\
&= F_{r \circ f}(A)
\end{aligned} \tag{103}$$

recovering the left-hand side of Eq. 98 as desired.  $\square$

This makes it possible to predict probabilities and perform sampling actions for all the possible outcomes  $y$  arising in different states (in the extreme where we know nothing about possible reward functions and the outcome is  $y = s$ ), and then convert that GFlowNet on the fly to one specialized to a given terminal reward function  $R = r \circ f$ . However, we note that it requires more computation for each action at run-time: we have to perform these sums (possibly via Monte-Carlo integration) over the outcome space, and there may be a computational time versus accuracy trade-off in the resulting decisions (based on how many Monte-Carlo samples are used to approximate the above sums).

## 8.2 Pareto GFlowNets

A related application of these ideas concerns Pareto optimization, where we are not sure about the correct reward function up to a few coefficients forming a convex combination of underlying objectives.

**Definition 42.** *The Pareto additive terminal reward functions can be written as*

$$R_\omega(s) = \sum_i \omega_i f_i(s) \tag{104}$$

where  $\omega \in \{\omega \in W \subset \mathbb{R}^d : \omega_i \geq 0, \sum_i \omega_i = 1\}$  are convex weights and the outcomes of interest are  $d$  objectives  $y_i = f_i(s)$ , and  $W$  is a discrete set of convex weights.

**Definition 43.** *The Pareto multiplicative terminal reward functions can be written as*

$$R_\omega(s) = e^{-\sum_i \omega_i e_i(s)} \tag{105}$$

where  $\omega \in \{\omega \in W \subset \mathbb{R}^d : \omega_i \geq 0, \sum_i \omega_i = 1\}$  are convex weights and the outcomes of interest are  $d$  objectives  $y_i = f_i(s)$ , and  $W$  is a discrete set of convex weights.

In these cases, we can train a conditional GFlowNet with  $\omega$  as conditioning input and  $R_\omega(s) = R(s|\omega)$  as conditional terminal reward function. At runtime, we can scan the set of  $\omega$ 's in order to obtain different policies or predicted probabilities or free energies. The above can easily be generalized to a non-convex and non-linear combination of the objectives, so long as the combined objective is parametrized by  $\omega$ . Note the similarity between this idea (and more generally outcome-conditioned GFlowNets) and the earlier work by Dosovitskiy and Djolonga (2019). The same idea of conditioning by a form of specification of the loss can be applied to GFlowNets to obtain a family of GFlowNets, one for each variant of the loss function.

A useful application of a Pareto GFlowNet with such reward functions is to draw samples from the Pareto frontier. Once the Pareto GFlowNet is trained, we can draw samples from the Pareto frontier by first sampling the convex weights  $\omega$  and then sampling trajectories. This can be useful in multi-objective optimization or sampling, where we want to draw a diversity of solutions corresponding to different trade-off points of the various objectives.

## 9 GFlowNets on Sets, Graphs, and to Marginalize Joint Distributions

### 9.1 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 44.** *The states of a set GFlowNets are subsets  $s$  of a set  $\mathcal{U}$ , i.e.,  $s \in 2^{\mathcal{U}}$ . The unconditional initial state is the empty set  $s_0 = \{\}$  and the action space consists of actions adding an element  $a \in \mathcal{U}$  to the current set  $s$ . The only allowed actions from state  $s$  are those  $a \in \mathcal{U} \setminus s$  not already in  $s$ . If the environment is deterministic and the action  $a_t$  is allowed from  $s_t$ , then the environment transition probability is  $P(s_t \rightarrow s_{t+1} | s_t, a_t) = 1_{s_{t+1} = s_t \cup \{a_t\}}$ , or equivalently, the transition function is  $T(s, a) = s \cup \{a\}$ . For the set GFlowNets to be well-defined, it must be given a terminal reward function  $R$  such that the partition function exists, i.e.,*

$$Z = \sum_{s \in 2^{\mathcal{U}}} R(s) < \infty. \quad (106)$$

For the rest of this paper, we assume that all set GFlowNets are well-defined. A set GFlowNet defines a terminating probability distribution  $P_T$  on states (see Def. 20 and Eq. 53), with

$$P_T(s) = e^{-\mathcal{E}(s) + \mathcal{F}(s_0)} = \frac{F(s \rightarrow s_f)}{F(s_0)} \quad (107)$$

when flows and terminal rewards are matched and  $\mathcal{E}$  represents the energy (i.e.,  $-\log R$ ). Similarly, Prop. 20 provides us with a formula for conditional probabilities of a given superset  $s'$  of a given set  $s$  under  $P_T$ ,

$$P_T(s'|s' \supseteq s) := e^{-\mathcal{E}(s') + \mathcal{F}(s)} = \frac{F(s' \rightarrow s_f)}{F(s|s)} \quad (108)$$

where  $\mathcal{F}$  indicates free energy (see Def. 24 and Prop. 19).

Remember that with a GFlowNet with flow estimator  $\hat{F}$  that is not completely trained, it is not guaranteed that  $\hat{F}(s) = R(s)$ , so we could estimate probabilities with

$$\hat{P}_T(s) = \frac{\hat{F}(s \rightarrow s_f)}{\hat{F}(s_0)} \quad (109)$$

or alternatively

$$\hat{P}_T(s) = \frac{R(s)}{\hat{F}(s_0)}. \quad (110)$$

Similarly, we can estimate conditional superset probabilities with Eq. 108 or with

$$\hat{P}_T(s'|s' \supseteq s) = \frac{R(s')}{\hat{F}(s|s)}, \quad (111)$$

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 32.** *Let  $\mathfrak{S}(s) = \{s' \supseteq s\}$  be the set of all supersets of a set  $s$ . The probability of drawing an element from  $\mathfrak{S}$  with a set GFlowNet trained to completion is*

$$P_T(\mathfrak{S}(s)) = \sum_{s' \supseteq s} P_T(s') = \frac{e^{-\mathcal{F}(s)}}{Z} = \frac{F(s|s)}{F(s_0)}. \quad (112)$$

*Proof.* We can rewrite the sum as follows, first applying the definition of  $P_T$  (Eq. 39), and then the definition of  $R$  (Eq. 34) and Prop. 19 (Eq. 64) and finally  $Z = F(s_0)$  (Eq. 35):

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

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



To summarize, a GFlowNet which is trained to match a given energy function (and 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.

## 9.2 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 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. Since graphs are sets, all the GFlowNet operations on sets can be applied on graphs.

## 9.3 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  $X = (X_1, X_2, \dots, X_n)$  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  $(i, x_i)$ . 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 GFlowNet 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. 112). We can sample the other variables by setting the state at  $s$  and continuing to sample from the GFlowNet’s policy. We can sample 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.

## 9.4 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  $v^i = (v_1, v_2, \dots)$  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.,

$$\mathcal{E}_{F^i}(v^i) = \mathcal{E}_{F^i}(v_1, v_2, \dots). \quad (113)$$

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

$$\mathcal{E}(g) = \sum_i \mathcal{E}_{F^i}(v^i). \quad (114)$$

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. 4.3 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 the  $V_j = v_j$  for the observed variables) and denote  $h$  the specification 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. 4.3 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).

## 10 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. 41, such sums are also hiding in the detailed balance constraint of Eq. 45. Indeed, these sums are implicit as part of the normalizing constant in the conditional density of the next state or previous state in  $P_F(s_{t+1}|s_t)$  and  $P_B(s_t|s_{t+1})$ . We consider below ideas to deal with this challenge.

### 10.1 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

$$s = (s^i, s^x) \tag{115}$$

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

$$P_F(s_{t+1}|s_t) = P(s_{t+1}^x|s_{t+1}^i, s_t)P(s_{t+1}^i|s_t). \tag{116}$$

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  $P(s_{t+1}^x | s_{t+1}^i, s_t)$ ) 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  $s_{t+1}^x | s_{t+1}^i, s_t$  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).

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.

## 10.2 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. 4.3. 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 cor-

respond 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.

## 11 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 (Moćkus, 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.

### 11.1 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 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. 4.3. 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  $\pi_T$ ), 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. 3.6, among others).

## 11.2 Contrast with Regularized Reinforcement Learning

The flow-matching loss of GFlowNets 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). 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

$\pi^* \in \operatorname{argmax}_{\pi} V^{\pi}(s) \forall s$  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

$$p(\tau) = \left[ p(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t) \right] \exp \left( \eta \sum_{t=0}^{T-1} R(s_t, a_t) \right) \quad (117)$$

where  $\tau = (s_0, a_0, s_1, a_1, \dots, s_T)$  and  $\eta$  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  $\rho$  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*  $d^{\pi}$  (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.

### 11.3 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 work 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.

## 12 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. It also introduces alternatives to the flow-matching objective which may bypass the slow “bootstrapping” propagation of credit information from the end of the action sequence to its beginnings, with so-called direct credit assignment (which has some similarity to policy gradient). 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.

In an attempt to better discern the links between GFlowNets and the more common forms of RL, this paper also considers the cases where rewards are provided not just at the end but possibly after every action, and where the environment may be stochastic. It also shows how a greedy policy that maximises returns could be obtained from a trained GFlowNet.

Another interesting question is whether the reward function needs to be known ahead of time, whether it could be learned (since GFlowNets could approximately sample from a current energy function), and whether one could use these ideas to learn a GFlowNet in a more unsupervised way (before knowing what the precise reward function will be), a special case of which enables learning to sample from a Pareto front.

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.

## Acknowledgements

The authors want to acknowledge the useful suggestions and feedback on the paper and its ideas provided by Alexandra Volokhova, Marc Bellemare and Valentin Thomas. They are also grateful for the financial support from CIFAR, Samsung, IBM, Google, Microsoft, JP Morgan Chase, and the Thomas C. Nelson Stanford Interdisciplinary Graduate Fellowship.

This research was funded in part by JPMorgan Chase & Co. Any views or opinions expressed herein are solely those of the authors listed, and may differ from the views and opinions expressed by JPMorgan Chase & Co. or its affiliates. This material is not a product of the Research Department of J.P. Morgan Securities LLC. This material should not be construed as an individual recommendation for any particular client and is not intended as a recommendation of particular securities, financial instruments or strategies for a particular client. This material does not constitute a solicitation or offer in any jurisdiction.

## References

- Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. Machine learning, 50(1):5–43, 2003.
- Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. SIAM journal on computing, 32(1):48–77, 2002.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. ICLR’2015, arXiv:1409.0473, 2014.
- Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. CoRR, abs/1909.01377, 2019. URL <http://arxiv.org/abs/1909.01377>.
- Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In International Conference on Machine Learning, 2017.
- Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. NeurIPS’2021, arXiv:2106.04399, 2021.
- Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations. In International conference on machine learning, pages 552–560. PMLR, 2013.
- Nathan Brown, Ben McKay, François Gilardoni, and Johann 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.
- Lars Buesing, Nicolas Heess, and Theophane Weber. Approximate inference in discrete distributions with monte carlo tree search and value functions, 2019.
- Lawrence Cayton. Algorithms for manifold learning. Univ. of California at San Diego Tech. Rep, 12(1-17):1, 2005.
- Hanjun Dai, Rishabh Singh, Bo Dai, Charles Sutton, and Dale Schuurmans. Learning discrete energy-based models via auxiliary-variable local exploration. In Neural Information Processing Systems (NeurIPS), 2020.
- Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. ICLR’2015 Workshop, arXiv:1410.8516, 2014.
- Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. ICLR’2017, arXiv:1605.08803, 2016.

- Alexey Dosovitskiy and Josip Djolonga. You only train once: Loss-conditional training of deep networks. In International Conference on Learning Representations, 2019.
- Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. Journal of Machine Learning Research, 6:503–556, 2005.
- Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal-conditioned policies. arXiv preprint arXiv:1811.07819, 2018.
- Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. Advances in neural information processing systems, 27, 2014.
- Anirudh Goyal and Yoshua Bengio. Inductive biases for deep learning of higher-level cognition. arXiv, abs/2011.15091, 2020. <https://arxiv.org/abs/2011.15091>.
- Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. ICLR’2021, arXiv:1909.10893, 2019.
- Will Grathwohl, Kevin Swersky, Milad Hashemi, David Duvenaud, and Chris J. Maddison. Oops i took a gradient: Scalable sampling for discrete distributions, 2021.
- Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. Constrained bayesian optimization for automatic chemical design. arXiv preprint arXiv:1709.05501, 2017.
- Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In International Conference on Machine Learning, pages 1352–1361. PMLR, 2017.
- W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. Biometrika, 1970.
- Ajay Jasra, Chris C Holmes, and David A Stephens. Markov chain monte carlo methods and the label switching problem in bayesian mixture modeling. Statistical Science, pages 50–67, 2005.
- Jan 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.
- Diederik P Kingma and Max Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.

- Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. IEEE Transactions on information theory, 47(2): 498–519, 2001.
- Rithesh Kumar, Sherjil Ozair, Anirudh Goyal, Aaron Courville, and Yoshua Bengio. Maximum entropy generators for energy-based models, 2019.
- Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In International Conference on Machine Learning, pages 1945–1954. PMLR, 2017.
- Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In Reinforcement learning, pages 45–73. Springer, 2012.
- Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. arXiv preprint arXiv:1805.00909, 2018.
- Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. The journal of chemical physics, 21(6):1087–1092, 1953.
- Jonas Moćkus. 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](https://doi.org/10.1162/EVCO_a_00048).
- Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. arXiv preprint arXiv:1702.08892, 2017.
- Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. arXiv preprint arXiv:1906.04733, 2019.
- Hariharan Narayanan and Sanjoy Mitter. Sample complexity of testing the manifold hypothesis. In NIPS’2010, pages 1786–1794, 2010.
- Charlie Nash and Conor Durkan. Autoregressive energy machines. In International Conference on Machine Learning, pages 1735–1744. PMLR, 2019.
- Emilia Pompe, Chris Holmes, and Krzysztof Łatuszyński. A framework for adaptive mcmc targeting multimodal distributions. The Annals of Statistics, 48(5):2930–2952, 2020.
- Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In International conference on machine learning, pages 1530–1538. PMLR, 2015.

- Martin 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.
- Salah Rifai, Yann N Dauphin, Pascal Vincent, Yoshua Bengio, and Xavier Muller. The manifold tangent classifier. Advances in neural information processing systems, 24:2294–2302, 2011.
- Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.
- Juergen Schmidhuber. Reinforcement learning upside down: Don’t predict rewards—just map them to actions. arXiv preprint arXiv:1912.02875, 2019.
- Bernhard Schölkopf, Dominik Janzing, Jonas Peters, Eleni Sgouritsa, Kun Zhang, and Joris Mooij. On causal and anticausal learning. In ICML’2012, pages 1255–1262, 2012.
- Ari Seff, Wenda Zhou, Farhan Damani, Abigail Doyle, and Ryan P Adams. Discrete object generation with reversible inductive construction. arXiv preprint arXiv:1907.08268, 2019.
- Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In International Conference on Machine Learning (ICML), 2010.
- Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. MIT press, 2018.
- Kevin Swersky, Yulia Rubanova, David Dohan, and Kevin Murphy. Amortized bayesian optimization over discrete spaces. In Conference on Uncertainty in Artificial Intelligence, pages 769–778. PMLR, 2020.
- Marc Toussaint and Amos 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.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in neural information processing systems, pages 5998–6008, 2017.
- Junfeng Wen, Bo Dai, Lihong Li, and Dale Schuurmans. Batch stationary distribution estimation. arXiv preprint arXiv:2003.00722, 2020.
- Yutong Xie, Chence Shi, Hao Zhou, Yuwei Yang, Weinan Zhang, Yong Yu, and Lei Li. {MARS}: Markov molecular sampling for multi-objective drug discovery. In International Conference on Learning Representations, 2021. URL <https://openreview.net/forum?id=kHSu4ebxFXy>.

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