GFlowNets: A Novel Framework for Diverse Generation in Combinatorial and Continuous Spaces

MBZUAI Paris Workshop

Salem Lahlou, MBZUAI Abu Dhabi, salem.lahlou@mbzuai.ac.ae, https://la7.lu

13 February 2025

What's a GFlowNet?

What it's not:
a particular Neural Net architecture

What's a GFlowNet?

What it's not:
a particular Neural Net architecture

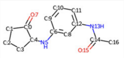
What it is:
A generative framework designed to sample combinatorial objects, with maximal diversity, based on an energy function or a reward function

- Since 2023: sample continuous things as well!

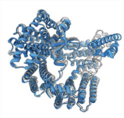
What it's not:
a particular Neural Net architecture

What it is:
A generative framework designed to sample combinatorial objects,
with maximal diversity, based on an energy function or a reward function

- Since 2023: sample continuous things as well!



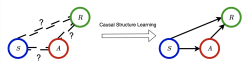
small molecules
(and their conformation)[2]



proteins[1]

Fine Tuning is fun for all!

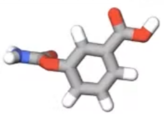
language



Causal/Bayesian Networks
(and their parameters)[3]

[1] "Scalable protein design using optimization in a relaxed sequence space", Frank et al., 2024
[2] "Molecular Graph Generation by Decomposition and Reassembling", Yamada and Sugiyama, 2023
[3] `https://causaldm.github.io/`

Motivation: Drug Discovery



Motivation: Molecular Search for COVID-19 Therapeutics

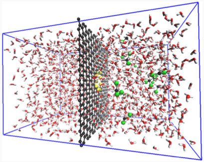
- Evaluation method: Physics-based simulation & molecular docking (noisy and expensive !)
- Challenges:
  - Many molecules appear promising in simulation
  - Good candidates are scattered across chemical space
- **Goal**: Select most promising candidates for laboratory testing

Motivation: Why is diversity Good?

Hidden blind spots

In organic chemistry (and many domains), the proxies we use are fundamentally imprecise

So we *must* maintain broad coverage, by **searching** comprehensively



Source: "Molecular dynamics simulation ...", Azamat et al. 2015

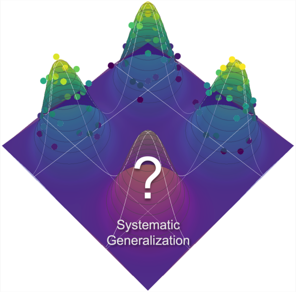
Molecular Dynamics can only do so much!

Motivation: Why is diversity Good?

Hidden blind spots

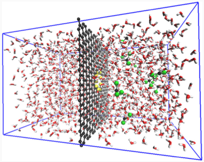
In organic chemistry (and many domains), the proxies we use are fundamentally imprecise

So we *must* maintain broad coverage, by **searching** comprehensively



Source: "GFlowNets for AI-Driven Scientific Discovery", Jai et al. 2023

If only one mode is correct, we'd rather try all 4 than 4 candidates in one mode



Source: "Molecular dynamics simulation ...", Azamat et al. 2015

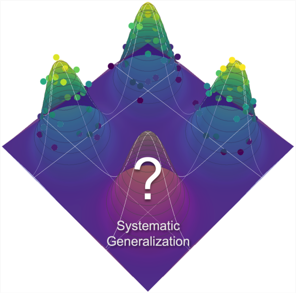
Molecular Dynamics can only do so much!

Motivation: Why is diversity Good?

Hidden blind spots

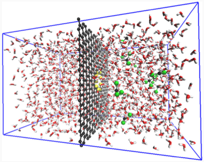
In organic chemistry (and many domains), the proxies we use are fundamentally imprecise

So we *must* maintain broad coverage, by **searching** comprehensively



Source: "GFlowNets for AI-Driven Scientific Discovery", Jai et al. 2023

If only one mode is correct, we'd rather try all 4 than 4 candidates in one mode



Source: "Molecular dynamics simulation ...", Azamat et al. 2015

Molecular Dynamics can only do so much!

→ **Systematic generalization**: With only 3 modes discovered, an ideal model should learn to generate data from the fourth mode.

Motivation: Existing Approaches

- **Reinforcement Learning**
  - Framework: MDP with actions and rewards
  - Challenge: Exploration remains a complex, unsolved problem
  - Result: Limited diversity in discovered solutions

Motivation: Existing Approaches

- **Reinforcement Learning**
  - Framework: MDP with actions and rewards
  - Challenge: Exploration remains a complex, unsolved problem
  - Result: Limited diversity in discovered solutions

Instead of maximizing reward, let's sample. We want a distribution $\pi(x)$ proportional to reward:

$$\pi(x) \approx \frac{R(x)}{Z} = \frac{R(x)}{\sum_{x' \in \mathcal{X}} R(x')}$$

Motivation: Existing Approaches

- **Reinforcement Learning**
  - Framework: MDP with actions and rewards
  - Challenge: Exploration remains a complex, unsolved problem
  - Result: Limited diversity in discovered solutions

Instead of maximizing reward, let's sample. We want a distribution $\pi(x)$ proportional to reward:

$$\pi(x) \approx \frac{R(x)}{Z} = \frac{R(x)}{\sum_{x' \in \mathcal{X}} R(x')}$$

- **Markov Chain Monte Carlo**
  - Challenge: Prohibitively slow mode mixing in practice

Motivation: Existing Approaches

- **Reinforcement Learning**
  - Framework: MDP with actions and rewards
  - Challenge: Exploration remains a complex, unsolved problem
  - Result: Limited diversity in discovered solutions

Instead of maximizing reward, let's sample. We want a distribution $\pi(x)$ proportional to reward:

$$\pi(x) \approx \frac{R(x)}{Z} = \frac{R(x)}{\sum_{x' \in \mathcal{X}} R(x')}$$

- **Markov Chain Monte Carlo**
  - Challenge: Prohibitively slow mode mixing in practice
- **Generative Models** (GANs/VAEs/Diffusion)
  - Limitation: Don't fully utilize scalar reward signals

- GFlowNets are a method for sampling from a desired distribution by learning a **flow** (to be defined a in a few slides) in a Directed Acyclic Graph (DAG).

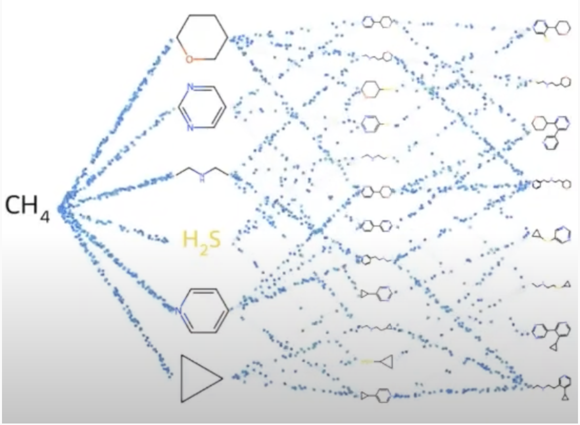


Figure from Emmanuel Bengio's tutorial at the Mila GFlowNet workshop, 2023

Introducing GFlowNets

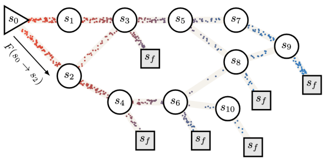
- GFlowNets are a method for sampling from a desired distribution by learning a **flow** (to be defined a in a few slides) in a Directed Acyclic Graph (DAG).
- Natural fit for **combinatorial spaces** (e.g., molecules, graphs, *natural language*).

Introducing GFlowNets

- GFlowNets are a method for sampling from a desired distribution by learning a **flow** (to be defined a in a few slides) in a Directed Acyclic Graph (DAG).
- Natural fit for **combinatorial spaces** (e.g., molecules, graphs, *natural language*).
- Specifically designed to address the limitations of other methods.

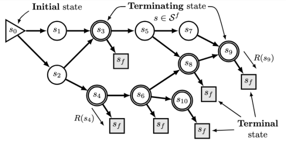
Introducing GFlowNets

- GFlowNets are a method for sampling from a desired distribution by learning a **flow** (to be defined a in a few slides) in a Directed Acyclic Graph (DAG).
- Natural fit for **combinatorial spaces** (e.g., molecules, graphs, *natural language*).
- Specifically designed to address the limitations of other methods.



Source: "GFlowNet Foundations", Bengio*, Lahlou*, Deleu* et al., JMLR 2023

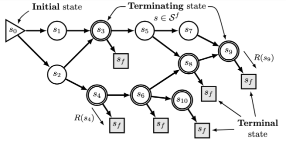
Notations and Problem Setting



Source: "GFlowNet Foundations", Bengio*, Lahlou*, Deleu* et al., JMLR 2023

- We'll be working with a Directed Acyclic Graph (DAG): $\mathcal{G} = (\mathcal{S}, \mathbb{A})$.

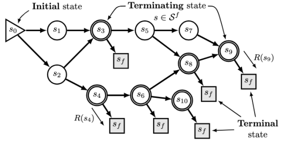
Notations and Problem Setting



Source: "GFlowNet Foundations", Bengio*, Lahlou*, Deleu* et al., JMLR 2023

- We'll be working with a Directed Acyclic Graph (DAG): $\mathcal{G} = (\mathcal{S}, \mathbb{A})$.
- $\mathcal{S}$ is the set of states, including special initial state s_0 and sink state s_f. (Warning: Some authors prefer not to use s_f. The math is equivalent.)

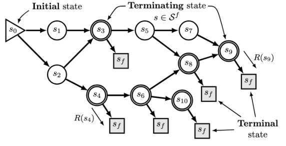
Notations and Problem Setting



Source: "GFlowNet Foundations", Bengio*, Lahlou*, Deleu* et al., JMLR 2023

- We'll be working with a Directed Acyclic Graph (DAG): $\mathcal{G} = (\mathcal{S}, \mathbb{A})$.
- $\mathcal{S}$ is the set of states, including special initial state s_0 and sink state s_f.
- $\mathcal{X} = \mathcal{S}^f \subseteq \mathcal{S}$ is the set of states we want to sample from, with a **given** reward function $R(s) > 0$ for each $s \in \mathcal{X}$.

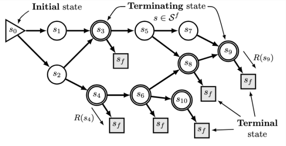
Notations and Problem Setting



Source: "GFlowNet Foundations", Bengio*, Lahlou*, Deleu* et al., JMLR 2023

- We'll be working with a Directed Acyclic Graph (DAG): $\mathcal{G} = (\mathcal{S}, \mathbb{A})$.
- $\mathcal{S}$ is the set of states, including special initial state s_0 and sink state s_f.
- $\mathcal{X} = \mathcal{S}^f \subseteq \mathcal{S}$ is the set of states we want to sample from, with a **given** reward function $R(s) > 0$ for each $s \in \mathcal{X}$.
- A **complete trajectory** τ is a path from s_0 to s_f. Denoted $\tau = (s_0 \to s_1 \to \ldots \to s_n \to s_{n+1} = s_f)$

Notations and Problem Setting

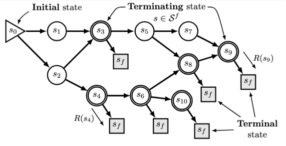


Source: "GFlowNet Foundations", Bengio*, Lahlou*, Deleu* et al., JMLR 2023

- We'll be working with a Directed Acyclic Graph (DAG): $\mathcal{G} = (\mathcal{S}, \mathbb{A})$.
- $\mathcal{S}$ is the set of states, including special initial state s_0 and sink state s_f.

- $\mathcal{X} = \mathcal{S}^f \subseteq \mathcal{S}$ is the set of states we want to sample from, with a **given** reward function $R(s) > 0$ for each $s \in \mathcal{X}$.
- A **complete trajectory** τ is a path from s_0 to s_f. Denoted $\tau = (s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s_{n+1} = s_f)$
- **Constructiveness assumption:** We can build states in $\mathcal{X}$ step-by-step, starting from s_0.

Notations and Problem Setting



Source: "GFlowNet Foundations", Bengio*, Lahlou*, Deleu* et al., JMLR 2023

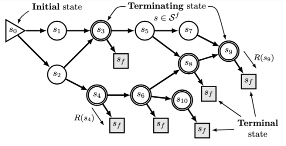
Given *local distributions* (the policy) $P_F(s' \mid s)$, we can define probability distributions over trajectories:

$$P_F(s_0 \to s_1 \to \ldots \to s_n \to s_{n+1} = s_f) = \prod_{i=0}^{n} P_F(s_{i+1} \mid s_i)$$

- **Learning Goal**: Given a DAG $\mathcal{G}$, and a reward function R, find a policy P_F such that the terminating state distribution satisfies for all $s_n \in \mathcal{X}$:

$$P_F^\top(s_n) := \sum_{\tau \in \mathcal{T}:\ \tau \text{ ends in } s_n \to s_f} P_F(\tau) = \frac{R(s_n)}{\sum_{x \in \mathcal{X}} R(x)}.$$

Notations and Problem Setting



Source: "GFlowNet Foundations", Bengio*, Lahlou*, Deleu* et al., JMLR 2023

Given *local distributions* (the policy) $P_F(s' \mid s)$, we can define probability distributions over trajectories:

$$P_F(s_0 \to s_1 \to \ldots \to s_n \to s_{n+1} = s_f) = \prod_{i=0}^{n} P_F(s_{i+1} \mid s_i)$$

- **Learning Goal**: Given a DAG $\mathcal{G}$, and a reward function R, find a policy P_F such that the terminating state distribution satisfies for all $s_n \in \mathcal{X}$:

$$P_F^{\top}(s_n) := \sum_{\tau \in \mathcal{T}:\ \tau \text{ ends in } s_n \to s_f} P_F(\tau) = \frac{R(s_n)}{\sum_{x \in \mathcal{X}} R(x)}.$$

We do this via **flows**: a function $F : \mathbb{A} \to \mathbb{R}^{\geq 0}$ that defines $P_F(s' \mid s) = \frac{F(s \to s')}{\sum_{s'' \in Ch(s)} F(s \to s'')}$.

Sampling Procedure

Algorithm 1 Sampling from a trained GFlowNet

Input: Edge flows $F(s \to s')$ for all edges.

$s \leftarrow s_0$ (Start at the initial state)

While $s \neq s_f$:

- Compute $P_F(s'|s) = \frac{F(s \to s')}{\sum_{s'' \in Child(s)} F(s \to s'')}$ for all children s' of s.
- Sample $s' \sim P_F(s'|s)$
- $s \leftarrow s'$

Return s

An edge-flow function $F : \mathbb{A} \to \mathbb{R}^{\geq 0}$ satisfies:

- the **flow-matching conditions**, if:

$$\forall s' \neq s_0, s_f, \quad \sum_{s \in Par(s')} F(s \to s') = \sum_{s'' \in Child(s')} F(s' \to s'')$$

- the **reward-matching conditions**, if:

$$\forall s \in \mathcal{X} = Par(s_f), \; F(s \to s_f) = R(s)$$

F is then said to be a valid flow.

Main Result

An edge-flow function $F : \mathbb{A} \to \mathbb{R}^{\geq 0}$ satisfies:

- the **flow-matching conditions**, if:

$$\forall s' \neq s_0, s_f, \sum_{s \in Par(s')} F(s \to s') = \sum_{s'' \in Child(s')} F(s' \to s'')$$

- the **reward-matching conditions**, if:

$$\forall s \in \mathcal{X} = Par(s_f),\ F(s \to s_f) = R(s)$$

F is then said to be a valid flow.

Let F be a valid flow. Then, Algorithm 1 samples states $s \in \mathcal{X}$ with probabilities proportional to $R(s)$. In other words, there exists a constant $\alpha > 0$ such that the probability of sampling $s \in \mathcal{X}$ is $\alpha R(s)$.

Naturally, $\alpha^{-1} = \sum_{s \in \mathcal{X}} R(s)$ is the *unknown* partition function.

Proof - Setup

- **Goal:**Prove that the sampling procedure samples states in $\mathcal{X}$ proportionally to their rewards.
- **Strategy:**We'll use strong induction on the maximum depth of a state, to show that $\forall s \in \mathcal{S},\ \sum_{\tau \text{ ending in } s} P(\tau) = \alpha \sum_{s' \in Child(s)} F(s \to s')$, where the sum is over trajectories **that are not necessarily complete**.
- **Notation:**
  - Let $P(\tau)$ be the probability of sampling a trajectory τ.
  - Let $d(s)$ be the maximum depth of state s (length of the longest path from s_0 to s).

Proof - Base Case

- **Base Case:** $d(s) = 1$, meaning $s = s_0$ (the initial state).
- We need to show that $\sum_{\tau \text{ ending in } s_0} P(\tau) = \alpha \sum_{s' \in Child(s_0)} F(s_0 \rightarrow s')$, for some constant α.
- Since s_0 is the initial state, there's only one trajectory ending in it: the empty trajectory.
- Thus, $\sum_{\tau \text{ ending in } s_0} P(\tau) = 1$.
- We can choose $\alpha = \frac{1}{\sum_{s' \in Child(s_0)} F(s_0 \rightarrow s')}$ to satisfy the equation.

Proof - Inductive Step (Part 1)

- **Inductive Hypothesis:** Assume the property holds for all states with maximum depth up to d.
- **Inductive Step:** Consider a state s' with maximum depth $d+1$.
- We want to show that $\sum_{\tau \text{ ends in } s'} P(\tau) = \alpha \sum_{s'' \in Child(s')} F(s' \to s'')$.
- We can write the sum of probabilities of trajectories ending in s' as:

$$\sum_{\tau \text{ ends in } s'} P(\tau) = \sum_{s \in Par(s')} P_F(s'|s) \sum_{\tilde{\tau} \text{ ends in } s} P(\tilde{\tau})$$

Proof - Inductive Step (Part 2)

- Using the inductive hypothesis, we can replace $\sum_{\tilde{\tau} \text{ ends in } s} P(\tilde{\tau})$ with $\alpha \sum_{s'' \in Child(s)} F(s \to s'')$.
- This gives us:

$$\sum_{\tau \text{ ends in } s'} P(\tau) = \sum_{s \in Par(s')} P_F(s'|s) \left(\alpha \sum_{s'' \in Child(s)} F(s \to s'') \right)$$

$$= \alpha \sum_{s \in Par(s')} \frac{F(s \to s')}{\sum_{s'' \in Child(s)} F(s \to s'')} \left(\sum_{s'' \in Child(s)} F(s \to s'') \right)$$

$$= \alpha \sum_{s \in Par(s')} F(s \to s')$$

- By the flow matching property, $\sum_{s \in Par(s')} F(s \to s') = \sum_{s'' \in Child(s')} F(s' \to s'')$.
- Therefore, $\sum_{\tau \text{ ends in } s'} P(\tau) = \alpha \sum_{s'' \in Child(s')} F(s' \to s'')$

Proof - Conclusion (Part 1)

- We have shown that for any state s', the sum of probabilities of trajectories ending in s' is proportional to the sum of flows leaving s'.
- Now, consider the probability of sampling a state $s \in \mathcal{X}$ (a terminal state connected to s_f).
- $P^\top(s) = \sum_{\tau \text{ ends in } s} P(\tau) P_F(s_f|s)$.

Proof - Conclusion (Part 2)

- Using the result from the inductive step, we have:

$$
\begin{aligned}
P^{\top}(s) &= \left(\alpha \sum_{s'' \in Child(s)} F(s \to s'')\right) P_F(s_f|s) \\
&= \left(\alpha \sum_{s'' \in Child(s)} F(s \to s'')\right) \frac{F(s \to s_f)}{\sum_{s'' \in Child(s)} F(s \to s'')} \\
&= \alpha F(s \to s_f)
\end{aligned}
$$

- By the reward matching property, $F(s \to s_f) = R(s)$.
- Therefore, $P(s) = \alpha R(s)$.
- **QED**

How to find the flows?

Simple way to find F: Solve the linear system of equations defined by flow-matching and reward-matching conditions, and positivity constraint. Number of unknowns: $|\mathbb{A}|$

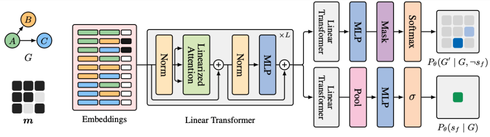
But impractical for interesting spaces (think of the “small molecule” space that is of size $> 10^{60}$).

Estimating Flows

- In practice, the state space is often too large to explicitly represent the flow network.
- **Solution:** Use a neural network to approximate the flow function.

Estimating Flows

- In practice, the state space is often too large to explicitly represent the flow network.
- **Solution:** Use a neural network to approximate the flow function.
- **Example:** For molecular graphs, we can use a Graph Neural Network (GNN) or a Transformer.

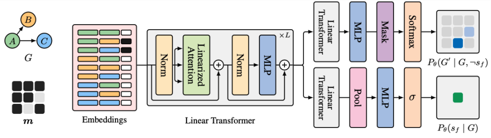


Neural network architecture for approximating the forward transition probabilities $P_\theta(G_{t+1}|G_t)$. The input graph G is encoded as a set of possible edges. Each edge is embedded and fed into a Linear Transformer. Two separate output heads predict the probability of adding a new edge and the probability of terminating the trajectory, respectively.

Source: "Bayesian Structure Learning with Generative Flow Networks", Deleu et al. 2022

Estimating Flows

- In practice, the state space is often too large to explicitly represent the flow network.
- **Solution:** Use a neural network to approximate the flow function.
- **Example:** For molecular graphs, we can use a Graph Neural Network (GNN) or a Transformer.



Neural network architecture for approximating the forward transition probabilities $P_\theta(G_{t+1}|G_t)$. The input graph G is encoded as a set of possible edges. Each edge is embedded and fed into a Linear Transformer. Two separate output heads predict the probability of adding a new edge and the probability of terminating the trajectory, respectively.

Source: "Bayesian Structure Learning with Generative Flow Networks", Deleu et al. 2022

And we get generalization for free!

Solution: Mean Squared Error

- **Idea:** Directly minimize the squared difference between the two sides of the flow matching equations.

Solution: Mean Squared Error

- **Idea:** Directly minimize the squared difference between the two sides of the flow matching equations.
- **Loss function:**

$$\sum_{s \in \mathcal{S} \setminus \{s_0, s_f\}} \left(\sum_{u \to s} F_\theta(u \to s) - R(s)\mathbb{1}(s \in \mathcal{X}) - \sum_{s \to v \neq s_f} F_\theta(s \to v) \right)^2$$

Solution: Mean Squared Error

- **Idea:** Directly minimize the squared difference between the two sides of the flow matching equations.
- **Loss function:**

$$\sum_{s \in \mathcal{S} \setminus \{s_0, s_f\}} \left(\sum_{u \to s} F_\theta(u \to s) - R(s)\mathbb{1}(s \in \mathcal{X}) - \sum_{s \to v \neq s_f} F_\theta(s \to v) \right)^2$$

Digression: Linear Least Squares and TD(0)

- **Linear Least Squares (LLS):** Given a system of linear equations $Ax = b$, LLS finds an approximate solution $\hat{x}$ that minimizes the squared Euclidean norm of the residual: $||A\hat{x} - b||^2$.
- **TD(0) in Reinforcement Learning:**
  - TD(0) learns the value function $V(s)$ of a state s under a policy π.
  - The update rule is: $V(s) \leftarrow V(s) + \alpha(R + \gamma V(s') - V(s))$.
  - This can done by minimizing the squared difference between the two sides of the Bellman equation.

Solution: Mean Squared Error

- **Idea:** Directly minimize the squared difference between the two sides of the flow matching equations.
- **Loss function:**

$$\sum_{s \in \mathcal{S} \setminus \{s_0, s_f\}} \left(\sum_{u \to s} F_\theta(u \to s) - R(s)\mathbb{1}(s \in \mathcal{X}) - \sum_{s \to v \neq s_f} F_\theta(s \to v) \right)^2$$

- **In practice:**

$$\sum_{s \in \mathcal{S} \setminus \{s_0, s_f\}} \left(\log \frac{\sum_{u \to s} F_\theta(u \to s)}{R(s)\mathbb{1}(s \in \mathcal{X}) + \sum_{s \to v \neq s_f} F_\theta(s \to v)} \right)^2$$

Solution: Mean Squared Error

- **Idea:** Directly minimize the squared difference between the two sides of the flow matching equations.
- **Loss function:**

$$\sum_{s\in\mathcal{S}\setminus\{s_0,s_f\}}\left(\sum_{u\to s}F_\theta(u\to s)-R(s)\mathbb{1}(s\in\mathcal{X})-\sum_{s\to v\neq s_f}F_\theta(s\to v)\right)^2$$

- **In practice:**

$$\sum_{s\in\mathcal{S}\setminus\{s_0,s_f\}}\left(\log\frac{\sum_{u\to s}F_\theta(u\to s)}{R(s)\mathbb{1}(s\in\mathcal{X})+\sum_{s\to v\neq s_f}F_\theta(s\to v)}\right)^2$$

- **Problem:** $\sum_{s\in\mathcal{S}\setminus\{s_0,s_f\}}$ is inaccessble in interesting settings.

Solution: Mean Squared Error

- **Idea:** Directly minimize the squared difference between the two sides of the flow matching equations.
- **Loss function:**

$$\sum_{s \in \mathcal{S} \setminus \{s_0, s_f\}} \left(\sum_{u \to s} F_\theta(u \to s) - R(s)\mathbb{1}(s \in \mathcal{X}) - \sum_{s \to v \neq s_f} F_\theta(s \to v) \right)^2$$

- **In practice:**

$$\sum_{s \in \mathcal{S} \setminus \{s_0, s_f\}} \left(\log \frac{\sum_{u \to s} F_\theta(u \to s)}{R(s)\mathbb{1}(s \in \mathcal{X}) + \sum_{s \to v \neq s_f} F_\theta(s \to v)} \right)^2$$

- **Problem:** $\sum_{s \in \mathcal{S} \setminus \{s_0, s_f\}}$ is inaccessble in interesting settings.
- **Solution:** we therefore minimize (an empirical approximation of) $\mathbb{E}_{s \sim p(s)}$, where p is *any* full-support distribution on $\mathcal{S}$, using SGD.

Experimental Setup: Molecule Generation

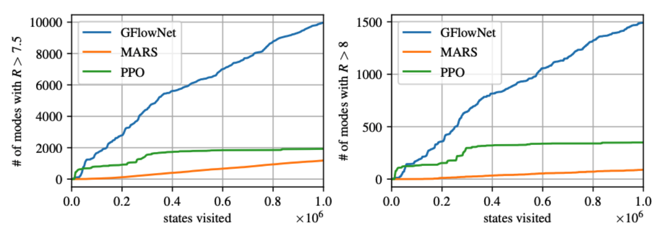
- **Goal:** Generate a diverse set of small molecules with high reward.
- **Environment:** Large-scale environment for sequential molecule generation (up to 10^{60} states, 100-2000 actions per state).
- **Molecule Generation:** Generate molecules by parts using a predefined vocabulary of building blocks (junction tree framework, also called *fragment-based drug design* – See Jin et al., 2020, Kumar et al., 2012, Xie et al., 2021.)
- **Actions:** Choose an atom to attach a block to, choose which block to attach, or stop the editing sequence.
- **DAG:** Multiple action sequences can lead to the same molecule graph.
- **Reward:** Pretrained proxy model (Message Passing NN) that predicts the binding energy of a molecule to a protein target (sEH).
- MCMC Baseline ("MARS: Markov Molecular Sampling for Multi-objective Drug Discovery", Xie et al., 2021. (SOTA before GFNs))

Experimental Setup: Molecule Generation (Continued)

- **Proxy Model:** MPNN over the atom graph, trained on 300k molecules with docking scores.
- **Flow Predictor:** MPNN over the junction tree graph (similar to MARS).
- **Training:** All models trained with up to 10^6 molecules.
- **Exploratory Policy:** Mixture between $P_F(a \mid s)$ with probability 0.95 and a uniform distribution over allowed actions with probability 0.05.

Experimental Results: Molecule Generation (Continued)

- **High-Reward Molecule Discovery:** GFlowNet finds significantly more unique molecules with a score above 8 than the proxy's dataset.
- **Diversity:** GFlowNet generates more diverse candidates (lower average pairwise Tanimoto similarity) compared to MARS and PPO.
- **Mode Discovery:** GFlowNet discovers significantly more modes (Bemis-Murcko scaffolds) than MARS.



Source: "Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation", E. Bengio et al., 2021

GFlowNet discovers significantly more modes (Bemis-Murcko scaffolds) than MARS.

Limitations of Flow Matching

$$\sum_{s \in \mathcal{S}} \left(\log \frac{\sum_{u \to s} F_\theta(u \to s)}{\sum_{s \to v} F_\theta(s \to v)} \right)^2$$

- **Cost:** Evaluating a term of the sum requires $n + 1$ neural network calls, where n is the number of parents of a state s.
- **Locality:** Flow matching objective is local - it only considers the in-flow and out-flow of individual states.
- **Slow Credit Assignment:** Updates mainly affect states near high-reward outcomes, leading to slow propagation of information.

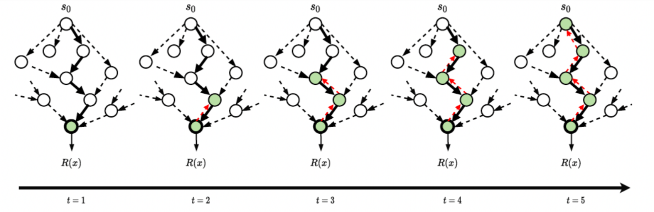


Illustration of slow credit assignment in flow matching. The update from a high-reward state propagates slowly backwards through the trajectory.

Detailed Balance Objective

- **Idea:** Instead of parameterizing edge flows directly, learn:
  - **Forward Policy** $P_F(\cdot \mid s)$: Distribution over children of each non-terminal state s .
  - **State Flow** $F(\cdot)$: A scalar value for each state.
  - **Backward Policy** $P_B(\cdot \mid s)$: Distribution over parents for each non-initial state s (can be either learned or fixed!)

$$\mathcal{L}_{DB}(s \to s') = \left(\log \frac{F_\theta(s) P_F^\theta(s'|s)}{\mathbb{1}_{s' \neq s_f} F_\theta(s') P_B^\theta(s|s') + \mathbb{1}_{s' = s_f} R(s)} \right)^2$$

This objective/loss is equivalent to the flow-matching + reward-matching objectives/loss – "GFlowNet Foundations", Bengio*, Lahlou*, Deleu* et al., JMLR 2023

Trajectory Balance Objective

Trajectory Balance: A Trajectory-Level Objective, "Trajectory balance: Improved credit assignment in GFlowNets", Malkin et al. 2023

$$\mathcal{L}_{TB}(\tau; Z^\theta, P_F^\theta, P_B^\theta) = \left(\log \frac{Z^\theta \prod_{i=1}^n P_F^\theta(s_i|s_{i-1})}{R(x) \prod_{i=1}^n P_B^\theta(s_{i-1}|s_i)}\right)^2 = \left(\log \frac{Z^\theta P_F^\theta(\tau)}{R(x) P_B^\theta(\tau|x)}\right)^2$$

While not satisfied:

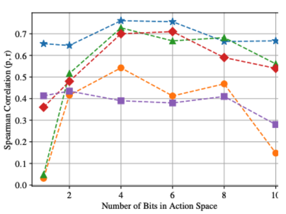
- Sample a trajectory τ by iteratively sampling states $s' \sim P_F(. \mid s)$ starting from s_0 - or a modified version of P_F (e.g., tempered - to induce diversity) - or any other "full support" policy
- Evaluate $\nabla_\theta \mathcal{L}_{TB}(\tau; Z^\theta, P_F^\theta, P_B^\theta)$ (automatic-differentiation)
- $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{TB}(\tau; Z^\theta, P_F^\theta, P_B^\theta)$

Bit Sequence Generation Task

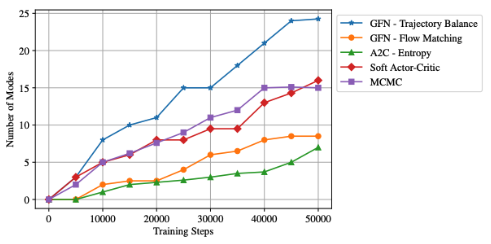
- **Goal:** Generate bit sequences of length $n = 120$ with modes at a fixed set M unknown to the learner.
- **Reward Function:** $R(x) = \exp(1 - \min_{y \in M} d(x, y)/n)$, where d is the edit distance.
- **Action Space:** For different integers k dividing n, actions append a k-bit "word" to the end of a partial sequence. Trajectory length is n/k.
- **Methods Compared:**
  - GFlowNet with TB
  - GFlowNet with FM (equivalent to DB and Soft Q-Learning in this case)
  - A2C with Entropy Regularization
  - Soft Actor-Critic (SAC)
  - MARS
- **Architecture:** Transformer-based architecture for all methods.

Bit Sequence Generation Results

$n = 120,\ |M| = 60,\ k \in \{1, 2, 4, 6, 8, 10\}$



Spearman correlation vs. number of bits k in the action space.



Number of modes discovered during training with $k = 1$.

- **Observation (Left):** GFlowNets with TB have the highest correlation across all action space sizes. FM's performance improves with increasing k (shorter trajectories) but degrades with larger action spaces.
- **Observation (Right):** For a fixed k, GFlowNets with TB discover more modes faster than other methods.

Conditional GFlowNets

- Like any *generative model*, we can condition a GFlowNet on some auxiliary data, or context
- For example, we could imagine the same structured space, but different reward functions encoding different desiderate
- We can make a GFlowNet conditional by training it with the **condition as an input**: $P_F(s' \mid s, condition)$
- For example, this has been used for language modeling, where the policy P_F corresponds to a (large) language model, and the *condition* is the *prompt* or *context*: "Amortizing intractable inference in large language models", Hu et al. 2023
- We get generalization across conditions for free!

Application in LLM reasoning

"GFlowNet-Finetuning of Language Models using Process Reward Models for Mathematical Reasoning", Younsi, ..., Lahlou, 2025 (under review)

Disclaimer: Even though "reasoning" is in the title, I do not *believe* that LLMs *reason*. In fact, I do not know what *reasoning* is.

This is about "chain-of-thought reasoning", or "finding the response to a prompt by generating intermediate steps"

Application in LLM reasoning

"GFlowNet-Finetuning of Language Models using Process Reward Models for Mathematical Reasoning", Younsi, ..., Lahlou, 2025 (under review)

Application in LLM reasoning

"GFlowNet-Finetuning of Language Models using Process Reward Models for Mathematical Reasoning", Younsi, ..., Lahlou, 2025 (under review)

Application in LLM reasoning

"GFlowNet-Finetuning of Language Models using Process Reward Models for Mathematical Reasoning", Younsi, ..., Lahlou, 2025 (under review)

- Generating tokens, or better yet, *reasoning steps*, can be described by a DAG (a tree actually)
- RLHF inherently aims to maximize a single reward signal, often leading to the exploitation of a narrow set of solution strategies
- We can score both "complete reasoning paths" and "partial reasoning paths" using MCTS-based *process reward models* (PRM, "Solving math word problems with process- and outcome-based feedback", Uesato et al. 2022)
- We finetune with 10k entries of the OpenMathInstruct-2 dataset (NVIDIA, 2024), using a PRM (Qwen2.5-7B-math finetuned on a dataset generated using 70k entries of OpenMathInstruct-2)

“GFlowNet-Finetuning of Language Models using Process Reward Models for Mathematical Reasoning”, Younsi, ..., Lahlou, 2025 (under review)

Find the sum of all three-digit positive integers whose cubes end with the digits 125.	To find the sum of all three-digit positive integers whose cubes end with the digits 125, let's analyze how a number's cube can end in 125. A number n whose cube ends in 125 must be of the form $n = 5k$, where k is an integer. This is because the last digit of n^3 is determined by the last digit of n, and only numbers ending in 5 have cubes that end in 5. Now, consider the last two digits of n^3, which are determined by the last two digits of n. The only possibilities for the last two digits of n that result in a cube ending in 25 are 25, 75, and 25 + 50 = 75, 75 + 50 = 25 + 100 = 125, etc. Thus, the last two digits of n must be 25, 75, or 25 + 50k for some integer k, where 25 + 50k < 1000. Since we're looking for three-digit numbers, the possible values for the hundreds digit of n are 1 through 9. Considering these constraints, we find the following three-digit numbers whose cubes end with the digits 125: * 125 * 375 * 625 * 875 Now, let's find the sum of these numbers: \[125 + 375 + 625 + 875 = 500 + 625 + 875 = 1125 + 875 = 2000 \] So, the sum of all three-digit positive integers whose cubes end with the digits 125 is $\boxed{2000}$.	2000

Example entry of the OpenMathInstruct-2 dataset

"GFlowNet-Finetuning of Language Models using Process Reward Models for Mathematical Reasoning", Younsi, ..., Lahlou, 2025 (under review)

Table 2: Performance Comparison on Mathematical Reasoning Tasks

Model	MATH Level 5	SAT MATH	GSM8K
Llama3.2-3B-it	14.46%	65.6%	67.8%
+ PPO	15.32%	70.0%	68.4%
+ GFlowNet	**17.05%**	**75.0%**	**68.5%**
Llama3.1-8B-it	17.96%	81.2%	78.1%
+ PPO	18.44%	81.2%	**79.1%**
+ GFlowNet	**18.67%**	**84.4%**	79.0%

Table 3: Solution Diversity Analysis

Model	Semantic Similarity
Llama3.2-3B-it	0.80
+ PPO	0.82
+ GFlowNet	**0.78**

Why Continuous GFlowNets?

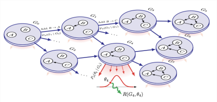
"A theory of continuous generative flow networks", Lahlou et al., ICML 2023

- GFlowNets have proven advantages over and connections to:
  - MCMC
  - Reinforcement Learning
  - Hierarchical Variational Inference
- The proven advantages have been confirmed in discrete scenarios:
  - Biological sequence design
  - Bayesian *structure* learning
  - Robust scheduling problem
  - *Discrete* image modeling
- Many interesting sampling problems do not exhibit a discrete DAG structure:

Why Continuous GFlowNets?

"A theory of continuous generative flow networks", Lahlou et al., ICML 2023

- GFlowNets have proven advantages over and connections to:
  - MCMC
  - Reinforcement Learning
  - Hierarchical Variational Inference
- The proven advantages have been confirmed in discrete scenarios:
  - Biological sequence design
  - Bayesian *structure* learning
  - Robust scheduling problem
  - *Discrete* image modeling
- Many interesting sampling problems do not exhibit a discrete DAG structure:
  - Bayesian *structure* learning with *parameters* (Given a dataset $\mathcal{D}$, learn $p(G, \boldsymbol{\theta} \mid \mathcal{D}) \propto p(\mathcal{D} \mid G, \boldsymbol{\theta})p(G, \boldsymbol{\theta})$, where $(G, \boldsymbol{\theta})$ is a directed graphical model)

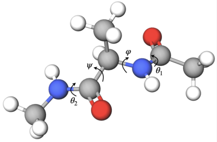


"Joint Bayesian Inference of Graphical Structure and Parameters with a Single Generative Flow Network", Deleu et al. 2023

Why Continuous GFlowNets?

"A theory of continuous generative flow networks", Lahlou et al., ICML 2023

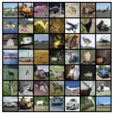
- GFlowNets have proven advantages over and connections to:
  - MCMC
  - Reinforcement Learning
  - Hierarchical Variational Inference
- The proven advantages have been confirmed in discrete scenarios:
  - Biological sequence design
  - Bayesian *structure* learning
  - Robust scheduling problem
  - *Discrete* image modeling
- Many interesting sampling problems do not exhibit a discrete DAG structure:
  - Bayesian *structure* learning with *parameters* (Given a dataset $\mathcal{D}$, learn $p(G, \boldsymbol{\theta} \mid \mathcal{D}) \propto p(\mathcal{D} \mid G, \boldsymbol{\theta})p(G, \boldsymbol{\theta})$, where $(G, \boldsymbol{\theta})$ is a directed graphical model)
  - Molecular conformation design



Why Continuous GFlowNets?

"A theory of continuous generative flow networks", Lahlou et al., ICML 2023

- GFlowNets have proven advantages over and connections to:
    - MCMC
    - Reinforcement Learning
    - Hierarchical Variational Inference
- The proven advantages have been confirmed in discrete scenarios:
    - Biological sequence design
    - Bayesian *structure* learning
    - Robust scheduling problem
    - *Discrete* image modeling
- Many interesting sampling problems do not exhibit a discrete DAG structure:
    - Bayesian *structure* learning with *parameters* (Given a dataset $\mathcal{D}$, learn $p(G, \boldsymbol{\theta} \mid \mathcal{D}) \propto p(\mathcal{D} \mid G, \boldsymbol{\theta}) p(G, \boldsymbol{\theta})$, where $(G, \boldsymbol{\theta})$ is a directed graphical model)
    - Molecular conformation design
    - Image generation



"Unifying Generative Models with GFlowNets and Beyond", Zhang et al. 2022

Why Continuous GFlowNets?

"A theory of continuous generative flow networks", Lahlou et al., ICML 2023

- GFlowNets have proven advantages over and connections to:
  - MCMC
  - Reinforcement Learning
  - Hierarchical Variational Inference
- The proven advantages have been confirmed in discrete scenarios:
  - Biological sequence design
  - Bayesian *structure* learning
  - Robust scheduling problem
  - *Discrete* image modeling
- Many interesting sampling problems do not exhibit a discrete DAG structure:
  - Bayesian *structure* learning with *parameters* (Given a dataset $\mathcal{D}$, learn $p(G, \boldsymbol{\theta} \mid \mathcal{D}) \propto p(\mathcal{D} \mid G, \boldsymbol{\theta}) p(G, \boldsymbol{\theta})$, where $(G, \boldsymbol{\theta})$ is a directed graphical model)
  - Molecular conformation design
  - Image generation
  - ...

Desiderata

- The ability to describe a **continuum** of *children* and *parents* of a state, of *arbitrary* **dimension**.

How to describe a DAG-like structure in a general space ?

Desiderata

- The ability to describe a **continuum** of *children* and *parents* of a state, of *arbitrary* **dimension**.
- The ability to mix between both **continuous** and a **discrete** components in describing *children* and *parents*:

How to describe a DAG-like structure in a general space ?

Desiderata

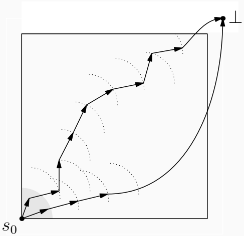
- The ability to describe a **continuum** of *children* and *parents* of a state, of *arbitrary* **dimension**.
- The ability to mix between both **continuous** and a **discrete** components in describing *children* and *parents*:
  - **Example**: The child set of a state s can be the union of a continuous subset of the state space $\mathcal{S}$ and the sink state s_f (denoted $\perp$ sometimes).

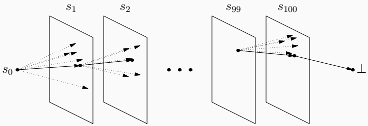
How to describe a DAG-like structure in a general space ?

Desiderata

- The ability to describe a **continuum** of *children* and *parents* of a state, of *arbitrary* **dimension**.
- The ability to mix between both **continuous** and a **discrete** components in describing *children* and *parents*:
  - **Example**: The child set of a state s can be the union of a continuous subset of the state space $\mathcal{S}$ and the sink state s_f (denoted $\perp$ sometimes).

Examples





How to describe a DAG-like structure in a general space ?

Desiderata

- The ability to describe a **continuum** of *children* and *parents* of a state, of *arbitrary* **dimension**.
- The ability to mix between both **continuous** and a **discrete** components in describing *children* and *parents*:

Examples

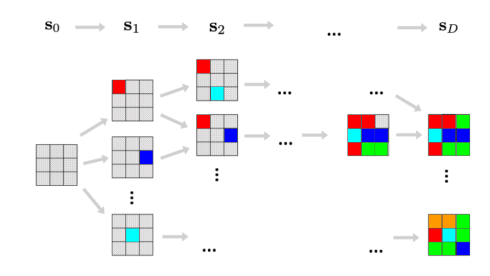


Figure (modified) from `Generative Flow Networks for Discrete Probabilistic Modeling`, Zhang et al., 2022

How to describe a DAG-like structure in a general space ?

Desiderata

- The ability to describe a **continuum** of *children* and *parents* of a state, of *arbitrary* **dimension**.
- The ability to mix between both **continuous** and a **discrete** components in describing *children* and *parents*:

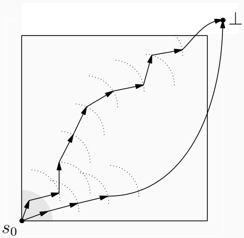
Appropriate mathematical tool

A transition kernel on a measurable space $(\mathcal{S}, \Sigma)$ is a function $\kappa : \mathcal{S} \times \Sigma \to \mathbb{R}^+$ such that:

- $\forall B \in \Sigma,\ s \mapsto \kappa(s, B)$ is measurable
- $\forall s \in \mathcal{S},\ B \mapsto \kappa(s, B)$ is a positive measure on $(\mathcal{S}, \Sigma)$

How to describe a DAG-like structure in a general space ?

Examples



$\mathcal{S} = [0,1]^2$

- $\kappa(s_0, B) = 0$ if B does not intersect the bottom left quarter disk $\rightarrow$ *Support* of $\kappa(s_0, -)$ is the quarter disk.
- $\kappa(s, B) = 0$ if B does not intersect the corresponding quarter circle, and does not contain $\perp$ $\rightarrow$ *Support* of $\kappa(s, -)$ is the union of the quarter circle and the singleton $\{\perp\}$.

Appropriate mathematical tool

A transition kernel on a measurable space $(\mathcal{S}, \Sigma)$ is a function $\kappa : \mathcal{S} \times \Sigma \rightarrow \mathbb{R}^+$ such that:

- $\forall B \in \Sigma,\ s \mapsto \kappa(s, B)$ is measurable
- $\forall s \in \mathcal{S},\ B \mapsto \kappa(s, B)$ is a positive measure on $(\mathcal{S}, \Sigma)$

Everything works out well with densities rather than PMFs

Discrete GFlowNets	**Generalized GFlowNets**
Directed acyclic pointed graph $G = (\mathcal{S}, \mathbb{A}, s_0, s_f)$	Measurable pointed graph $G = (\bar{\mathcal{S}}, \mathcal{T}, \Sigma, s_0, s_f, \kappa, \kappa^b, \nu)$
Children and parents of a state s	Supports of measures $\kappa(s, -)$ and $\kappa^b(s, -)$
State flow function F	Flow measure μ, of density u wrt ν
Forward policy P_F	Forward kernel P_F, of density p_F wrt κ
Reward function R	Reward measure R, of density r wrt ν

$(\bar{\mathcal{S}}, \mathcal{T})$ is a topological space ($\mathcal{T}$ is the set of open subsets of $\bar{\mathcal{S}}$). Σ is the Borel σ-algebra associated to the topology on $\bar{\mathcal{S}}$.

s_0 and s_f are the source and sink states.

κ, κ^b are two σ-finite kernels on $(\bar{\mathcal{S}}, \Sigma)$. ν is a σ-finite measure on $(\bar{\mathcal{S}}, \Sigma)$.

$$L_{DB}(s, s'; \theta) = \left(\log \frac{u(s; \theta) p_F(s, s'; \theta)}{u(s'; \theta) p_B(s', s; \theta)} \right)^2$$

$$L_{TB}^n(\tau; \theta) = \left(\log \frac{Z_\theta \prod_{t=0}^{n} p_F(s_t, s_{t+1}; \theta)}{r(s_n) \prod_{t=0}^{n-1} p_B(s_{t+1}, s_t; \theta)} \right)^2$$

Everything works out well with densities rather than PMFs

Discrete GFlowNets	**Generalized GFlowNets**
Directed acyclic pointed graph $G = (\mathcal{S}, \mathbb{A}, s_0, s_f)$	Measurable pointed graph $G = (\bar{\mathcal{S}}, \mathcal{T}, \Sigma, s_0, s_f, \kappa, \kappa^b, \nu)$
Children and parents of a state s	Supports of measures $\kappa(s, -)$ and $\kappa^b(s, -)$
State flow function F	Flow measure μ, of density u wrt ν
Forward policy P_F	Forward kernel P_F, of density p_F wrt κ
Reward function R	Reward measure R, of density r wrt ν

$(\bar{\mathcal{S}}, \mathcal{T})$ is a topological space ($\mathcal{T}$ is the set of open subsets of $\bar{\mathcal{S}}$). Σ is the Borel σ-algebra associated to the topology on $\bar{\mathcal{S}}$.

s_0 and s_f are the source and sink states.

κ, κ^b are two σ-finite kernels on $(\bar{\mathcal{S}}, \Sigma)$. ν is a σ-finite measure on $(\bar{\mathcal{S}}, \Sigma)$.

$$L_{DB}(s, s'; \theta) = \left(\log \frac{u(s; \theta) p_F(s, s'; \theta)}{u(s'; \theta) p_B(s', s; \theta)} \right)^2$$

$$L_{TB}^n(\tau; \theta) = \left(\log \frac{Z_\theta \prod_{t=0}^{n} p_F(s_t, s_{t+1}; \theta)}{r(s_n) \prod_{t=0}^{n-1} p_B(s_{t+1}, s_t; \theta)} \right)^2$$

Everything works out well with densities rather than PMFs

Discrete GFlowNets	**Generalized GFlowNets**
Directed acyclic pointed graph $G = (\mathcal{S}, \mathbb{A}, s_0, s_f)$	Measurable pointed graph $G = (\bar{\mathcal{S}}, \mathcal{T}, \Sigma, s_0, s_f, \kappa, \kappa^b, \nu)$
Children and parents of a state s	Supports of measures $\kappa(s, -)$ and $\kappa^b(s, -)$
State flow function F	Flow measure μ, of density u wrt ν
Forward policy P_F	Forward kernel P_F, of density p_F wrt κ
Reward function R	Reward measure R, of density r wrt ν

$(\bar{\mathcal{S}}, \mathcal{T})$ is a topological space ($\mathcal{T}$ is the set of open subsets of $\bar{\mathcal{S}}$). Σ is the Borel σ-algebra associated to the topology on $\bar{\mathcal{S}}$.

s_0 and s_f are the source and sink states.

κ, κ^b are two σ-finite kernels on $(\bar{\mathcal{S}}, \Sigma)$. ν is a σ-finite measure on $(\bar{\mathcal{S}}, \Sigma)$.

$$L_{DB}(s, s'; \theta) = \left(\log \frac{u(s;\theta) p_F(s, s'; \theta)}{u(s';\theta) p_B(s', s; \theta)}\right)^2$$

$$L_{TB}^n(\tau; \theta) = \left(\log \frac{Z_\theta \prod_{t=0}^{n} p_F(s_t, s_{t+1}; \theta)}{r(s_n) \prod_{t=0}^{n-1} p_B(s_{t+1}, s_t; \theta)}\right)^2$$

Everything works out well with densities rather than PMFs

Discrete GFlowNets	**Generalized GFlowNets**
Directed acyclic pointed graph $G = (\mathcal{S}, \mathbb{A}, s_0, s_f)$	Measurable pointed graph $G = (\bar{\mathcal{S}}, \mathcal{T}, \Sigma, s_0, s_f, \kappa, \kappa^b, \nu)$
Children and parents of a state s	Supports of measures $\kappa(s, -)$ and $\kappa^b(s, -)$
State flow function F	Flow measure μ, of density u wrt ν
Forward policy P_F	Forward kernel P_F, of density p_F wrt κ
Reward function R	Reward measure R, of density r wrt ν

$(\bar{\mathcal{S}}, \mathcal{T})$ is a topological space ($\mathcal{T}$ is the set of open subsets of $\bar{\mathcal{S}}$). Σ is the Borel σ-algebra associated to the topology on $\bar{\mathcal{S}}$.

s_0 and s_f are the source and sink states.

κ, κ^b are two σ-finite kernels on $(\bar{\mathcal{S}}, \Sigma)$. ν is a σ-finite measure on $(\bar{\mathcal{S}}, \Sigma)$.

$$L_{DB}(s, s'; \theta) = \left(\log \frac{u(s;\theta)p_F(s, s';\theta)}{u(s';\theta)p_B(s', s;\theta)}\right)^2$$

$$L_{TB}^n(\tau; \theta) = \left(\log \frac{Z_\theta \prod_{t=0}^{n} p_F(s_t, s_{t+1}; \theta)}{r(s_n) \prod_{t=0}^{n-1} p_B(s_{t+1}, s_t; \theta)}\right)^2$$

Everything works out well with densities rather than PMFs

Discrete GFlowNets	Generalized GFlowNets
Directed acyclic pointed graph $G = (\mathcal{S}, \mathbb{A}, s_0, s_f)$	Measurable pointed graph $G = (\bar{\mathcal{S}}, \mathcal{T}, \Sigma, s_0, s_f, \kappa, \kappa^b, \nu)$
Children and parents of a state s	Supports of measures $\kappa(s, -)$ and $\kappa^b(s, -)$
State flow function F	Flow measure μ, of density u wrt ν
Forward policy P_F	Forward kernel P_F, of density p_F wrt κ
Backward policy P_B	Backward kernel P_B, of density p_B wrt κ^b
Reward function R	Reward measure R, of density r wrt ν

$(\bar{\mathcal{S}}, \mathcal{T})$ is a topological space ($\mathcal{T}$ is the set of open subsets of $\bar{\mathcal{S}}$). Σ is the Borel σ-algebra associated to the topology on $\bar{\mathcal{S}}$.

s_0 and s_f are the source and sink states.

κ, κ^b are two σ-finite kernels on $(\bar{\mathcal{S}}, \Sigma)$. ν is a σ-finite measure on $(\bar{\mathcal{S}}, \Sigma)$.

$$L_{DB}(s, s'; \theta) = \left(\log \frac{u(s;\theta) p_F(s, s'; \theta)}{u(s';\theta) p_B(s', s; \theta)}\right)^2$$

$$L_{TB}^n(\tau; \theta) = \left(\log \frac{Z_\theta \prod_{t=0}^{n} p_F(s_t, s_{t+1}; \theta)}{r(s_n) \prod_{t=0}^{n-1} p_B(s_{t+1}, s_t; \theta)}\right)^2$$

GFlowNets and HVMs

"GFlowNets and variational inference", Malkin*, Lahlou*, Deleu* et al., ICLR 2023

Given *any* backward policy $P_B(s \mid s')$, and target marginal $\frac{R(x)}{Z}$, that jointly define a target distribution over trajectories $P_B(\tau)$:

$$P_B(\tau) = \underbrace{\frac{R(x_\tau)}{Z}}_{\text{unknown}} \prod_{s \to s' \in \tau, s' \neq s_f} P_B(s \mid s')$$

If we find a policy $P_F(s' \mid s)$, defining a distribution over trajectories $P_F(\tau) = \prod_{s \to s'} P_F(s' \mid s)$, that equals the target $P_B(\tau)$

Then, naturally, *following that policy* would lead to samples from the target marginal

GFlowNets and HVMs

"GFlowNets and variational inference", Malkin*, Lahlou*, Deleu* et al., ICLR 2023

Given *any* backward policy $P_B(s \mid s')$, and target marginal $\frac{R(x)}{Z}$, that jointly define a target distribution over trajectories $P_B(\tau)$:

$$P_B(\tau) = \underbrace{\frac{R(x_\tau)}{Z}}_{\text{unknown}} \prod_{s \to s' \in \tau, s' \neq s_f} P_B(s \mid s')$$

If we find a policy $P_F(s' \mid s)$, defining a distribution over trajectories $P_F(\tau) = \prod_{s \to s'} P_F(s' \mid s)$, that equals the target $P_B(\tau)$

Then, naturally, *following that policy* would lead to samples from the target marginal

$$\mathcal{L}_{\mathrm{HVI},f}(P_F, P_B) = D_f(P_B(\tau) \| P_F(\tau))$$

GFlowNets and HVMs

"GFlowNets and variational inference", Malkin*, Lahlou*, Deleu* et al., ICLR 2023

Given *any* backward policy $P_B(s \mid s')$, and target marginal $\frac{R(x)}{Z}$, that jointly define a target distribution over trajectories $P_B(\tau)$:

$$P_B(\tau) = \underbrace{\frac{R(x_\tau)}{Z}}_{\text{unknown}} \prod_{s \to s' \in \tau, s' \neq s_f} P_B(s \mid s')$$

If we find a policy $P_F(s' \mid s)$, defining a distribution over trajectories $P_F(\tau) = \prod_{s \to s'} P_F(s' \mid s)$, that equals the target $P_B(\tau)$

Then, naturally, *following that policy* would lead to samples from the target marginal

$$\mathcal{L}_{\text{HVI},f}(P_F, P_B) = D_f(P_B(\tau) \| P_F(\tau))$$

Example:

$$\begin{aligned} D_{\mathrm{KL}}(P_F \| P_B) &= \mathbb{E}_{P_F(\tau)}\left[\log \frac{P_F(\tau)}{P_B(\tau)}\right] \\ &= \mathbb{E}_{P_F(\tau)}\left[\log \frac{P_F(\tau)}{R(x_\tau) \prod_{s \to s' \in \tau, s' \neq s_f} P_B(s \mid s')}\right] + \log Z \end{aligned}$$

GFlowNets and HVMs

"GFlowNets and variational inference", Malkin*, Lahlou*, Deleu* et al., ICLR 2023

Given *any* backward policy $P_B(s \mid s')$, and target marginal $\frac{R(x)}{Z}$, that jointly define a target distribution over trajectories $P_B(\tau)$:

$$P_B(\tau) = \underbrace{\frac{R(x_\tau)}{Z}}_{\text{unknown}} \prod_{s \to s' \in \tau, s' \neq s_f} P_B(s \mid s')$$

If we find a policy $P_F(s' \mid s)$, defining a distribution over trajectories $P_F(\tau) = \prod_{s \to s'} P_F(s' \mid s)$, that equals the target $P_B(\tau)$

Then, naturally, *following that policy* would lead to samples from the target marginal

	Loss	
Algorithm	P_F (sampler)	P_B (posterior)
Reverse KL	$D_{\mathrm{KL}}(P_F \| P_B)$	$D_{\mathrm{KL}}(P_F \| P_B)$
Forward KL	$D_{\mathrm{KL}}(P_B \| P_F)$	$D_{\mathrm{KL}}(P_B \| P_F)$
Wake-sleep (WS)	$D_{\mathrm{KL}}(P_B \| P_F)$	$D_{\mathrm{KL}}(P_F \| P_B)$
Reverse wake-sleep	$D_{\mathrm{KL}}(P_F \| P_B)$	$D_{\mathrm{KL}}(P_B \| P_F)$

GFlowNets and HVMs

"GFlowNets and variational inference", Malkin*, Lahlou*, Deleu* et al., ICLR 2023

Given *any* backward policy $P_B(s \mid s')$, and target marginal $\frac{R(x)}{Z}$, that jointly define a target distribution over trajectories $P_B(\tau)$

If we find a policy $P_F(s' \mid s)$, defining a distribution over trajectories $P_F(\tau) = \prod_{s \to s'} P_F(s' \mid s)$, that equals the target $P_B(\tau)$

Then, naturally, *following that policy* would lead to samples from the target marginal

GFlowNet (Trajectory Balance)

$$L_{\text{TB}}(\tau) = \left(\log \frac{Z_\phi P_F(\tau)}{R(x_\tau) P_B(\tau \mid x_\tau)}\right)^2$$

The learner is free to decide where trajectories τ come from: off-policy, RL exploration methods, ...

HVM

$$\mathcal{L}_{\text{HVI},f}(P_F, P_B) = D_f(P_B(\tau) \| P_F(\tau))$$

	(Surrogate) loss	
Algorithm	P_F (sampler)	P_B (posterior)
Reverse KL	$D_{\mathrm{KL}}(P_F \| P_B)$	$D_{\mathrm{KL}}(P_F \| P_B)$
Forward KL	$D_{\mathrm{KL}}(P_B \| P_F)$	$D_{\mathrm{KL}}(P_B \| P_F)$
Wake-sleep (WS)	$D_{\mathrm{KL}}(P_B \| P_F)$	$D_{\mathrm{KL}}(P_F \| P_B)$
Reverse wake-sleep	$D_{\mathrm{KL}}(P_F \| P_B)$	$D_{\mathrm{KL}}(P_B \| P_F)$

Objectives in red and off-policy training require importance weighting

GFlowNets and HVMs

"GFlowNets and variational inference", Malkin*, Lahlou*, Deleu* et al., ICLR 2023

Given *any* backward policy $P_B(s \mid s')$, and target marginal $\frac{R(x)}{Z}$, that jointly define a target distribution over trajectories $P_B(\tau)$

If we find a policy $P_F(s' \mid s)$, defining a distribution over trajectories $P_F(\tau) = \prod_{s \to s'} P_F(s' \mid s)$, that equals the target $P_B(\tau)$

Then, naturally, *following that policy* would lead to samples from the target marginal

GFlowNet (Trajectory Balance)

$$L_{\text{TB}}(\tau) = \left(\log \frac{Z_\phi P_F(\tau)}{R(x_\tau) P_B(\tau \mid x_\tau)}\right)^2$$

The learner is free to decide where trajectories τ come from: off-policy, RL exploration methods, ...

HVM

$$\mathcal{L}_{\text{HVI},f}(P_F, P_B) = D_f(P_B(\tau) \| P_F(\tau))$$

	(Surrogate) loss	
Algorithm	P_F (sampler)	P_B (posterior)
Reverse KL	$D_{\mathrm{KL}}(P_F \| P_B)$	$D_{\mathrm{KL}}(P_F \| P_B)$
Forward KL	$D_{\mathrm{KL}}(P_B \| P_F)$	$D_{\mathrm{KL}}(P_B \| P_F)$
Wake-sleep (WS)	$D_{\mathrm{KL}}(P_B \| P_F)$	$D_{\mathrm{KL}}(P_F \| P_B)$
Reverse wake-sleep	$D_{\mathrm{KL}}(P_F \| P_B)$	$D_{\mathrm{KL}}(P_B \| P_F)$

Objectives in red and off-policy training require importance weighting

GFlowNets are more amenable to stable off-policy training and thus allow to easily promote exploration

Summary of theoretical connections

In certain cases, hierarchical variational algorithms are equivalent, in the sense of expected gradients, to special cases of GFlowNets

$$\nabla_\theta D_{\mathrm{KL}}(P_F^\theta \parallel P_B^\phi) = \frac{1}{2}\,\mathbb{E}_{\tau\sim P_F}\left[\nabla_\theta L_{\mathrm{TB}}(\tau)\right]$$

$$\nabla_\phi D_{\mathrm{KL}}(P_B^\phi \parallel P_F^\theta) = \frac{1}{2}\,\mathbb{E}_{\tau\sim P_B}\left[\nabla_\phi L_{\mathrm{TB}}(\tau)\right]$$

But...

$$D_{\mathrm{KL}}(P_F(.;\theta)\|P_B(.;\phi)) = \mathbb{E}_{P_F(\tau;\theta)}\left[\log \frac{P_F(\tau;\theta)}{R(x_\tau)P_B(\tau \mid x_\tau;\phi)}\right] + \log Z$$

The gradient requires a *score function estimator* (REINFORCE).
The GFlowNet TB loss performs variance reduction for free ($\log Z$ plays the role of a *learned* control variate / baseline)

You can play with GFlowNets using
`https://github.com/saleml/torchgfn`

Thank you for your attention

salem.lahlou@mbzuai.ac.ae
`https://la7.lu`