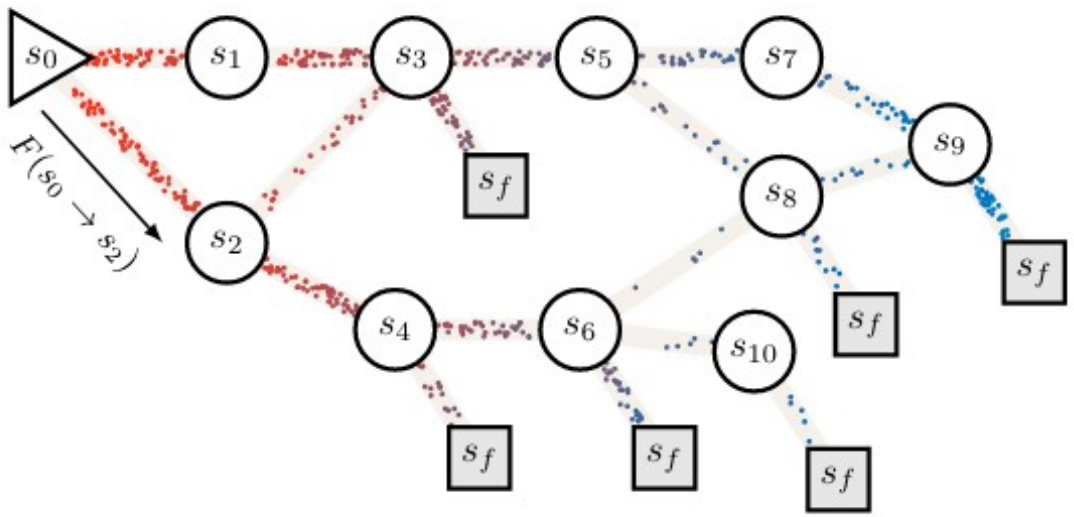
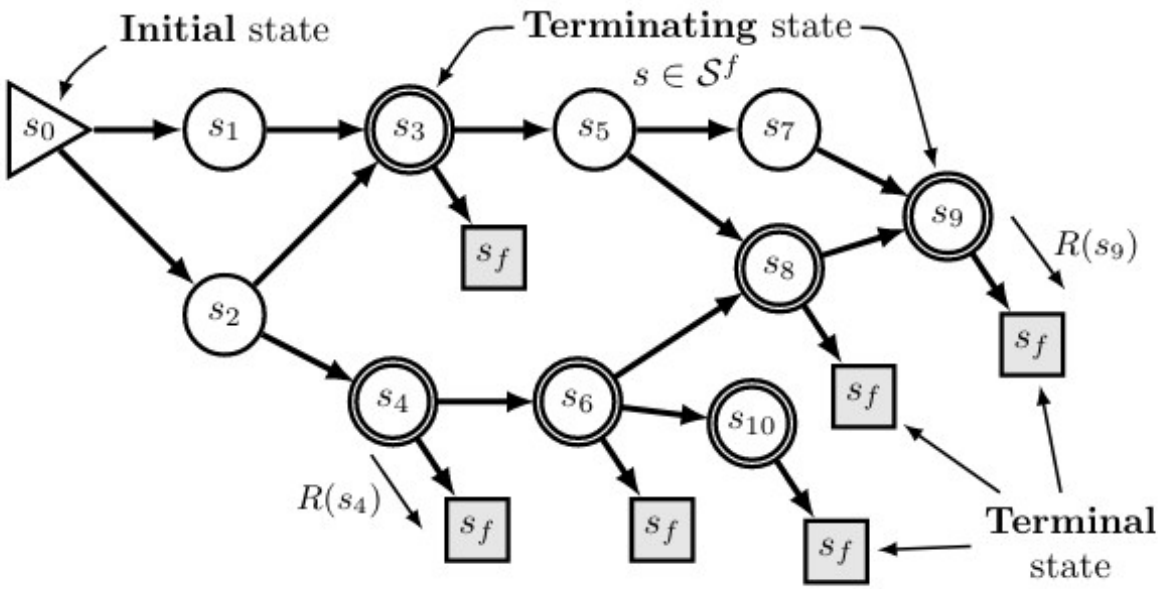
Let’s Improve GFN





Flow Conservation !!!

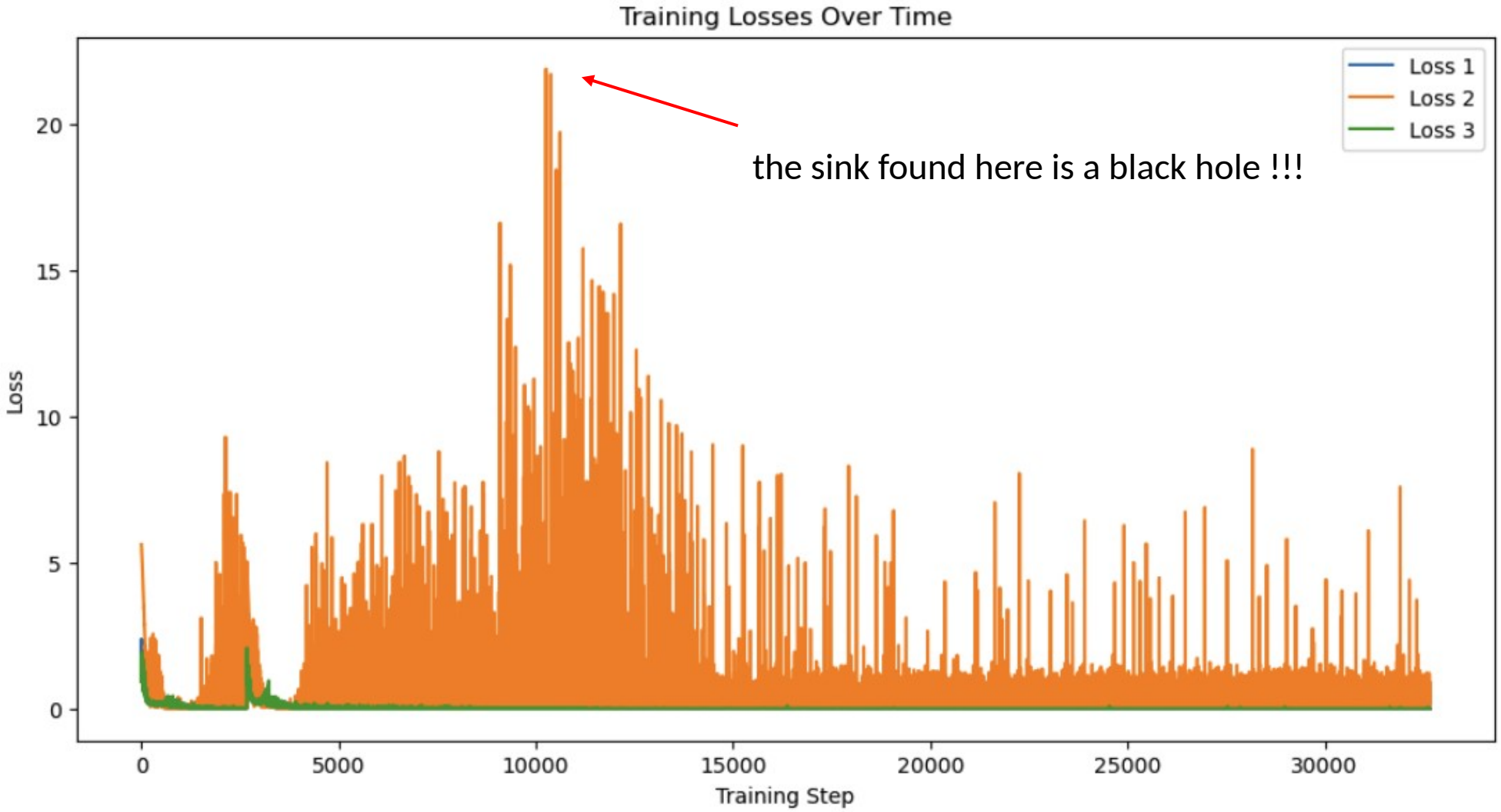
```
Empirical Reward Distribution: {0.1: 0.9951261467889908, 1.1: 0.0047305045871559636, 2.1: 0.00014334862385321102}
  0%|                                                  | 201/90001 [00:51<10:43:05,  2.33it/s]
Empirical Reward Distribution: {0.1: 0.9951026119402985, 1.1: 0.004780783582089552, 2.1: 0.0001166044776119403}
  0%|                                                  | 301/90001 [01:37<10:14:40,  2.43it/s]
Empirical Reward Distribution: {0.1: 0.9932193396226415, 1.1: 0.0066823899371069185, 2.1: 9.827044025157233e-05}
  0%|                                                  | 401/90001 [02:20<11:09:15,  2.23it/s]
Empirical Reward Distribution: {0.1: 0.991593070652174, 1.1: 0.008237092391304348, 2.1: 0.00016983695652173913}
  1%|                                                   | 502/90001 [02:51<5:02:04,  4.94it/s]
Empirical Reward Distribution: {0.1: 0.9916267942583732, 1.1: 0.008223684210526315, 2.1: 0.00014952153110047846}
  1%|                                                   | 602/90001 [03:05<2:30:18,  9.91it/s]
Empirical Reward Distribution: {0.1: 0.9921207264957265, 1.1: 0.007745726495726496, 2.1: 0.00013354700854700856}
```

```
  4%|▊                                                 | 3601/90001 [12:29<5:31:29,  4.34it/s]
Empirical Reward Distribution: {0.1: 0.8612487296747967, 1.1: 0.12006161077235772, 2.1: 0.018610264227642278, 4.1: 1.5879065040650408e-05, 8.1: 1.5879065040650408e-05, 3.1: 4.763719512195122e-05}
  4%|▊                                                 | 3701/90001 [12:52<5:37:35,  4.26it/s]
Empirical Reward Distribution: {0.1: 0.8565721010901883, 1.1: 0.12442703171456888, 2.1: 0.018907953419226957, 4.1: 1.5485629335976214e-05, 8.1: 1.5485629335976214e-05, 3.1: 6.194251734390486e-05}
  4%|▊                                                 | 3802/90001 [13:14<4:59:08,  4.80it/s]
Empirical Reward Distribution: {0.1: 0.8525598404255319, 1.1: 0.12835469052224371, 2.1: 0.01899480174081238, 4.1: 1.5111218568665377e-05, 8.1: 1.5111218568665377e-05, 3.1: 6.044487427466151e-05}
  4%|▊                                                 | 3901/90001 [13:37<6:08:20,  3.90it/s]
Empirical Reward Distribution: {0.1: 0.8482648725212465, 1.1: 0.13252478753541078, 2.1: 0.019121813031161474, 4.1: 1.475448536355052e-05, 8.1: 1.475448536355052e-05, 3.1: 5.901794145420208e-05}
```

```
 40%|████████████████▊                                | 36301/90001 [7:36:44<6:08:52,  2.43it/s]
Empirical Reward Distribution: {1.1: 0.28209, 0.1: 0.66285, 2.1: 0.04266, 3.1: 0.00655, 4.1: 0.00292, 10.1: 0.00013, 6.1: 0.00055, 12.1: 8e-05, 5.1: 0.00085, 7.1: 0.0005, 14.1: 4e-05, 8.1: 0.00019, 11.1
17, 9.1: 0.00027, 15.1: 3e-05, 16.1: 3e-05, 13.1: 6e-05, 20.1: 1e-05, 19.1: 1e-05, 17.1: 1e-05}
 40%|████████████████▊                                | 36401/90001 [7:41:49<6:28:37,  2.30it/s]
Empirical Reward Distribution: {0.1: 0.66401, 1.1: 0.28091, 2.1: 0.04269, 3.1: 0.00657, 6.1: 0.00055, 5.1: 0.00086, 7.1: 0.00051, 10.1: 0.00012, 4.1: 0.00289, 14.1: 4e-05, 8.1: 0.00019, 11.1: 0.00017, 9
0027, 12.1: 7e-05, 15.1: 3e-05, 16.1: 3e-05, 13.1: 6e-05, 20.1: 1e-05, 19.1: 1e-05, 17.1: 1e-05}
 41%|████████████████▉                                | 36501/90001 [7:42:55<6:35:19,  2.26it/s]
Empirical Reward Distribution: {0.1: 0.66566, 1.1: 0.2796, 2.1: 0.0424, 4.1: 0.0029, 14.1: 4e-05, 3.1: 0.00655, 7.1: 0.0005, 8.1: 0.0002, 6.1: 0.00054, 5.1: 0.00084, 11.1: 0.00017, 10.1: 0.00011, 9.1:
12.1: 7e-05, 15.1: 3e-05, 16.1: 3e-05, 13.1: 6e-05, 20.1: 1e-05, 19.1: 1e-05, 17.1: 1e-05}
```

Empirical reward dis

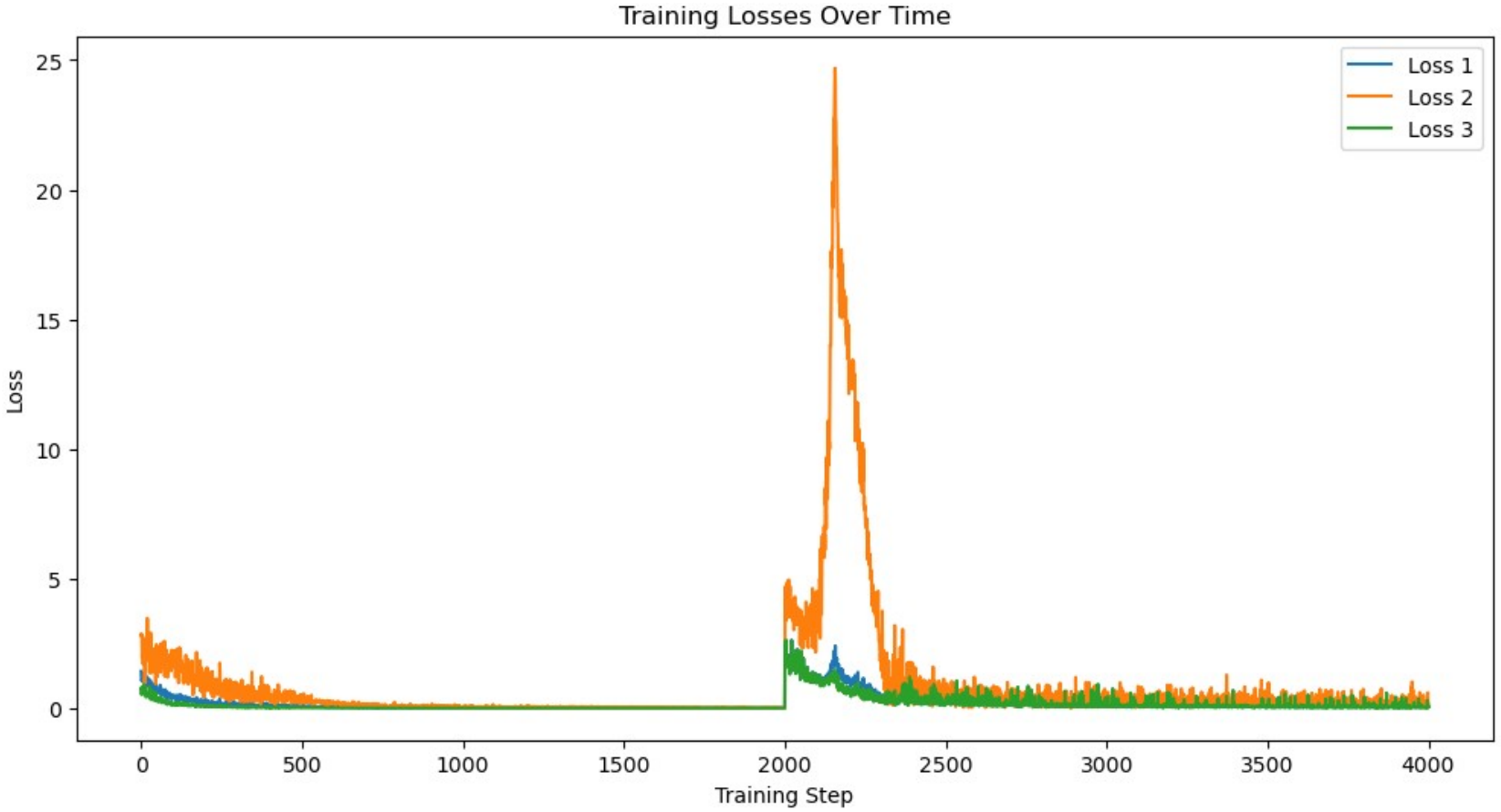
0.1 = basic income = min reward



3 node -> 10 node

choice R(x) = n^10

Problem with Long Trajectories



overfitting to new found sinks very quickly

reward from terminal states don’t propagate to early sates

Trajectory balance:
Improved credit assignment in GFlowNets

trajectory balance,
is computed on sampled full action sequences (trajectories)
from the initial state to a terminal state

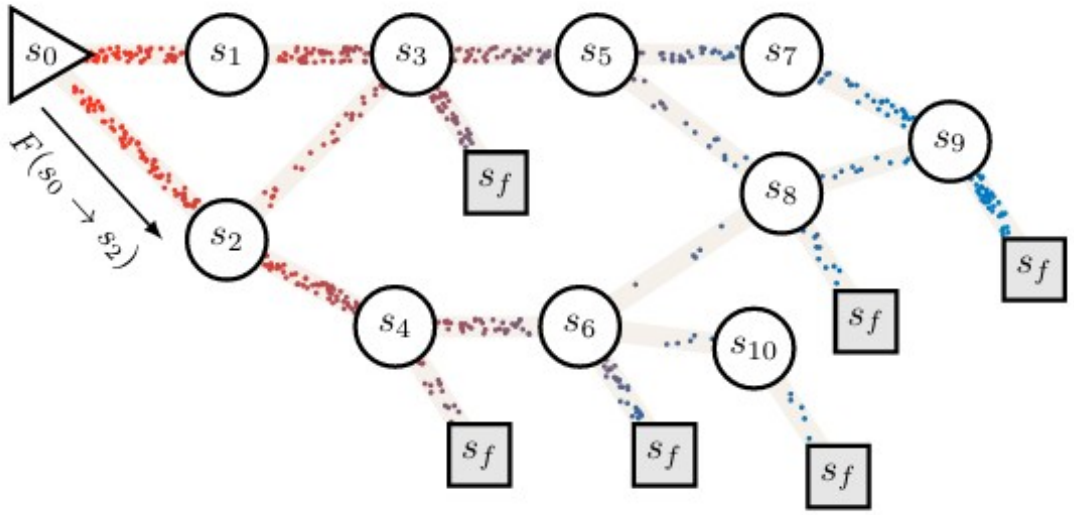
change Obj Func:

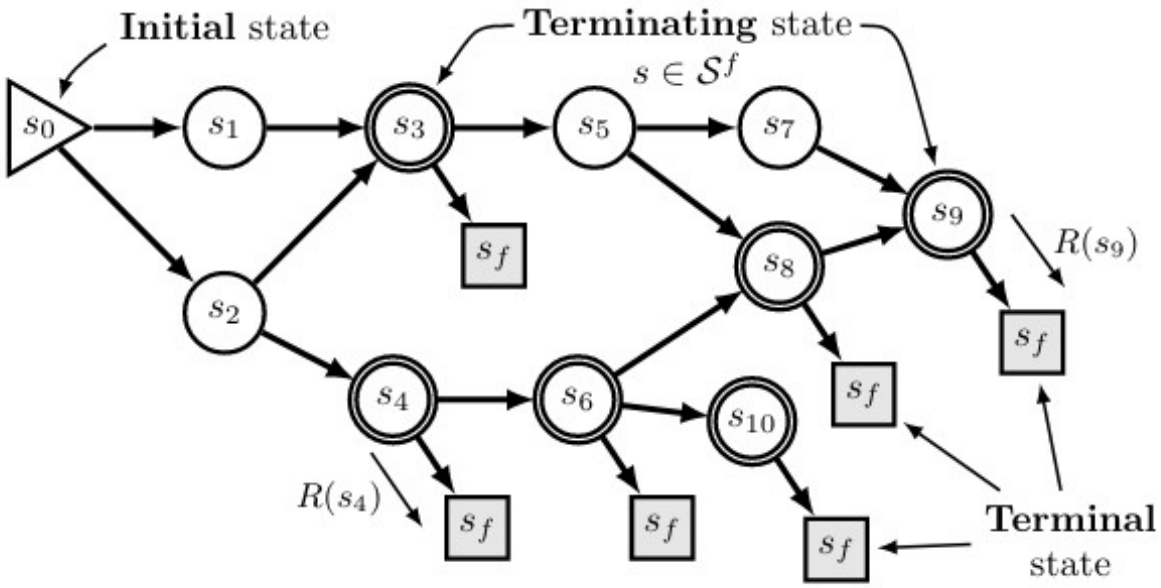
$$Z \prod_{t=1}^{n} P_F(s_t|s_{t-1}) = F(x) \prod_{t=1}^{n} P_B(s_{t-1}|s_t),$$

GFN in short

GFN is a sequential generative model based on flow networks, treating the root as a source and all possible generated objects as sinks, which generates objects proportionally to their reward.

$$p(x) \propto R(x)$$





Flow Conservation !!!

Motivation

Need a better alternative sampling method than MCMC

- MCMC methods are typically slower to converge
- produce samples that are highly correlated
- lots of random-walk, modes (local minima) are not diverse enough.

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

The goal is to have:

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

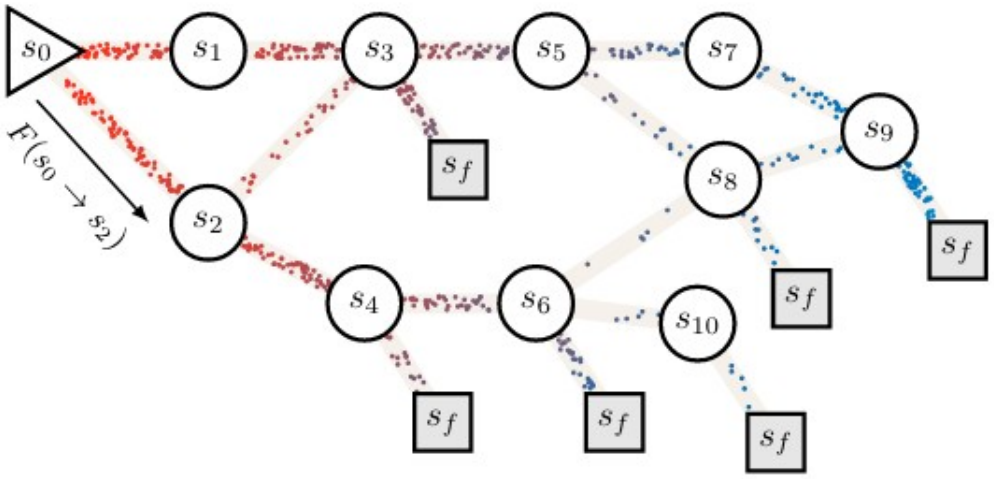
$$p(x) \propto R(x)$$

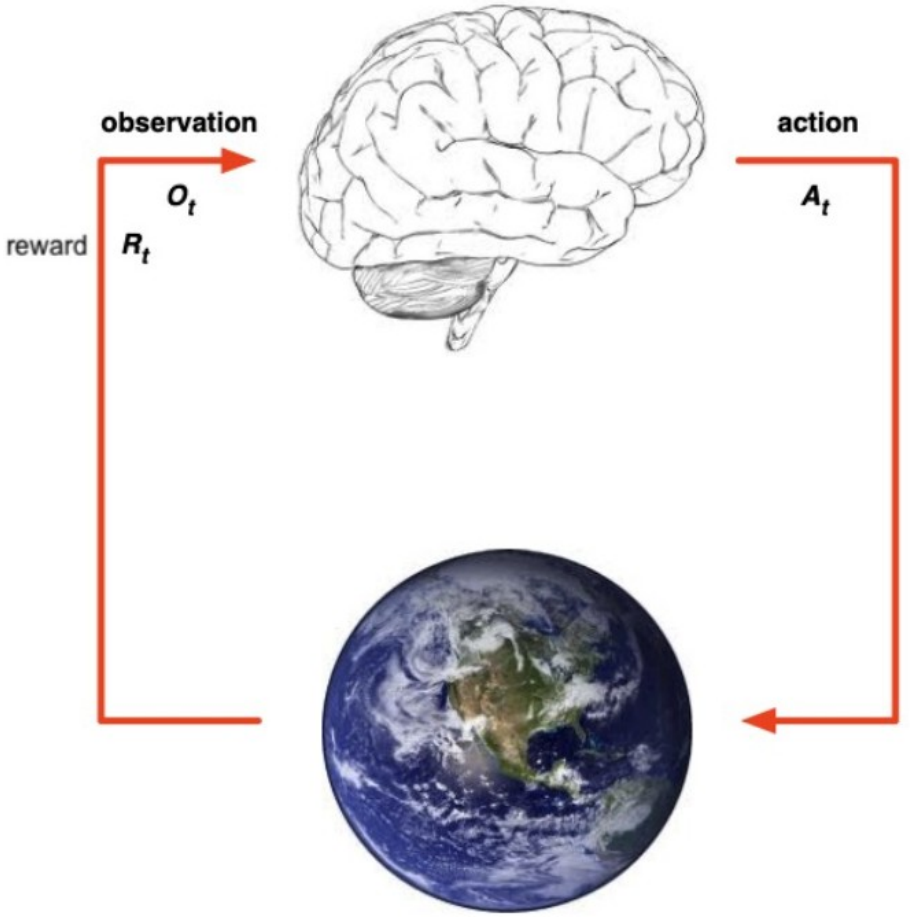
GFN in practice

S: State space

R: Reward

A: Actions

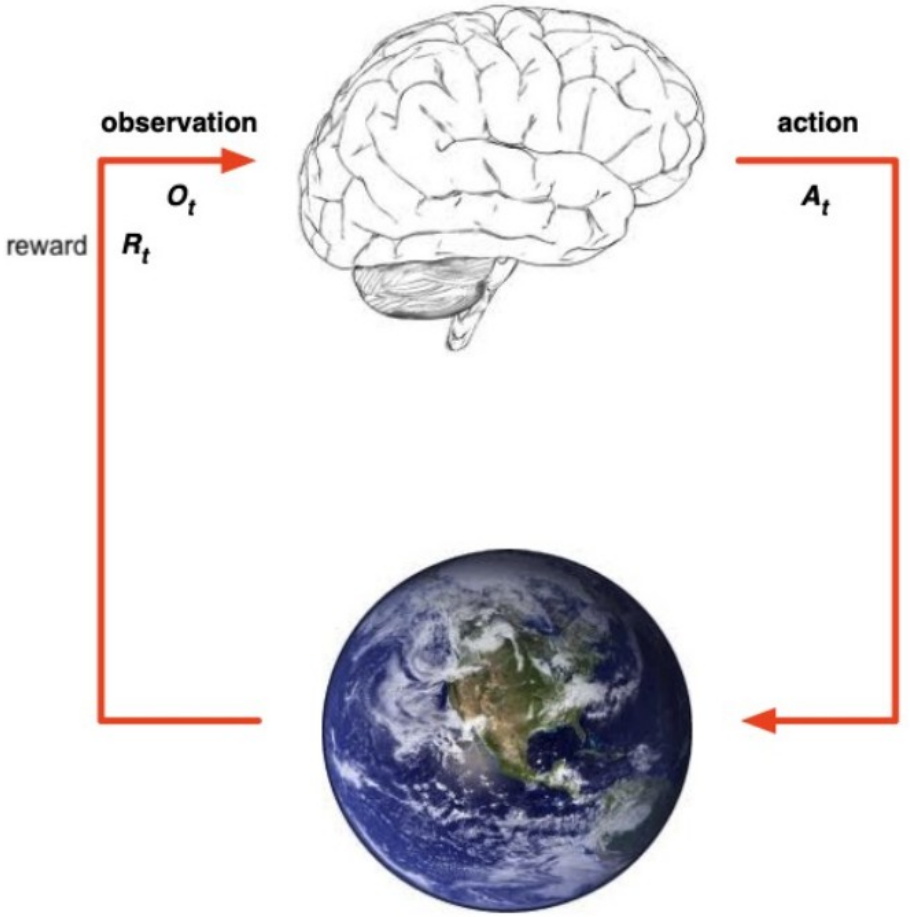




Reinforcement Learning

Goal:

$$\pi^*_{RL} = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{T} r(s_t, s_{t+1}) \right]$$



Reward is defined by:

$$\sum_{t=0}^{T} r(s_t, s_{t+1}) = -\mathcal{E}(s_T)$$

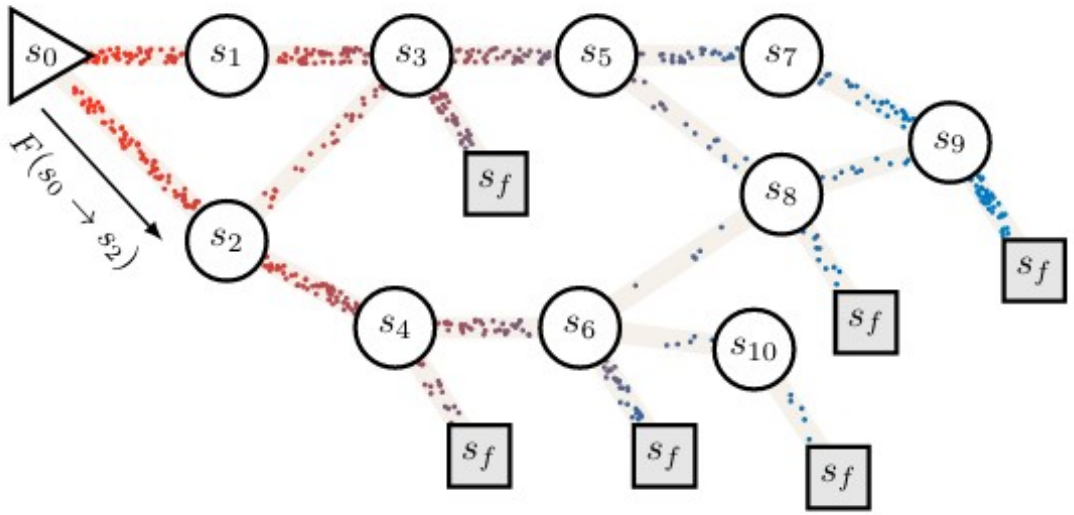
A policy is a distribution over actions given states:

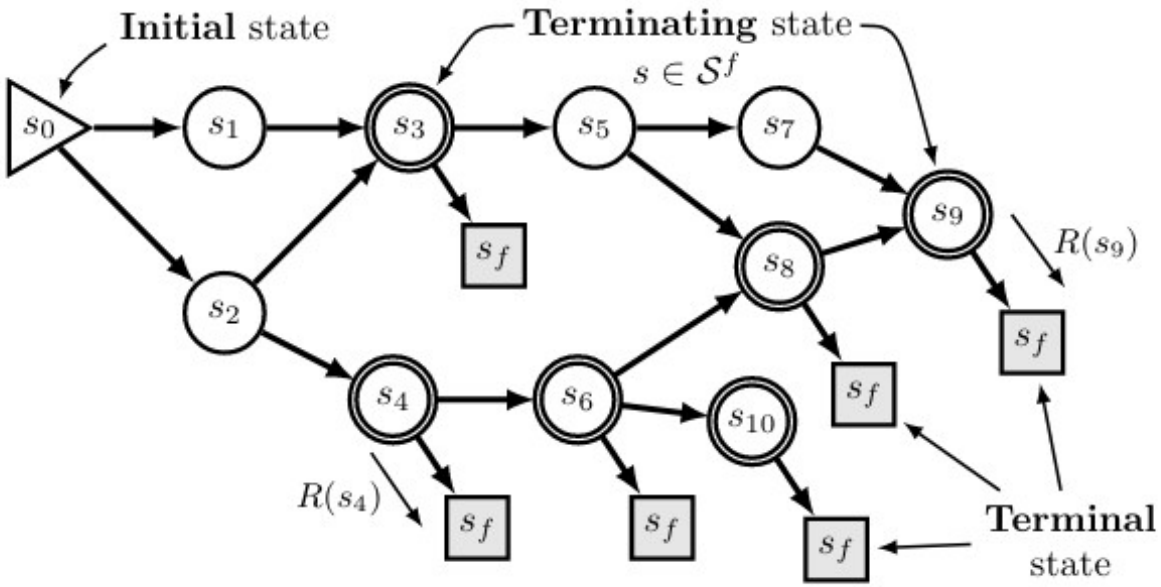
$$\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$$

Back to GFN

GFN is a sequential generative model based on flow networks, treating the root as a source and all possible generated objects as sinks, which generates objects proportionally to their reward.

$$p(x) \propto R(x)$$



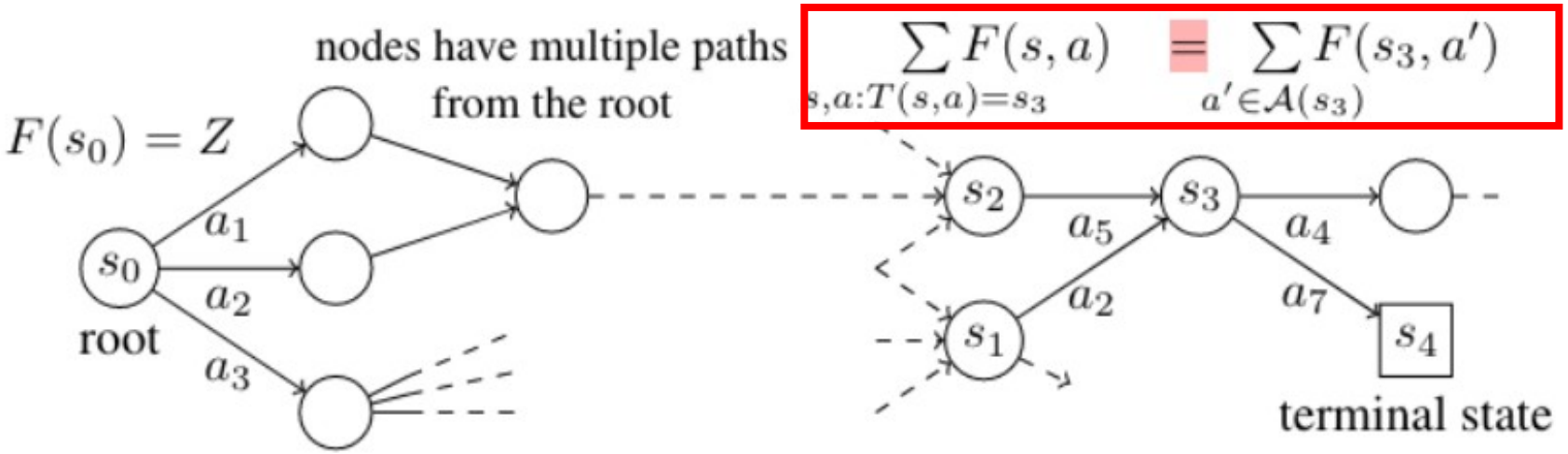


Flow Conservation !!!

Flow Conservation

in-flow = out-flow (flow conservation property)
meaning that the amount of **probability mass** flowing into a state equals the amount flowing out, which is crucial for generating valid samples from the target distribution.

s_0: a single source of the water, the root
x_i: the sinks, terminal states. We'll assign to each sink x an out-flow R(x).
$\sum_x R(x) = Z$: the flow out of the unique source s_0, the partition function.



Flow Conservation

in-flow = out-flow :

$$\sum_{s,a:T(s,a)=s'} F(s,a) = F(s') = \sum_{a' \in \mathcal{A}(s')} F(s',a')$$

$R(s) = 0$ for interior nodes

=>

$$\sum_{s,a:T(s,a)=s'} F(s,a) = R(s') + \sum_{a' \in \mathcal{A}(s')} F(s',a')$$

Training of GFN

Review of ML:

The goal is to find f such that $y = f(x)$

f can be parametrized by ... $y = f_\theta(x)$

$Loss = y_0 - f(x_0, \theta)$ —GD—> θ_0

theth -> Loss

$y = f_{\theta_0}(x) = f(x)$

GFN:

The goal is to find $F(s, a)$ such that $p_{mass} = F(s, a)$

$p_{mass} = F_\theta(s, a)$

Loss = ...

Flow Matching Obj Func

Objective Functions for GFlowNet

$$\tilde{L}_\theta(\tau) \equiv \sum_{s' \in \tau \neq s_0} \left(\sum_{s,a:T(s,a)=s'} F_\theta(s,a) - R(s') - \sum_{a' \in \mathcal{A}(s')} F_\theta(s',a') \right)^2 .$$

Flow Predictor $F_\theta(s,a)$:

- F_θ is the neural network parameterized by θ that predicts these flows.
- For a given state s and action a, the flow predictor $F_\theta(s,a)$ predicts the probability of taking action a in state s to transition to the next state $s\prime$.
- The flow predictor is trained to ensure that the flow conservation property is satisfied.

Flow Matching Obj Func

Objective Functions for GFlowNet

$$\tilde{L}_\theta(\tau) \equiv \sum_{s' \in \tau \neq s_0} \left(\sum_{s,a:T(s,a)=s'} F_\theta(s,a) - R(s') - \sum_{a' \in \mathcal{A}(s')} F_\theta(s',a') \right)^2 .$$

Flow Predictor $F_\theta(s,a)$:

- F_θ is the neural network parameterized by θ that predicts these flows.
- For a given state s and action a, the flow predictor $F_\theta(s,a)$ predicts the probability of taking action a in state s to transition to the next state $s\prime$.
- The flow predictor is trained to ensure that the flow conservation property is satisfied.

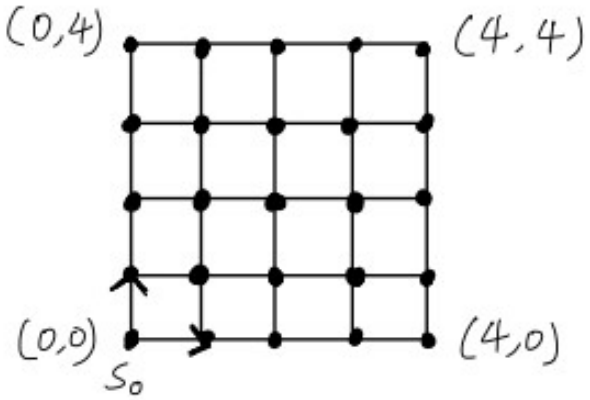
But this is a bad Objective Function …

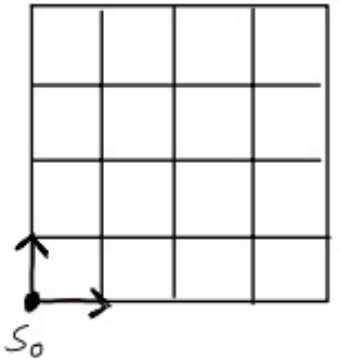
Problem with Long Trajectories

instead of 2d --> 100d

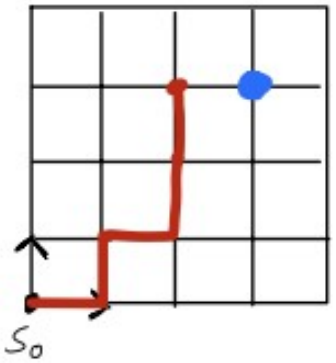
→ if width = 8 then size(s space) = 8^100

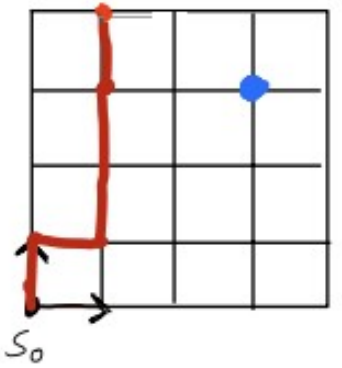
→ to reach (8, 8, 8, 8 .....) will take 8 * 100 steps !!!



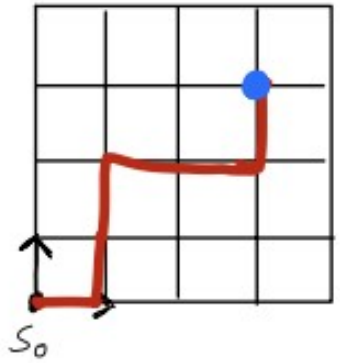


State 19: (0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0), Reward: 0.6

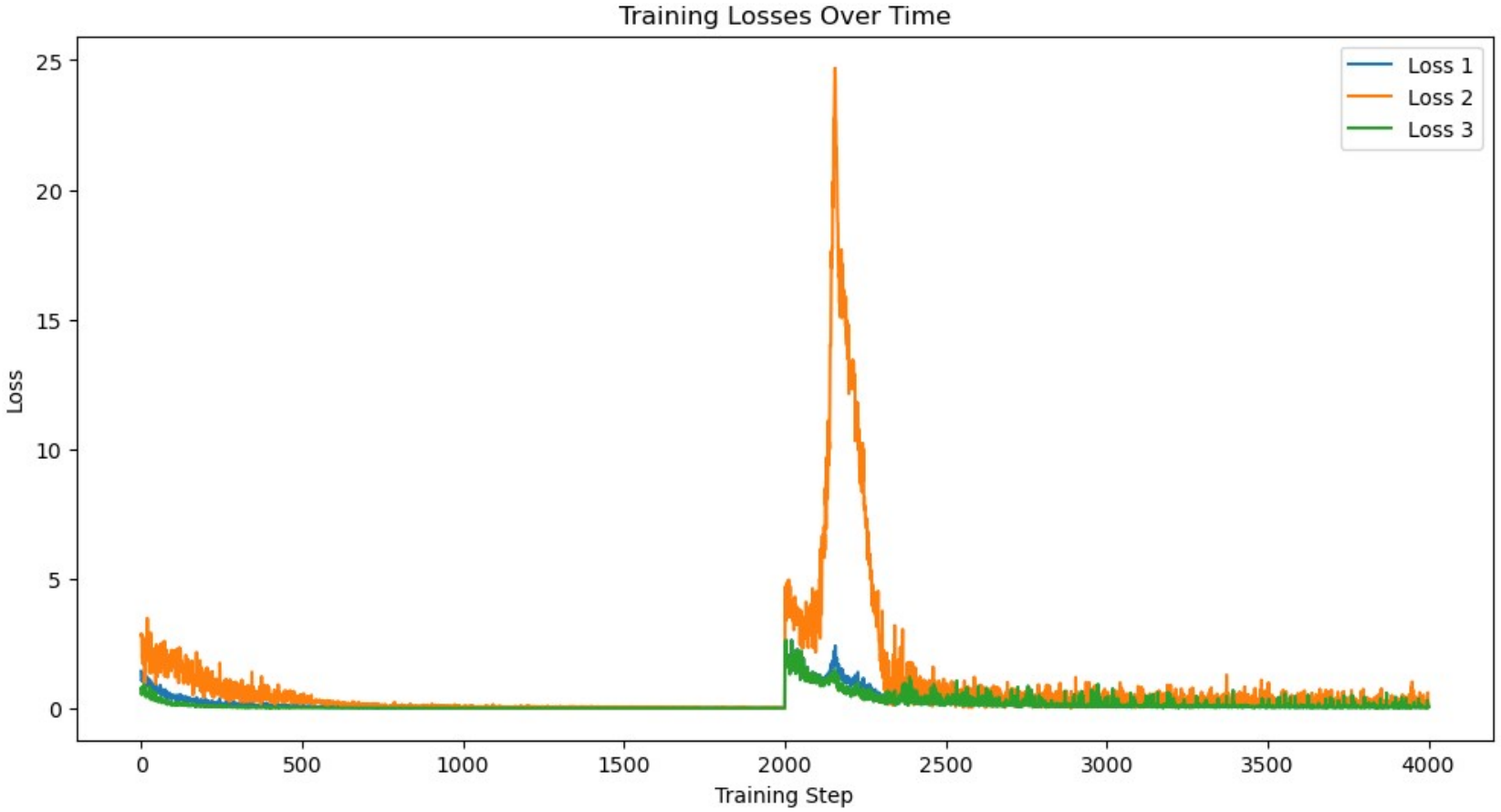




State 289: (8, 8, 8, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 8,
8, 8, 0, 0, 8, 8, 0, 0, 0, 8, 8, 0, 0, 8, 8, 0, 0, 0, 8,
8, 8, 0, 0, 8, 8, 0, 8, 0, 8, 8, 0, 0, 8, 8, 0, 0, 0, 8,
8, 8, 0, 0, 8, 8, 0, 0, 0, 8, 8, 0, 0, 8, 8, 0, 0, 0,
8,), Reward: 2.6



Problem with Long Trajectories



overfitting to new found sinks very quickly

reward from terminal states don’t propagate to early sates

TB Obj Function

Trajectory balance: Improved credit assignment in GFlowNets

trajectory balance,
is computed on sampled full action sequences (trajectories)
from the initial state to a terminal state

previously:

$$F(s) = \sum_{(s'' \to s) \in \mathcal{A}} F(s'' \to s) = \sum_{(s \to s') \in \mathcal{A}} F(s \to s').$$

FM obj

TB Obj Function

Trajectory balance: Improved credit assignment in GFlowNets

trajectory balance,
is computed on sampled full action sequences (trajectories)
from the initial state to a terminal state

now:

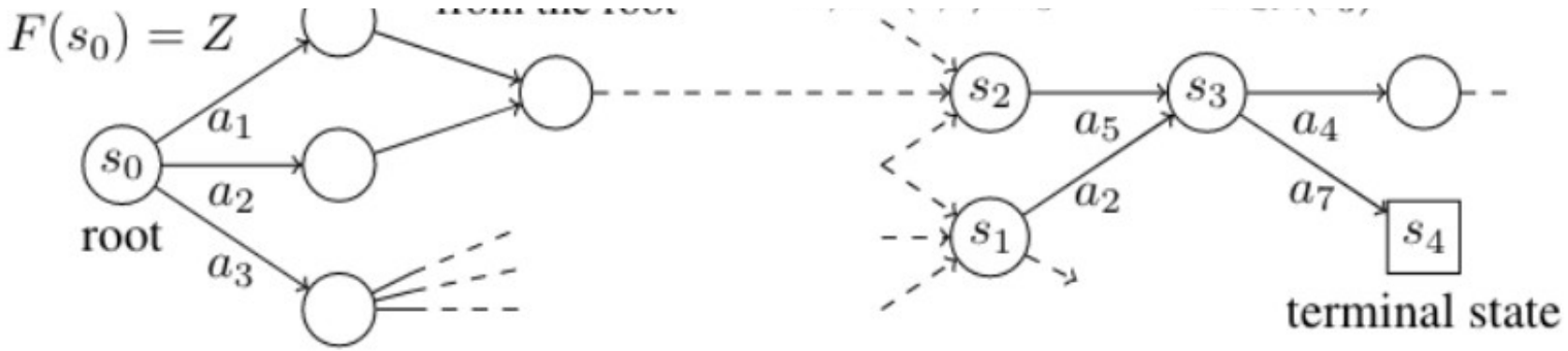
$$Z \prod_{t=1}^{n} P_F(s_t|s_{t-1}) = F(x) \prod_{t=1}^{n} P_B(s_{t-1}|s_t),$$

Trajectory balance: Improved credit assignment in GFlowNets

$$P_F(s'|s) = \frac{F(s \to s')}{F(s)}, \quad P_B(s|s') = \frac{F(s \to s')}{F(s')},$$

$$P(\tau = (s_0 \to \ldots \to s_n)) = \prod_{t=1}^{n} P_F(s_t|s_{t-1}).$$

$$P(\tau = (s_0 \to \ldots \to s_n)|s_n = x) = \prod_{t=1}^{n} P_B(s_{t-1}|s_t).$$



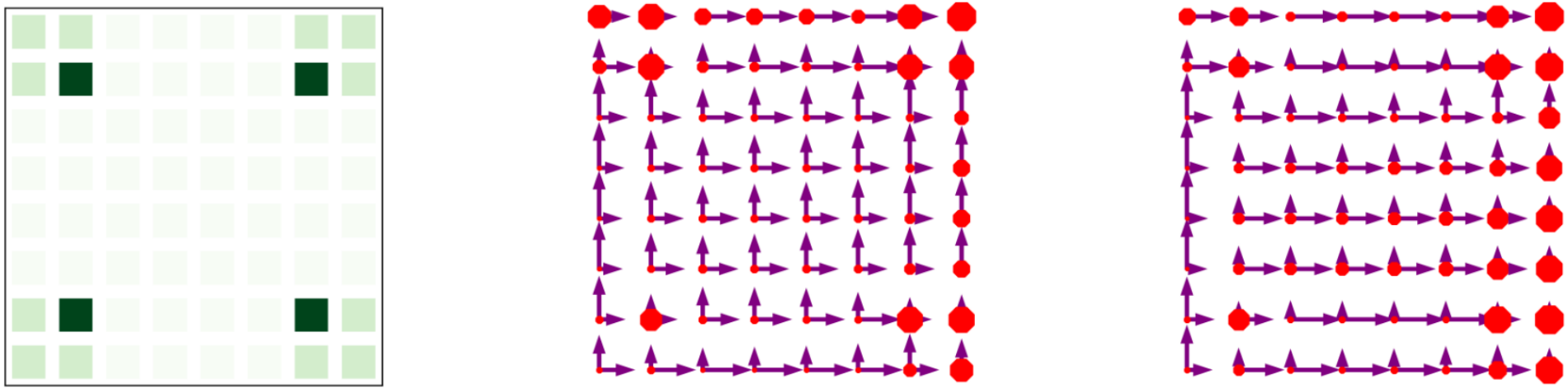


Figure 1: *Left:* The reward function on an 8×8 grid environment (§5.1) with $R_0 = 0.1$. *Centre and right:* Two forward action policies – with fixed uniform P_B and with a learned non-uniform P_B – that sample from this reward. The lengths of arrows pointing up and right from each state are proportional to the likelihoods of the corresponding actions under P_F, and the sizes of the red octagons are proportional to the termination action likelihoods.

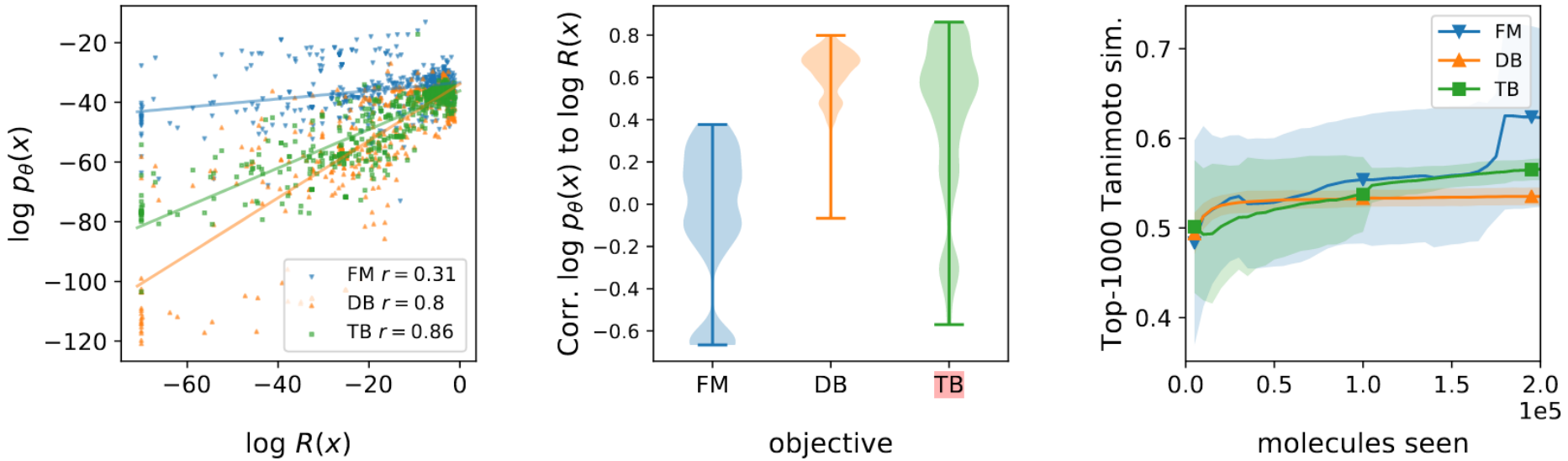
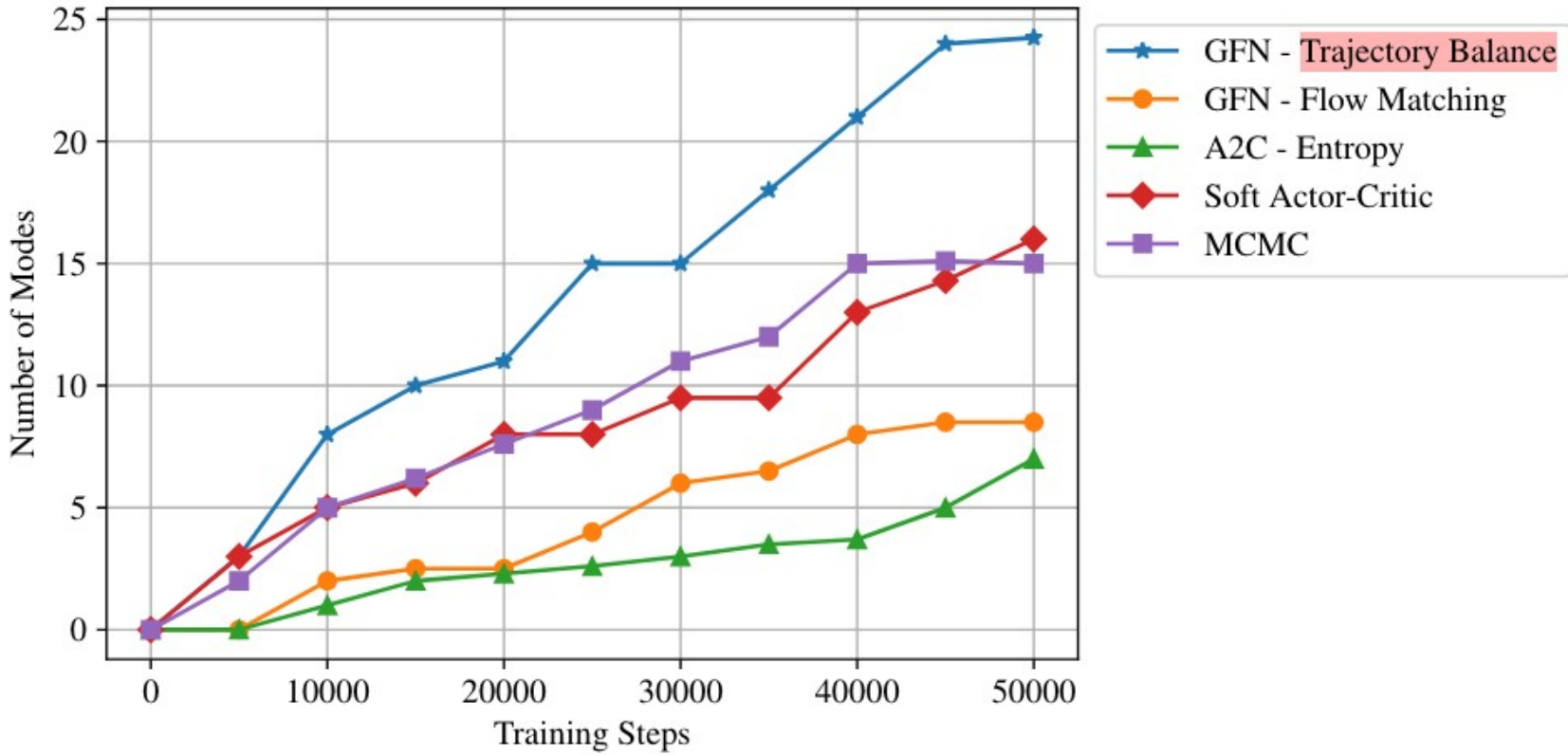


Figure 3: *Left, Centre:* Pearson correlations between rewards and sampling probability. $\log p_\theta(x)$ is the log-likelihood that a trajectory sampled from the learned policy $P_F(-|-;\theta)$ terminates at x. *Left:* Scatter plot on a test set of x's for the best hyperparameters of TB, FM, and DB. *Centre:* Violin plot of correlations for 16 hyperparameter settings and 3 seeds for each setting, showing TB being capable of fitting better. *Right:* Average pairwise Tanimoto similarity for the top 1000 samples generated by GFlowNets as training progresses. Lines are the average across runs, shaded regions the standard deviation. Models trained with TB have consistently lower similarity than those with FM, hence greater diversity. We hypothesize that the higher variance, in correlation and diversity, of TB relative to DB is related to high variance of the stochastic gradient; see [18].


25
20
15
10
5
0
Number of Modes
0
10000
20000
30000
40000
50000
Training Steps
GFN - Trajectory Balance
GFN - Flow Matching
A2C - Entropy
Soft Actor-Critic
MCMC

6 Discussion and conclusion

We introduced a novel training loss for GFlowNets, trajectory balance (TB), which yields faster and better training than the previously proposed flow matching (FM) and detailed balance (DB) losses. We proved that this objective, when minimized, yields the desired GFlowNet property of sampling from the target distribution specified by an unnormalized reward function. This new loss was motivated by the observation that the FM and DB losses are local in the action sequence and may require many iterations for credit assignment to propagate to early actions: if a gradient update introduces a flow inconsistency at some state far from the initial state (such as when a novel high-reward state is sampled), the parent of this state must be visited before the update is propagated closer to the root, akin to the slow propagation of reward signals in temporal difference learning.

We empirically found that TB discovered more modes of the energy function faster and was more robust than FM and DB to the exponential growth of the state space, due in part to the lengths of sequences and in part to the size of the action space. A factor to consider when interpreting our experimental results is that because we use a neural net rather than a tabular representation of policies, the early states' transitions are informed by downstream credit assignment via parameter sharing. Early states also get many more visits because there are more possible states near the ends of sequences than near the initial state. Finally, TB trades off the advantage of immediately providing credit to early states with the disadvantage of relying on sampling of long trajectories and thus a potentially higher variance of the stochastic gradient. The high gradient variance is a possible limitation of TB in difficult environments, and ways to overcome it by interpolating between local and trajectory-level objectives have been studied in subsequent work [18].

Trajectory balance:
Improved credit assignment in GFlowNets

So TB is the way to go?

Trajectory balance:
Improved credit assignment in GFlowNets

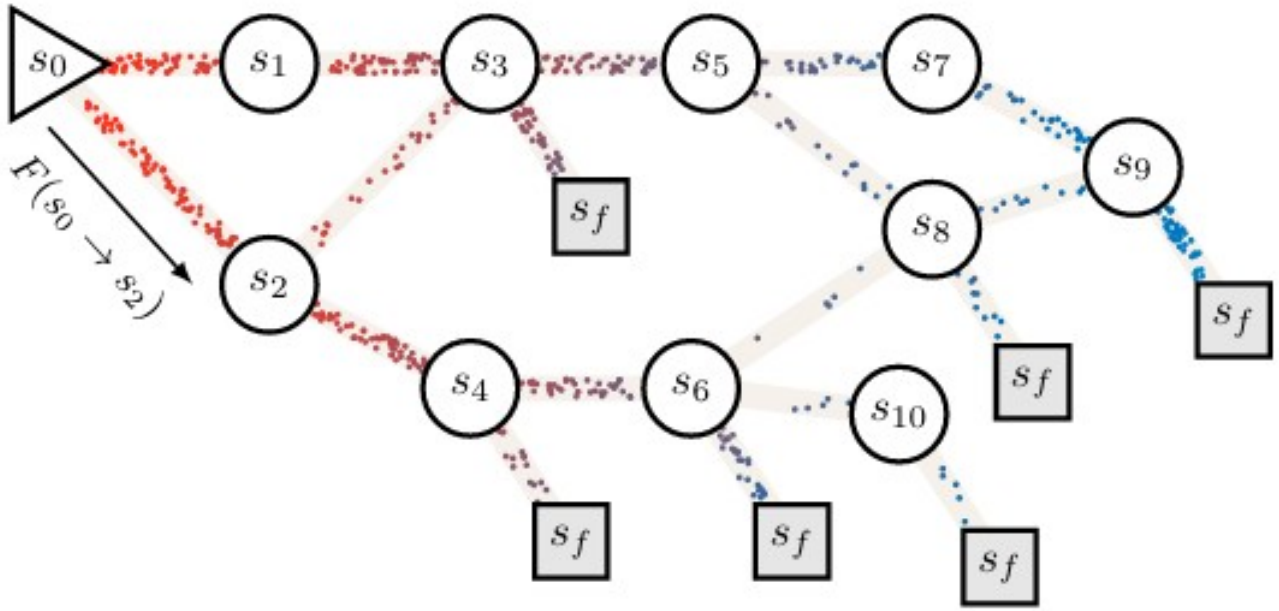
SO TB is the way to go?

Not Really…

Trajectory balance:
Improved credit assignment in GFlowNets

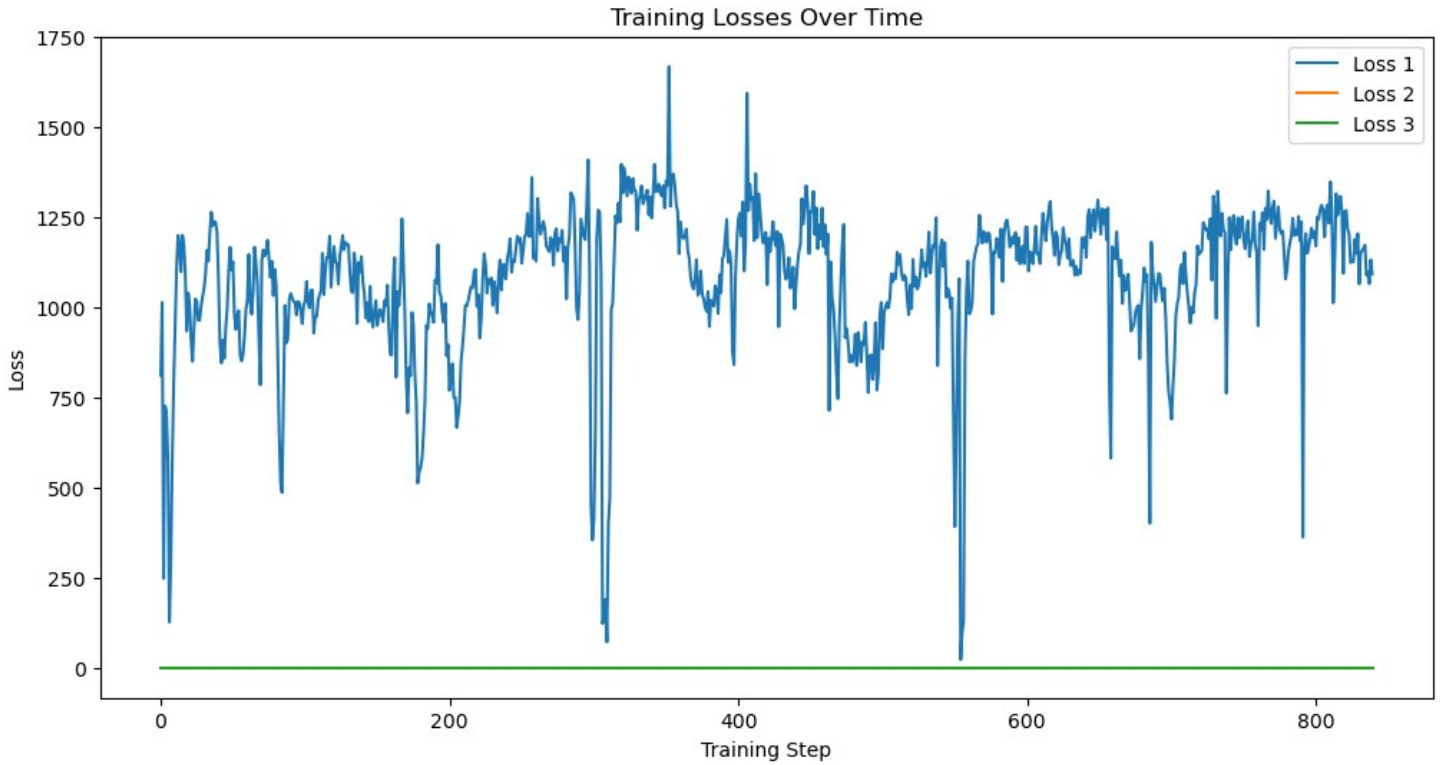
SO TB is the way to go?

Not Really…



This is also a bad Objective Function …

Trajectory balance:
Improved credit assignment in GFlowNets



High Loss
not converging
Large variane
don’t fit well to new sinks

This is also a bad Objective Function …

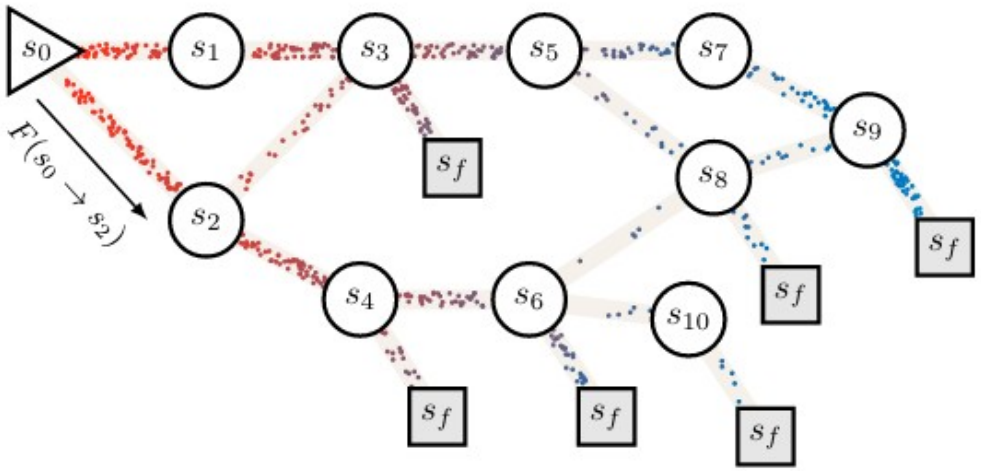
Trajectory balance:
Improved credit assignment in GFlowNets

Learning GFlowNets From Partial Episodes For Improved Convergence And Stability

aka. sub-TB(λ)

FM: local sate

TB: entire sampling trajectory.



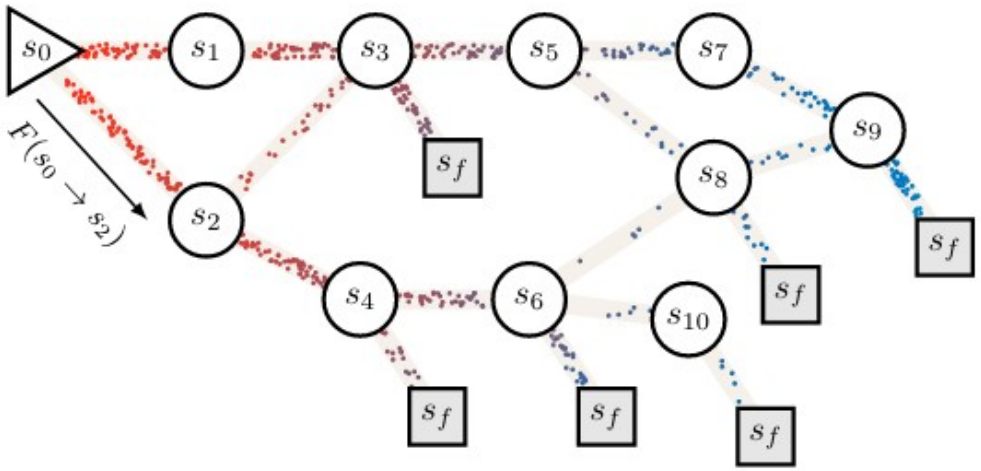
2.3. Subtrajectory balance: Learning from partial episodes

Recall the GFlowNet parametrization used in the DB objective above, with a state flow estimator $F(-|-;\theta)$ and a pair of policies $P_F(-|-;\theta)$, $P_B(-|-;\theta)$. It is shown in §A.2 of Malkin et al. (2022) that the detailed balance conditions (6) are satisfied for all actions if and only if the following *sub-trajectory balance* constraint holds for all (not necessarily complete) trajectories $\tau = (s_m \to \ldots \to s_n)$:

$$F(s_m;\theta)\prod_{i=m}^{n-1} P_F(s_{i+1}|s_i;\theta) = F(s_n;\theta)\prod_{i=m}^{n-1} P_B(s_i|s_{i+1};\theta),$$

Hypothesized benefits.:

- Variance reduction.
- - Faster learning due to generalization of state flows.



subtrajectories:

$$\tau_{i:j} := (s_i \to s_{i+1} \to \ldots \to s_j), \quad 0 \le i < j \le n. \tag{10}$$

Having sampled a complete trajectory τ for training, we make gradient steps on a convex combination of the subtrajectory balance losses $\mathcal{L}_{\text{SubTB}}(\tau_{i:j})$: $\theta \leftarrow \theta - \nabla_\theta \mathcal{L}$, where

$$\mathcal{L} = \frac{\sum_{0 \le i < j \le n} \lambda^{j-i} \mathcal{L}_{\text{SubTB}}(\tau_{i:j})}{\sum_{0 \le i < j \le n} \lambda^{j-i}}. \tag{11}$$

Here $\lambda > 0$ is a hyperparameter controlling the weights assigned to subtrajectories of different lengths, and when λ is set to 1, it leads to a uniform weighting scheme. Notice that the $\lambda \to 0^+$ limit leads precisely to the average detailed balance loss $\mathcal{L}_{\text{DB}}(s_i \to s_{i+1})$ over all transitions in τ, while the $\lambda \to +\infty$ limit gives the trajectory balance objective $\mathcal{L}_{\text{TB}}(\tau)$.[1]

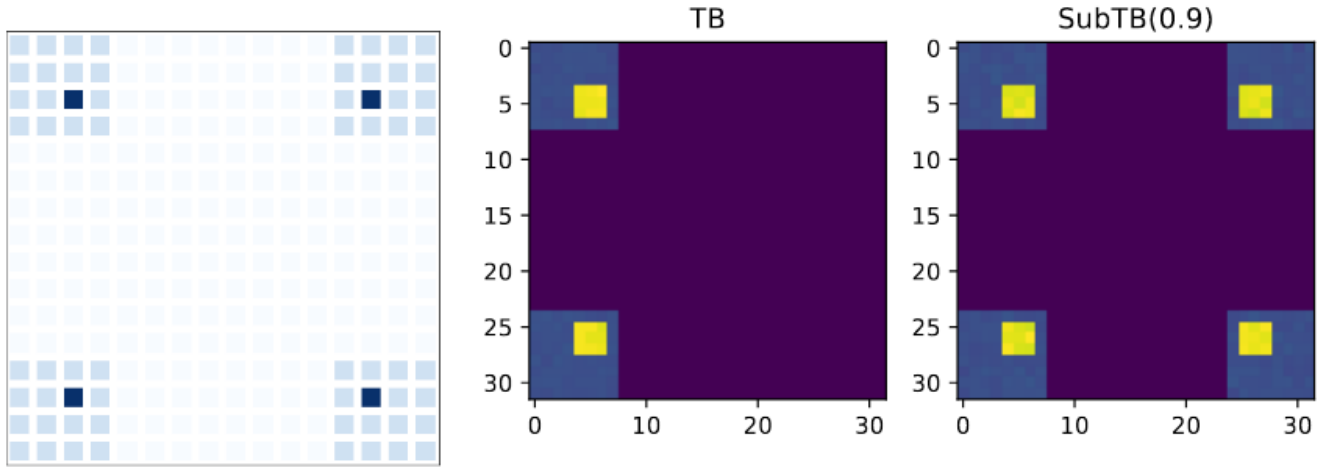


Figure 1. **Left:** 16×16 hypergrid reward function. **Right:** Distribution of 2×10^5 samples from GFlowNets trained on the harder variant of the 32×32 grid with TB and SubTB(λ) objectives.

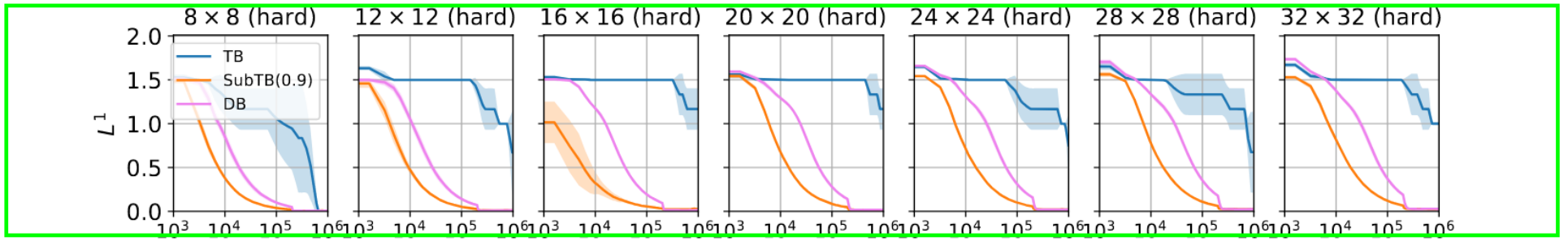
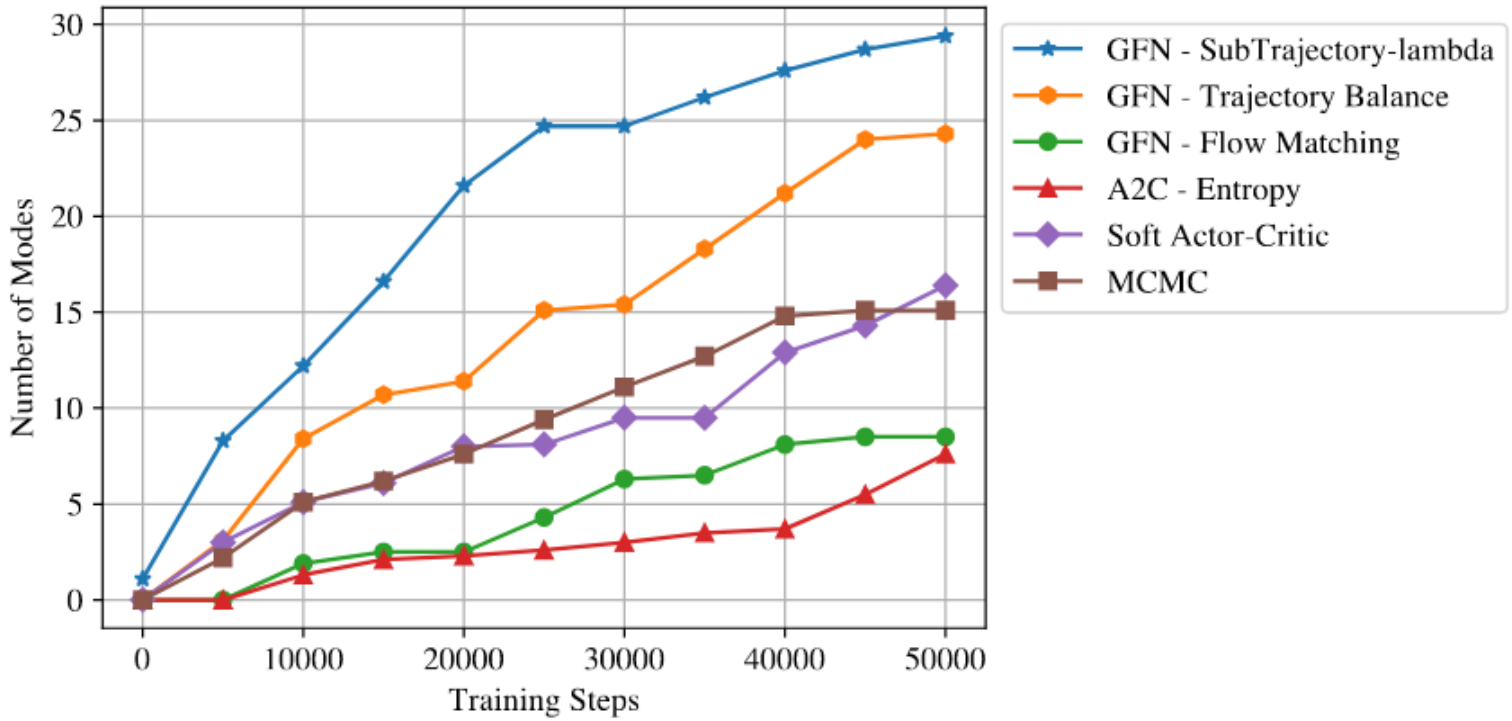
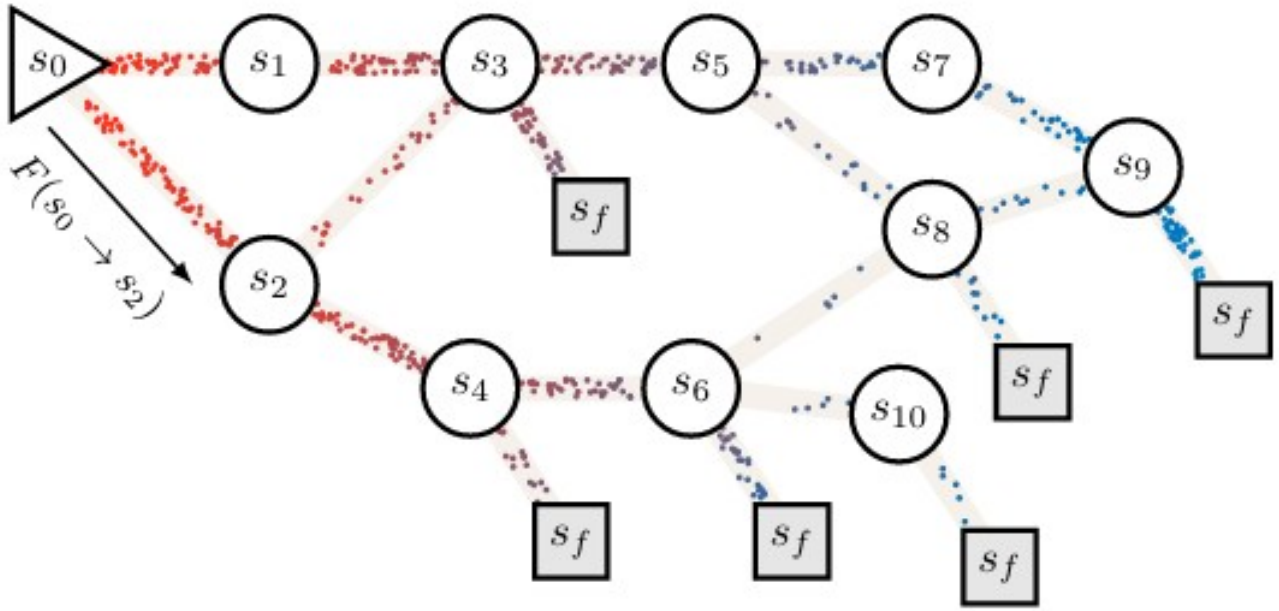


Figure 2. L^1 distance between empirical and target distributions over the course of training on the hypergrid environment. SubTB($\lambda = 0.9$) consistently gives faster convergence than TB, the strongest objective from past work, on all grid sizes. The difference is especially visible for the harder variant of the reward function (last row). The x-axis is the cumulative number of training trajectories (episodes).



Learning GFlowNets From Partial Episodes For Improved Convergence And Stability

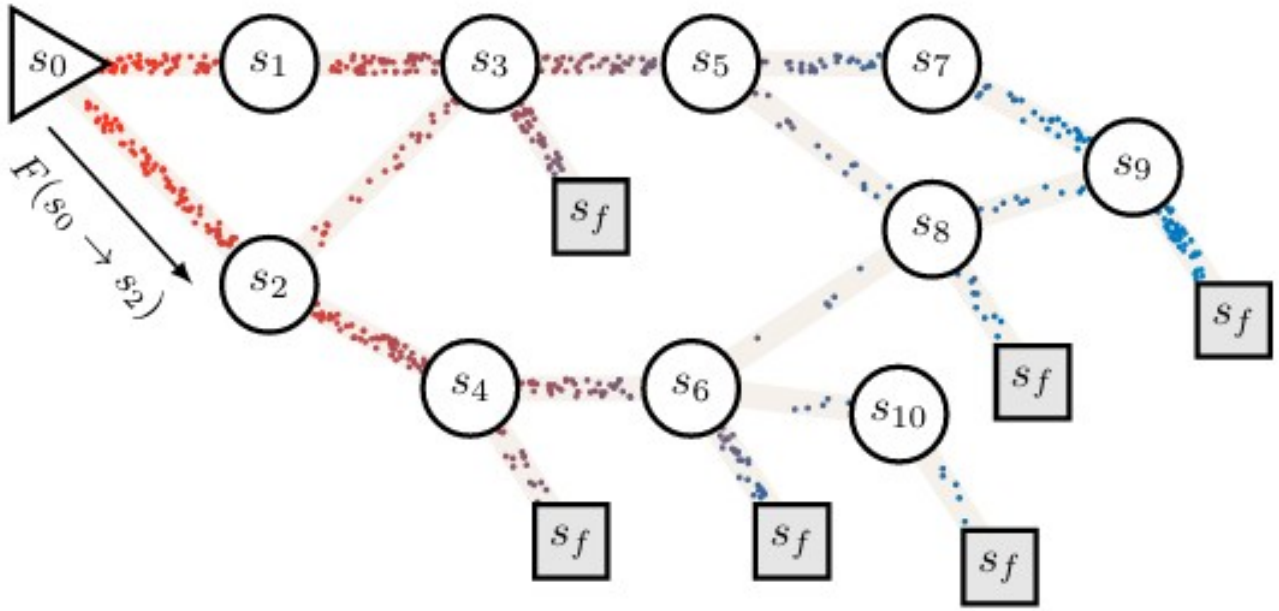
SO sub-TB(λ) is the way to go?



Learning GFlowNets From Partial Episodes For Improved Convergence And Stability

SO sub-TB(λ) is the way to go?

Not Really…



This is also a bad Objective Function …

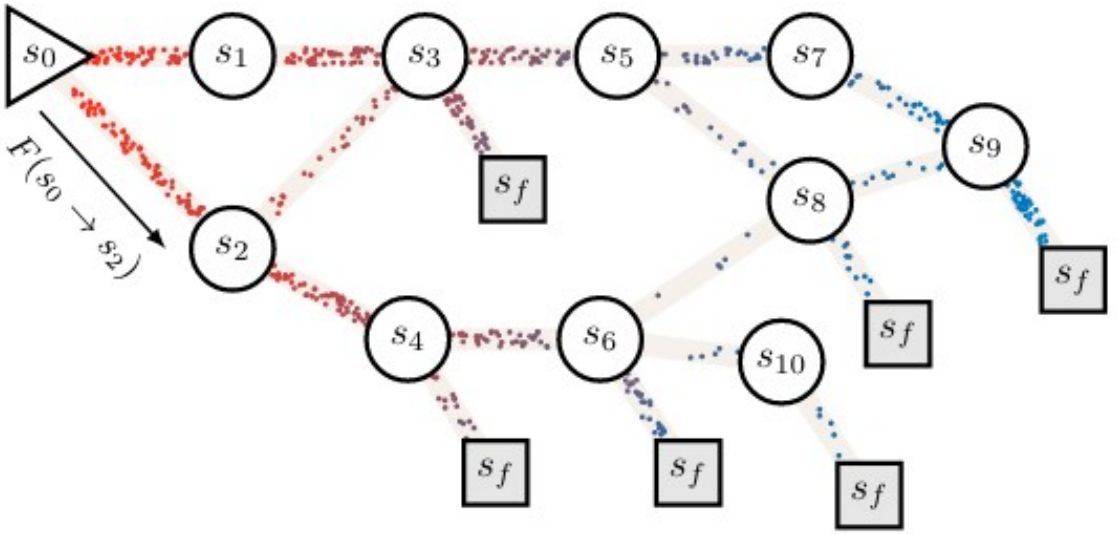
Learning GFlowNets From Partial Episodes For Improved Convergence And Stability

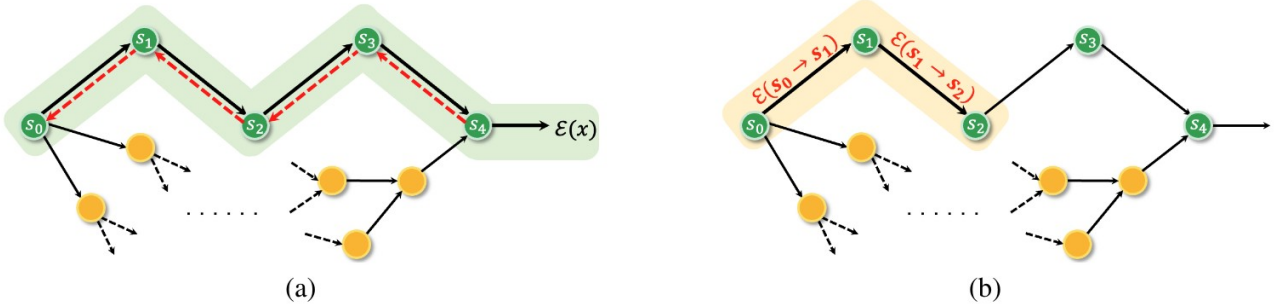
Better Training of GFlowNets with Local Credit and Incomplete Trajectories

Learning GFlowNets From Partial Episodes For Improved Convergence And Stability

Better Training of GFlowNets with Local Credit and Incomplete Trajectories

add energy at intermediate states, or equivalently, an additive energy decomposition





(a) (b)

Figure 1. (a) Reward propagation in previous learning objectives of GFlowNets and (b) in FL-GFN, with additive energy terms for each transition. Note that (a) requires learning based on complete trajectories since it only learn from the terminal energy $\mathcal{E}(x)$, while (b) makes it possible to learn from short incomplete trajectories.

Assumption 4.1. The terminal state energy function $\mathcal{E} : \mathcal{X} \to \mathbb{R}$ can be extended to the set of all states, i.e., there exists $\mathcal{E} : \mathcal{S} \to \mathbb{R}$.

As a consequence, we can also define an energy function over transitions:

$$\mathcal{E}(s \to s') \stackrel{\text{def}}{=} \mathcal{E}(s') - \mathcal{E}(s). \tag{4}$$

Similarly, we can define the energy differential associated with all trajectories from s to x as

$$\mathcal{E}(s \to x) \stackrel{\text{def}}{=} \mathcal{E}(x) - \mathcal{E}(s). \tag{5}$$

We obtain that the energy of a state s_t (terminal or not) can be written additively over transitions for any trajectory $s_0 \to s_1 \to \cdots \to s_n$ if we define $\mathcal{E}(s_0) \stackrel{\text{def}}{=} 0$:

$$\mathcal{E}(s_t) = \sum_{i=1}^{n} \mathcal{E}(s_{i-1} \to s_i). \tag{6}$$

4.2. Forward-looking GFlowNets (FL-GFN)

Consider the flow at s and exploit Assumption 4.1 to rewrite the terminal energy $\mathcal{E}(x)$ as $\mathcal{E}(s) + \mathcal{E}(s \to x)$:

$$F(s) = \sum_{x \geq s} P_B(s|x) e^{-\mathcal{E}(x)} \tag{7}$$

$$= e^{-\mathcal{E}(s)} \sum_{x \geq s} P_B(s|x) e^{-\mathcal{E}(s \to x)}. \tag{8}$$

Here $P_B(s|x)$ denotes the probability of reaching the intermediate state s from the terminal state x via the one-step backward policy $P_B(s|s')$. The idea is to take advantage of the fact that we already know $\mathcal{E}(s)$ when we have reached state s and that we can thus factor it out of the right-hand side, as above. With this in mind, we define the **forward-looking flow** as

$$\widetilde{F}(s) \stackrel{\text{def}}{=} e^{\mathcal{E}(s)} F(s) = \sum_{x \geq s} P_B(s|x) e^{-\mathcal{E}(s \to x)}, \tag{9}$$

i.e., $\widetilde{F}(s)$ only contains a sum over terms with energies of transitions to come (hence the name of forward-looking flow). Using this, we can write a variant of the detailed balance constraint (Eq. (1)), called the FL-DB constraint, expressed in terms of forward-looking flows:

$$\widetilde{F}(s) P_F(s'|s) = \widetilde{F}(s') P_B(s|s') e^{-\mathcal{E}(s \to s')}. \tag{10}$$

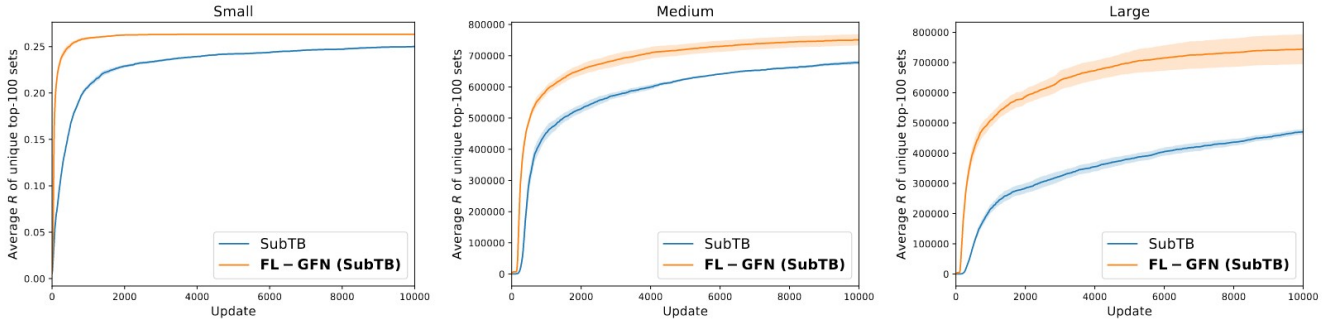
Implementation. In practice, we train the FL-GFN based on the FL-DB constraint in Eq. (10), following Algorithm 1, to minimize the corresponding loss function over transitions $s \to s'$ in the log-space:

$$\mathcal{L}(s, s') = \Big(\log \widetilde{F}(s;\theta) + \log P_F(s'|s;\theta) - \log \widetilde{F}(s';\theta) - \log P_B(s|s';\theta) + \mathcal{E}(s \to s')\Big)^2. \tag{11}$$

An alternative implementation is to make the following substitution and optimize for the DB constraint Eq. (1):

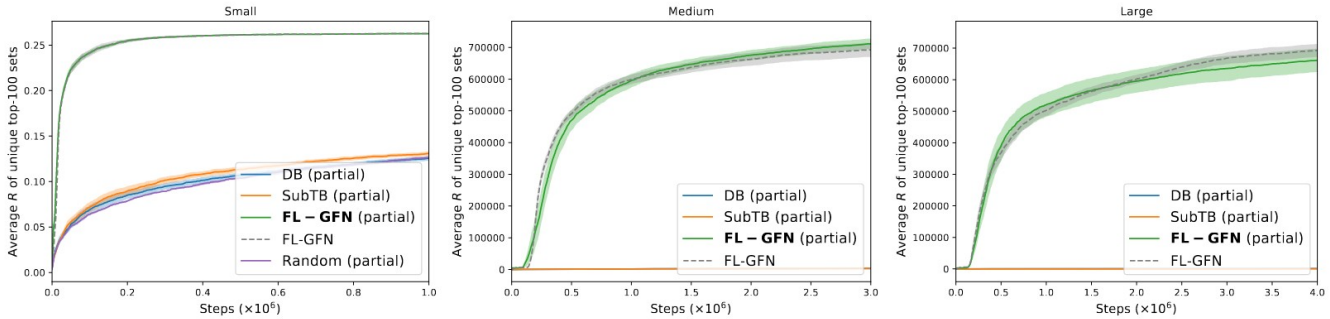
$$\log F(s) = -\mathcal{E}(s) + \log \tilde{F}(s;\theta), \tag{12}$$

where $\log \tilde{F}(s;\theta)$ is a learned model.



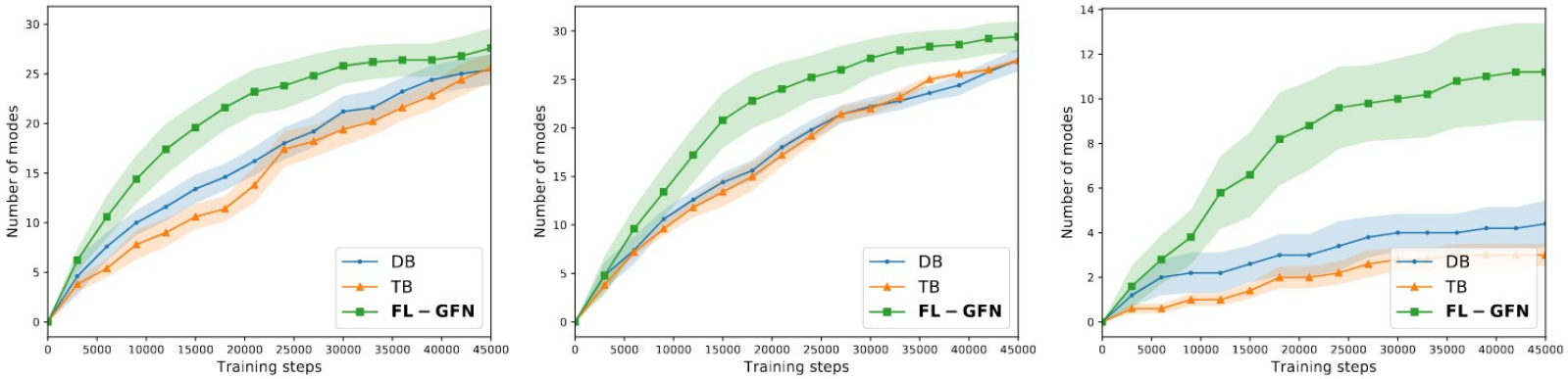
(a) (b) (c)

Figure 3. Performance comparison of FL-GFN (SubTB) and SubTB in the set generation task with different problem sizes including (a) small (b) medium (c) large. FL-GFN (SubTB)converges faster and to better solutions, especially for larger sets.



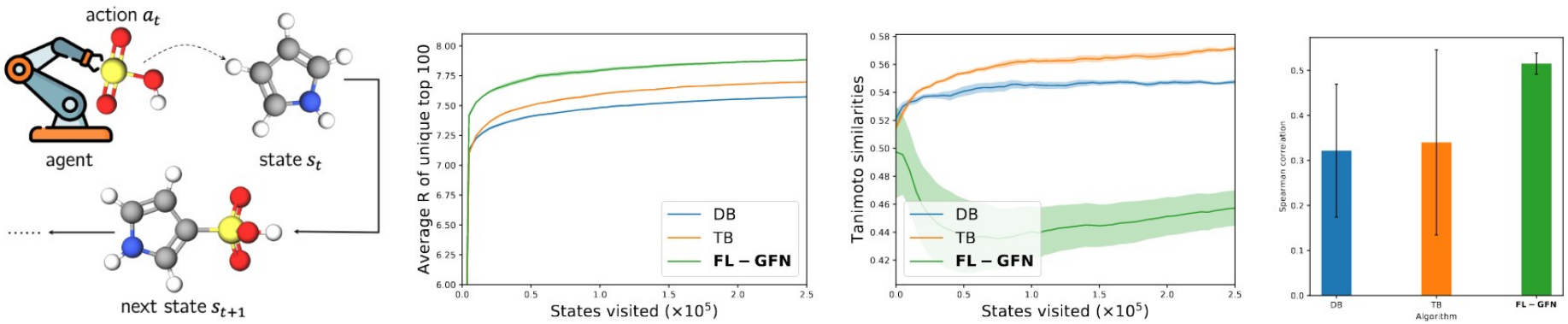
(a) (b) (c)

Figure 4. Performance comparison of baselines trained with incomplete trajectories in the set generation task with different scales including (a) small (b) medium and (c) large. The advantage of FL-GFN increases with the length of trajectories.



(a) (b) (c)

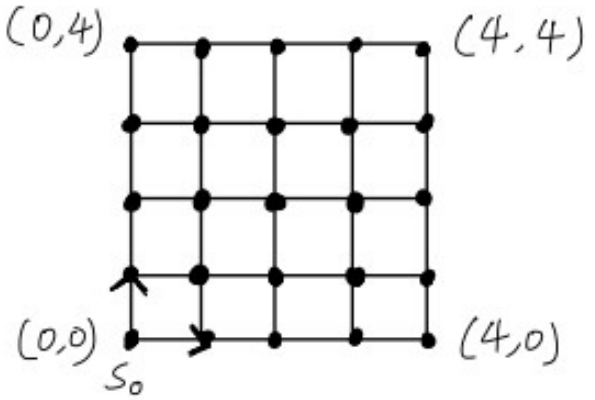
Figure 5. The number of modes discovered by each method in the bit sequence generation task with increasing lengths of the sequences. Different problem sizes are presented from left to right: (a) normal (b) long (c) very long. The advantage of FL-GFN increases with the size of the problem.

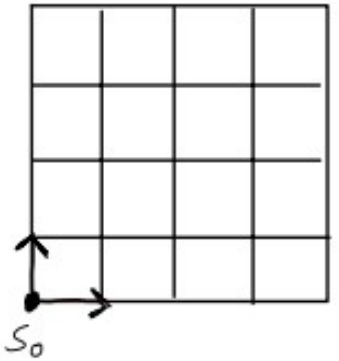


(a) (b) (c) (d)

Figure 6. Results on molecule generation task. (a) Illustration of the GFlowNet generation pipeline. (b) Average reward of the top-100 molecule candidates, showing faster and better training of FL-GFN. (c) The Tanimoto similarity which quantifies diversity (lower is better), showing greater diversity with FL-GFN. (d) The correlation between model log likelihood and the log rewards computed on a held-out set, larger with FL-GFN.

2d grid example

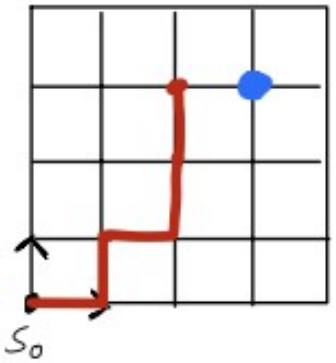


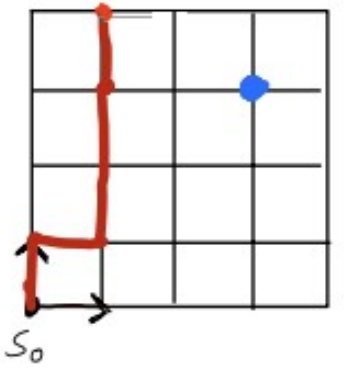


S: State space

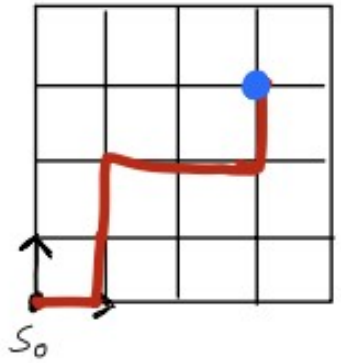
R: Reward

A: Actions (up, right, stop)





Terminate the trajectory when
- chose “stop” action by chance
- hit the boundary

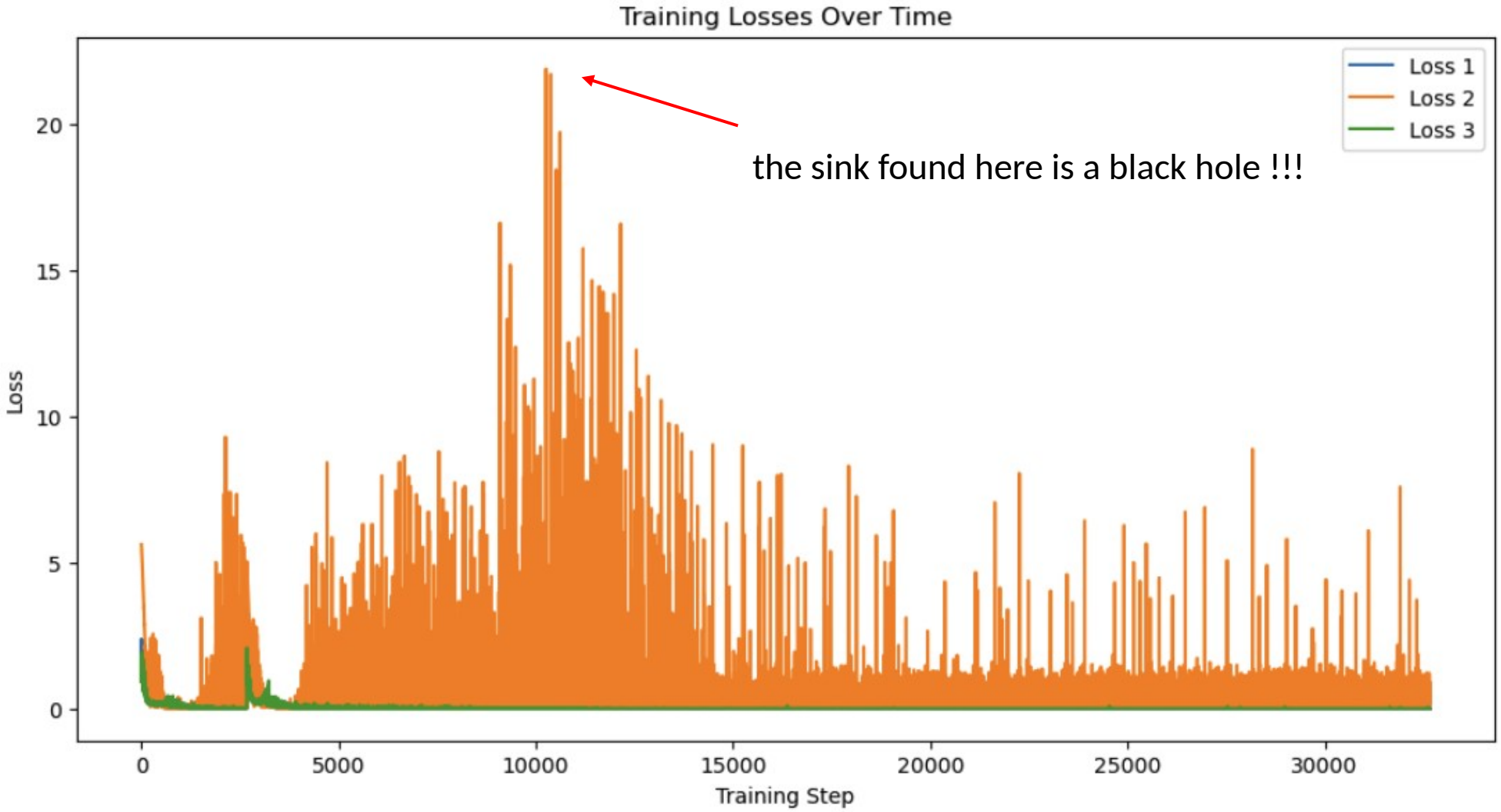



```
done = s.max() >= self.horizon - 1 or a == self.ndim
```

```
Empirical Reward Distribution: {0.1: 0.9951261467889908, 1.1: 0.0047305045871559636, 2.1: 0.00014334862385321102}
  0%|                                                  | 201/90001 [00:51<10:43:05,  2.33it/s]
Empirical Reward Distribution: {0.1: 0.9951026119402985, 1.1: 0.004780783582089552, 2.1: 0.0001166044776119403}
  0%|                                                  | 301/90001 [01:37<10:14:40,  2.43it/s]
Empirical Reward Distribution: {0.1: 0.9932193396226415, 1.1: 0.0066823899371069185, 2.1: 9.827044025157233e-05}
  0%|                                                  | 401/90001 [02:20<11:09:15,  2.23it/s]
Empirical Reward Distribution: {0.1: 0.991593070652174, 1.1: 0.008237092391304348, 2.1: 0.00016983695652173913}
  1%|▍                                                 | 502/90001 [02:51<5:02:04,  4.94it/s]
Empirical Reward Distribution: {0.1: 0.9916267942583732, 1.1: 0.008223684210526315, 2.1: 0.00014952153110047846}
  1%|▍                                                 | 602/90001 [03:05<2:30:18,  9.91it/s]
Empirical Reward Distribution: {0.1: 0.9921207264957265, 1.1: 0.007745726495726496, 2.1: 0.00013354700854700856}
```

```
  4%|██                                                | 3601/90001 [12:29<5:31:29,  4.34it/s]
Empirical Reward Distribution: {0.1: 0.8612487296747967, 1.1: 0.12006161077235772, 2.1: 0.018610264227642278, 4.1: 1.5879065040650408e-05, 8.1: 1.5879065040650408e-05, 3.1: 4.763719512195122e-05}
  4%|██                                                | 3701/90001 [12:52<5:37:35,  4.26it/s]
Empirical Reward Distribution: {0.1: 0.8565721010901883, 1.1: 0.12442703171456888, 2.1: 0.018907953419226957, 4.1: 1.5485629335976214e-05, 8.1: 1.5485629335976214e-05, 3.1: 6.194251734390486e-05}
  4%|██                                                | 3802/90001 [13:14<4:59:08,  4.80it/s]
Empirical Reward Distribution: {0.1: 0.8525598404255319, 1.1: 0.12835469052224371, 2.1: 0.01899480174081238, 4.1: 1.5111218568665377e-05, 8.1: 1.5111218568665377e-05, 3.1: 6.044487427466151e-05}
  4%|██                                                | 3901/90001 [13:37<6:08:20,  3.90it/s]
Empirical Reward Distribution: {0.1: 0.8482648725212465, 1.1: 0.13252478753541078, 2.1: 0.019121813031161474, 4.1: 1.475448536355052e-05, 8.1: 1.475448536355052e-05, 3.1: 5.901794145420208e-05}
  4%|██                                                | 4001/90001 [13:59<5:09:41,  4.63it/s]
```

```
 40%|████████████████████                              | 36301/90001 [7:36:44<6:08:52,  2.43it/s]
Empirical Reward Distribution: {1.1: 0.28209, 0.1: 0.66285, 2.1: 0.04266, 3.1: 0.00655, 4.1: 0.00292, 10.1: 0.00013, 6.1: 0.00055, 12.1: 8e-05, 5.1: 0.00085, 7.1: 0.0005, 14.1: 4e-05, 8.1: 0.00019, 11.1
17, 9.1: 0.00027, 15.1: 3e-05, 16.1: 3e-05, 13.1: 6e-05, 20.1: 1e-05, 19.1: 1e-05, 17.1: 1e-05}
 40%|████████████████████                              | 36401/90001 [7:41:49<6:28:37,  2.30it/s]
Empirical Reward Distribution: {0.1: 0.66401, 1.1: 0.28091, 2.1: 0.04269, 3.1: 0.00657, 6.1: 0.00055, 5.1: 0.00086, 7.1: 0.00051, 10.1: 0.00012, 4.1: 0.00289, 14.1: 4e-05, 8.1: 0.00019, 11.1: 0.00017, 9
0027, 12.1: 7e-05, 15.1: 3e-05, 16.1: 3e-05, 13.1: 6e-05, 20.1: 1e-05, 19.1: 1e-05, 17.1: 1e-05}
 41%|████████████████████                              | 36501/90001 [7:42:55<6:35:19,  2.26it/s]
Empirical Reward Distribution: {0.1: 0.66566, 1.1: 0.2796, 2.1: 0.0424, 4.1: 0.0029, 14.1: 4e-05, 3.1: 0.00655, 7.1: 0.0005, 8.1: 0.0002, 6.1: 0.00054, 5.1: 0.00084, 11.1: 0.00017, 10.1: 0.00011, 9.1:
12.1: 7e-05, 15.1: 3e-05, 16.1: 3e-05, 13.1: 6e-05, 20.1: 1e-05, 19.1: 1e-05, 17.1: 1e-05}
```


Training Losses Over Time
Loss 1
Loss 2
Loss 3
the sink found here is a black hole !!!
Loss
Training Step
0
5
10
15
20
0
5000
10000
15000
20000
25000
30000