Symmetry-Aware GFlowNets

Hohyun Kim [1] **Seunggeun Lee** [1] **Min-hwan Oh** [1]

Abstract

Generative Flow Networks (GFlowNets) offer a powerful framework for sampling graphs in proportion to their rewards. However, existing approaches suffer from systematic biases due to inaccuracies in state transition probability computations. These biases, rooted in the inherent symmetries of graphs, impact both atom-based and fragment-based generation schemes. To address this challenge, we introduce Symmetry-Aware GFlowNets (SA-GFN), a method that incorporates symmetry corrections into the learning process through reward scaling. By integrating bias correction directly into the reward structure, SA-GFN eliminates the need for explicit state transition computations. Empirical results show that SA-GFN enables unbiased sampling while enhancing diversity and consistently generating high-reward graphs that closely match the target distribution.

1. Introduction

GFlowNets have emerged as a powerful framework for learning generative models capable of sampling complex, compositional objects with probabilities proportional to a given reward. Inspired by reinforcement learning (RL), GFlowNets generate these objects through a sequence of actions that iteratively modify the structure of the object being built. This approach is particularly well-suited for generating compositional objects, such as graphs, where there are multiple paths for constructing an object. A prominent application of GFlowNets is molecule generation, where molecules are sequentially constructed as graphs (Bengio et al., 2021; Jain et al., 2023a).

However, GFlowNet training objectives rely on the accurate computation of the transition probability of a policy, which

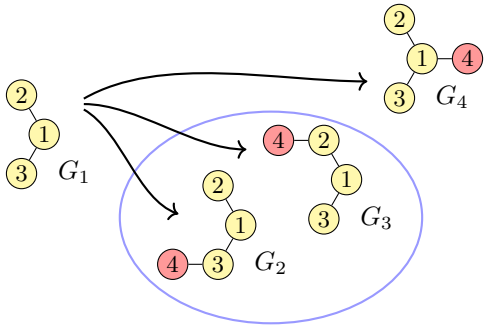


Figure 1: Illustration of graph transitions from G_1 to various successor graphs. The blue oval highlights graphs G_2 and G_3 are isomorphic.

becomes particularly challenging in graph-building environments due to the presence of *equivalent actions*. These are actions that, although different, lead to the same graph structure. For instance, consider Figure 1, where connecting a new node (node 4) to either of two existing nodes (nodes 2 or 3) results in isomorphic graphs. Although these actions are distinct, they lead to structurally identical graphs, meaning their transition probabilities must be summed. More generally, when multiple actions lead to the same state, the transition probability must account for all equivalent actions. This issue, referred to as the *equivalent action problem*, arises because determining whether two actions result in the same state requires computationally expensive graph isomorphism tests (Ma et al., 2024).

While GFlowNets were initially popularized for their reward-matching capabilities, experiments conducted by Ma et al. (2024) demonstrate that neglecting to account for equivalent actions can introduce bias into GFlowNets. Our analysis further shows that the bias is systematic: it skews the model toward sampling graphs with fewer symmetries in node-by-node generation while favoring symmetric components in fragment-based generation. This bias is particularly problematic for tasks such as molecular generation, as molecules inherently possess natural symmetries. For instance, in the ZINC250k dataset, over 50% of molecules exhibit more than one symmetry, with 18% containing four or more symmetries. Ignoring symmetries results in incorrect modeling and inaccurate molecular structure generation, ultimately reducing the accuracy of the sampled molecules.

[1]Graduate School of Data Science, Seoul National University, Seoul, Republic of Korea. Correspondence to: Seunggeun Lee <lee7801@snu.ac.kr>, Min-hwan Oh <minoh@snu.ac.kr>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

In this paper, we propose a simple yet effective modification to the GFlowNet training objectives to resolve the equivalent action problem. Our method adjusts the reward based on the number of symmetries in a graph, requiring only minimal changes to the existing training algorithms. Additionally, we introduce a new unbiased estimator for the model likelihood. Our key contributions are as follows:

- We present a rigorous formulation of autoregressive graph generation within the GFlowNet framework, explicitly addressing the equivalent action problem.
- We propose a simple yet effective method to address the equivalent action problem. Our approach scales the reward based on the automorphism group of the generated graph, enabling GFlowNets to accurately model and sample from the target distribution. Using a similar technique, we also derive an unbiased estimator for the model likelihood.
- Through experiments, we validate our theoretical results, and demonstrate the effectiveness of our method in generating diverse and high-reward samples.

2. Related Work

Autoregressive graph generation. There are two primary formulations of autoregressive models: one based on adjacency matrices and the other based on graph sequences (Chen et al., 2021). Methods based on adjacency matrices (You et al., 2018b; Popova et al., 2019; Liao et al., 2019) are unlikely to suffer from the equivalent action problem because they preserve the node order information generated so far, making each pair of (graph, node order) a unique state. In contrast, equivalent actions arise in methods based on graph sequences (You et al., 2018a; Li et al., 2018; Shi et al., 2020). This becomes problematic if a method requires state transition probabilities, as in GFlowNets. Chen et al. (2021) suggest that, for graph sequence-based methods, the size of a node's orbit is related to the number of equivalent transitions, which inspired our work.

GFlowNets. Several learning objectives have been proposed for GFlowNets, including flow matching (Bengio et al., 2021), detailed balance (Bengio et al., 2023), trajectory balance (Malkin et al., 2022), sub-trajectory balance (Madan et al., 2023), as well as their variants to improve training efficiency (Pan et al., 2023; Shen et al., 2023). Recently, GFlowNets have been found to be equivalent to maximum entropy reinforcement learning (Tiapkin et al., 2024; Mohammadpour et al., 2024), which was previously known to be inadequate for directed acyclic graph (DAG) environments (Bengio et al., 2021). However, none of these objectives can avoid the equivalent action problem, as they are formalized based on state transitions, where multiple isomorphic graphs can represent the next state.

The work most closely related to ours is Ma et al. (2024), which highlighted the importance of accounting for equivalent actions to compute exact transition probabilities. Their approach involved an approximate test using positional encoding (PE) to detect equivalent actions at each transition. However, the bias in GFlowNets was validated only through experiments on synthetic datasets, without any theoretical analysis. Furthermore, their method relies on approximate tests that must be applied at every transition, making it computationally expensive. In contrast, our work provides an exact and efficient solution, requiring corrections only once at the end of trajectories rather than at each transition. This simplification not only reduces computational overhead but also makes our method straightforward to implement. Additionally, we present a comprehensive analysis showing that this bias arises in general settings, affecting both atom- and fragment-based generation schemes, and significantly impacts learning, particularly for highly symmetric graphs. Additional comparisons can be found in Appendix B.

3. Preliminaries

3.1. Graph Theory

Let $G = (V, E)$ denote a graph, where $V = \{v_1, \ldots, v_n\}$ is the set of n vertices, and $E \subseteq V \times V$ is the set of edges. For heterogeneous graphs, we also define labeling functions l_n, l_e, and l_g, which map nodes, edges, and graphs to their respective attributes. We denote $\mathcal{G}$ as the set of all such graphs under consideration. A permutation π is a bijective mapping defined on the vertex set. We extend the permutation to sets as $\pi(V) = \{\pi(v) : v \in V\}$ and $\pi(E) = \{(\pi(v_i), \pi(v_j)) : (v_i, v_j) \in E\}$, as well as to the graph as $\pi(G) = (\pi(V), \pi(E))$. Since any permutation simply relabels node indices, it maps to a structurally identical graph. This notion is formalized as graph isomorphism.

Definition 3.1 (Isomorphism)**.** Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic, denoted $G \cong G'$, if there exists a permutation $\pi : V \to V'$ such that $\pi(E) = E'$. For heterogeneous graphs, the permutation must also preserve labels: for every $v \in V$, $l_n(v) = l'_n(\pi(v))$, for every $(u, v) \in E$, $l_e(u, v) = l'_e(\pi(u), \pi(v))$, and $l_g(G) = l'_g(G')$.

An automorphism is a special case of an isomorphism where the graph is mapped to itself.

Definition 3.2 (Automorphism)**.** An automorphism of a graph $G = (V, E)$ is a permutation π on the vertex set V that preserves the edge set, meaning $\pi(E) = E$. If labels are present, they must also be preserved under the permutation. The set of all automorphisms of a graph G is called the automorphism group of G, denoted by $\mathrm{Aut}(G)$.

In Figure 1, graph G_1 has two automorphisms: the identity mapping and one that permutes nodes 2 and 3. We denote the order (or size) of the automorphism group as $|\mathrm{Aut}(G)|$, which represents the number of symmetries in the graph.

Definition 3.3 (Orbit)**.** The orbit of a node $u \in V$ in graph G is defined as $\mathrm{Orb}(G, u) = \{v \in V : \exists \pi \in \mathrm{Aut}(G), \pi(u) = v\}$. Similarly, the orbit of an edge $(u, v) \in E$ in graph G is defined as $\mathrm{Orb}(G, u, v) = \{(h, k) \in E : \exists \pi \in \mathrm{Aut}(G), (\pi(u), \pi(v)) = (h, k)\}$. More generally, the orbit of a node set $U \subseteq V$ in graph G is defined as $\mathrm{Orb}(G, U) = \{U' : \exists \pi \in \mathrm{Aut}(G), \pi(U) = U'\}$.

An orbit is a set of nodes or edges that are structurally identical. In Figure 1, the orbit of node 2 in graph G_1 is $\{2, 3\}$. Equivalent actions occur because they act on nodes in the same orbit; since nodes 2 and 3 are in the same orbit, adding a new node to either one is equivalent. This point will be further discussed in Section 4.

3.2. Generative Flow Networks

The generation process of GFlowNets is defined by a finite directed acyclic graph (DAG) $(\mathcal{S}, \mathcal{A})$, where $\mathcal{S} = \{s_i\}$ is the set of states, and $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ is the set of state transitions. Let $s_0 \in \mathcal{S}$ denote the special starting point of the process, called the initial state, with no incoming edges in the transition graph. Let $\mathcal{X} \subseteq \mathcal{S}$ be the set of terminal states, for which rewards are given. From the initial state s_0, objects are constructed sequentially by the forward transition policy $p_{\mathcal{A}}(\cdot|s)$ until reaching terminal states. A set of complete trajectories, denoted as $\mathcal{T}$, consists of sequences of transitions $\tau = (s_0, \ldots, s_n)$ starting from the initial state s_0 and terminating at $s_n \in \mathcal{X}$, such that $(s_t, s_{t+1}) \in \mathcal{A}$. Let $\bar{p}_{\mathcal{A}}(s)$ be the probability of reaching s by following the policy $p_{\mathcal{A}}$. The goal of GFlowNets is to train a policy $p_{\mathcal{A}}$ that generates objects with a probability proportional to their reward. Specifically, the policy satisfies $\bar{p}_{\mathcal{A}}(x) = R(x)/Z$ for all $x \in \mathcal{X}$, where Z is a normalizing constant. This is achieved by training $p_{\mathcal{A}}$ using the following objectives.

Trajectory Balance (Malkin et al., 2022). The Trajectory Balance (TB) objective is based on the flow consistency constraint at the trajectory level. Given a complete trajectory τ, the TB objective is defined as follows:

$$\mathcal{L}_{\mathrm{TB}}(\tau) = \left(\log \frac{Z \prod_{t=0}^{n-1} p_{\mathcal{A}}(s_{t+1}|s_t)}{R(s_n) \prod_{t=0}^{n-1} q_{\mathcal{A}}(s_t|s_{t+1})}\right)^2.$$

It introduces a backward policy $q_{\mathcal{A}}$ that reverses the process. Given $q_{\mathcal{A}}$, the forward policy $p_{\mathcal{A}}$ and the normalizing constant Z are trained to match the backward flow induced by the reward function and the backward policy.

Detailed Balance (Bengio et al., 2023). The Detailed Balance (DB) objective is based on the flow consistency constraint at the state transition level. The objective is defined for each transition (s, s') as:

$$\mathcal{L}_{\mathrm{DB}}(s, s') = \left(\log \frac{F(s) p_{\mathcal{A}}(s'|s)}{F(s') q_{\mathcal{A}}(s|s')}\right)^2.$$

The DB objective requires learning the state flow function $F : \mathcal{S} \to \mathbb{R}^+$, which represents the unnormalized probability that the policy visits state s.

4. The Equivalent Action Problem

In this section, we formalize the graph generation process in the context of GFlowNets and investigates its properties.

4.1. Problem Definition

Consider a sequential graph generation process $(\mathcal{G}, \mathcal{E})$ that constructs graphs by modifying the nodes and edges of existing partial graphs, where $\mathcal{E} \subseteq \mathcal{G} \times \mathcal{G}$ represents the set of transitions between graphs. In this section, we clarify the relationship between the two processes, $(\mathcal{S}, \mathcal{A})$ and $(\mathcal{G}, \mathcal{E})$.

Since isomorphism is an equivalence relation, it partitions the space $\mathcal{G}$ into classes, where each graph in a class is structurally identical to the others. Let $[G] = \{G' \in \mathcal{G} : G' \cong G\}$ denote the equivalence class of G induced by graph isomorphism. The state space $\mathcal{S}$ is defined as the set of equivalence classes of graphs, $\mathcal{S} = \{[G] : G \in \mathcal{G}\}$, rather than the graph space $\mathcal{G}$ itself. This is because our goal in using GFlowNets is to sample *any* graph within the equivalence class $s = [G]$ in proportion to $R(s)$. State transitions $\mathcal{A}$ can also be defined by partitioning the graph transitions $\mathcal{E}$ by the following equivalence relation:

Definition 4.1 (Transition equivalence)**.** Graph transitions (G_1, G_1') and (G_2, G_2') are transition-equivalent if $G_1 \cong G_2$ and $G_1' \cong G_2'$.

In practice, a graph generation process $(\mathcal{G}, \mathcal{E})$ is constructed by first designing a set of allowable actions in a given graph. For example, $\texttt{AddEdge}(G, u, v)$ adds an edge (u, v) to the existing graph G, and $\texttt{AddNode}(G, u)$ adds a new node to an existing node u. The $\texttt{Stop}(G)$ terminates the process, in which case the graph-level attribute is flagged as terminated. Through this construction, graph transitions $\mathcal{E}$ are ensured to be structured so that any pair of isomorphic graphs have isomorphic successors.

In this setup, the state transition probability can be expressed in terms of graph transitions as follows: Let $p_{\mathcal{E}}$ denote a forward policy over the graph space, and let $\mathcal{E}(G) = \{G' : (G, G') \in \mathcal{E}\}$ denote the set of next graphs reachable from G. If $p_{\mathcal{E}}$ is parameterized by permutation-equivariant network, then

$$p_{\mathcal{A}}(s'|s) = \sum_{G' \in \mathcal{E}(G) \cap s'} p_{\mathcal{E}}(G'|G), \tag{1}$$

for any $G \in s$, where $\mathcal{E}(G) \cap s'$ is the set of next graphs that are in the same equivalence class s' (see Appendix F.1 for the derivation). This is an exact formula for computing state transition probabilities, which we use as a reference for comparison. However, it requires looking one step ahead and comparing the resulting graphs to identify $\mathcal{E}(G) \cap s'$. This process involves multiple computationally expensive graph isomorphism tests for each transition.

As an alternative, Ma et al. (2024) suggested using orbits to identify transition-equivalent actions, which we formalize next. Define *graph actions* as triples, $\bar{\mathcal{E}} = \{(G, t, u)\}$, where t denotes the action type and u specifies the nodes or edges affected by the action. Since graph transitions $\mathcal{E}$ are constructed by predefined set of graph actions, there is a one-to-one correspondence between $\mathcal{E}$ and $\bar{\mathcal{E}}$. However, we explicitly distinguish between them to introduce another equivalence relation.

Definition 4.2 (Orbit equivalence)**.** Graph actions (G_1, t_1, u_1) and (G_2, t_2, u_2) are orbit-equivalent if $t_1 = t_2$ and there exists a permutation π such that $\pi(G_1) = G_2$ and $\pi(u_1) = u_2$. When $G_1 = G_2$, orbit equivalence indicates that u_1 and u_2 belong to the same orbit.

In Figure 1, the transitions (G_1, G_2) and (G_1, G_3) are transition-equivalent because the resulting graphs are isomorphic, $G_2 \cong G_3$. These transitions are induced by the actions $\texttt{AddNode}(G_1, 2)$ and $\texttt{AddNode}(G_1, 3)$, which are orbit-equivalent since nodes 2 and 3 belong to the same orbit in graph G_1. Similarly to $\mathcal{A}$, which is induced by transition equivalence relation, we define $\bar{\mathcal{A}}$ as the set of equivalence classes induced by the orbit equivalence relation on $\bar{\mathcal{E}}$. The notion of orbit equivalence is particularly useful because it serves as a replacement for transition equivalence. The next theorem establishes that orbit-equivalent actions induce equivalent transitions.

Theorem 4.3. *Let (G_1, t_1, u_1, G_1') and (G_2, t_2, u_2, G_2') be two graph transitions induced by actions $e_1 = (G_1, t_1, u_1)$ and $e_2 = (G_2, t_2, u_2)$. If e_1 and e_2 are orbit-equivalent, then (G_1, G_1') and (G_2, G_2') are transition-equivalent.*

In other words, graph actions operating on the same orbit lead to isomorphic graphs. This is because orbits, rather than individual nodes or edges, are structurally important in determining actions. The theorem implies that, in a given graph G, transition-equivalent actions—actions that lead to isomorphic graphs—can be identified by computing orbits. However, transition-equivalent actions are not always orbit-equivalent (see Appendix D), meaning that the orbit equivalence relation provides a finer partition of $\mathcal{E}$ than the transition equivalence.

Given the distinction between transition-equivalent actions $\mathcal{A}$ and orbit-equivalent actions $\bar{\mathcal{A}}$, we define the state-action probability $p_{\bar{\mathcal{A}}}(a|s)$ differently from the state transition probability $p_{\mathcal{A}}(s'|s)$: while $p_{\mathcal{A}}(s'|s)$ accounts for all transition-equivalent actions, $p_{\bar{\mathcal{A}}}(a|s)$ aggregates only over orbit-equivalent actions. Formally, for a given graph G, let $\bar{\mathcal{E}}(G)$ denote the set of graph actions available from G. Then, for a state-action pair $s \in \mathcal{S}$ and $a \in \bar{\mathcal{A}}$, the state-action probability is defined as:

$$p_{\bar{\mathcal{A}}}(a|s) = \sum_{e \in \bar{\mathcal{E}}(G) \cap a} p_{\bar{\mathcal{E}}}(e|G) \tag{2}$$

where $p_{\bar{\mathcal{E}}}(e|G)$ denotes $p_{\mathcal{E}}(G'|G)$ for G' being the next graph. Here, $\bar{\mathcal{E}}(G) \cap a$ represents the set of orbit-equivalent actions from G. By Theorem 4.3, it follows that $p_{\bar{\mathcal{A}}}(a|s) \leq p_{\mathcal{A}}(s'|s)$ in general.

Computing exact state transition probabilities includes expensive graph isomorphism tests as stated. Instead, Ma et al. (2024) proposed using Equation (2), observing that $p_{\bar{\mathcal{A}}}(a|s) = p_{\mathcal{A}}(s'|s)$ in most cases. However, computing orbits for every transition remains computationally intensive, which led them to develop approximate solutions. In the next section, we present an exact and efficient solution to address this challenge.

4.2. Properties of Equivalent Actions

In fact, accounting for orbit equivalence is sufficient for GFlowNets, as demonstrated by the following theorem.

Theorem 4.4 (Sufficiency of orbit equivalence)**.** *Let $q_{\bar{\mathcal{A}}}$ denote the backward state-action policy. State-action flow constraints are defined as $F(s)p_{\bar{\mathcal{A}}}(a|s) = F(s')q_{\bar{\mathcal{A}}}(a|s')$. If state-action flow constraints are satisfied for all possible state-action pairs, then the state transition flow constraints, $F(s)p_{\mathcal{A}}(s'|s) = F(s')q_{\mathcal{A}}(s|s')$, are also satisfied.*

GFlowNets are typically formulated such that each pair of states are connected at most once in a DAG $(\mathcal{S}, \mathcal{A})$, ensuring that every action leads to a unique next state. In contrast, in $(\mathcal{S}, \bar{\mathcal{A}})$, distinct actions can lead to the same next state. This can be interpreted as multiple pathways connecting the same pair of states, enabling parallel flows. Theorem 4.4 essentially states that edge flows can be subdivided into multiple flows within a transition.

Our next goal is to simplify the computation of state-action probabilities. First note that $p_{\bar{\mathcal{A}}}(a|s)$ can simply be expressed as:

$$p_{\bar{\mathcal{A}}}(a|s) = |\bar{\mathcal{E}}(G) \cap a| \cdot p_{\mathcal{E}}(G'|G)$$

where $|\bar{\mathcal{E}}(G) \cap a|$ represents the number of orbit-equivalent actions. This is because when actions are parameterized using graph neural networks (GNNs), orbit-equivalent actions are assigned equal probabilities. In general, permutation-equivariant functions produce identical representations for nodes within the same orbit (as detailed in Appendix E.1).

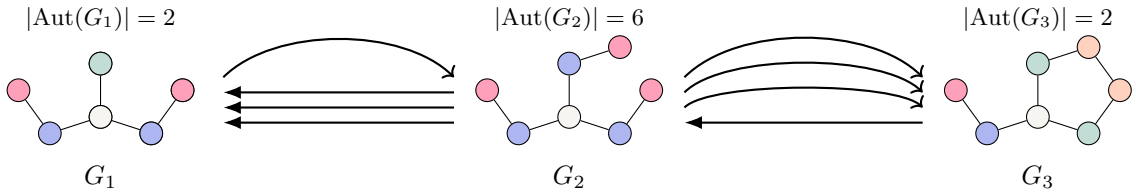


Figure 2: Graphs representing transitions (G_1, G_2, G_3), where the first transition is performed by `AddNode` and the second by `AddEdge`. The number of forward/backward actions are represented as the number of arrows. Symmetries in each graph is related to orbit-equivalent actions, as seen in the ratio $|\mathrm{Aut}(G_1)|/|\mathrm{Aut}(G_2)| = |\mathrm{Orb}(G_1, \circ)|/|\mathrm{Orb}(G_2, \circ)|$. Nodes in the same orbit are given the same color.

When node representations are aggregated to compute edge representations using invariant aggregators such as `SUM` or `MEAN`, edges within the same orbit also receive identical representations. Alternative parameterizations, such as the relative edge parameterization proposed by Shen et al. (2023), also assign equal probabilities to orbit-equivalent actions, while enhancing representational power.

Note that the number of orbit-equivalent actions, $|\bar{\mathcal{E}}(G) \cap a|$, corresponds to the size of the orbit of the associated nodes or edges. The following lemma shows that, when considering ratios, counting orbit-equivalent actions simplifies to counting automorphisms.

Lemma 4.5. *Let $G' = G[E \cup (u, v)]$ be a graph induced by adding an edge (u, v) to graph G. Then, the following relationship holds:*

$$\frac{|\mathrm{Orb}(G, u, v)|}{|\mathrm{Orb}(G', u, v)|} = \frac{|\mathrm{Aut}(G)|}{|\mathrm{Aut}(G')|}$$

We presented Lemma 4.5 in the context of `AddEdge` for simplicity, but similar lemma holds for other action types such as `AddNode`, `AddNodeAttributes`, and others used in node-by-node generation (See Appendix C for the full list of considered actions). We also consider `AddFragment` in our experiments, but discuss its properties separately in Appendix H, as the corresponding formula differs in fragment-based schemes.

In Figure 2, we observe that the number of equivalent actions changes as the graph evolves. For instance, from G_1, there is only one forward equivalent action, while from G_2, there are three. The number of backward actions also varies with each transition, making it seem daunting to account for all equivalent actions step-by-step. However, the ratio of forward and backward orbit-equivalent actions can be simply expressed as the ratio of the sizes of their automorphism groups. This is the basis for the next theorem.

Theorem 4.6 (Automorphism correction)**.** *Let $q_\mathcal{E}$ denote a graph-level policy defined for the backward process. Let (G, G') be the graph transition, and (s, a, s') denote a corresponding state transition. If permutation-equivariant functions are used for $p_\mathcal{E}$ and $q_\mathcal{E}$, then the following holds:*

$$\frac{p_{\bar{\mathcal{A}}}(a|s)}{q_{\bar{\mathcal{A}}}(a|s')} = \frac{|\mathrm{Aut}(G)|}{|\mathrm{Aut}(G')|} \cdot \frac{p_\mathcal{E}(G'|G)}{q_\mathcal{E}(G|G')}.$$

The theorem suggests a simple adjustment method when considering the ratio. This simplification leads to the straightforward reward-scaling method presented in the next section.

5. Symmetry-Aware GFlowNets

In this section, we analyze GFlowNet objectives using our previous results. The following theorem shows that a naive implementation of the TB objective, which does not account for equivalent actions, will train a model biased toward graphs with fewer symmetries.

Corollary 5.1 (TB correction)**.** *Assume that G_0 is the empty graph or a single node, so that $|\mathrm{Aut}(G_0)| = 1$. Given the complete graph trajectory $\tau = (G_0, G_1, \ldots, G_n)$, constructed using a node-by-node generation scheme, the trajectory balance loss is given by:*

$$\mathcal{L}_{\mathrm{TB}}(\tau) = \left(\log \frac{Z \prod_{t=0}^{n-1} p_\mathcal{E}(G_{t+1}|G_t)}{|\mathrm{Aut}(G_n)| R(G_n) \prod_{t=0}^{n-1} q_\mathcal{E}(G_t|G_{t+1})} \right)^2.$$

The equation follows from Theorem 4.6 and the application of a telescoping sum.

Implication. Corollary 5.1 shows that we need to multiply the reward by the order of the automorphism group of the terminal state to properly account for equivalent actions. If we do not scale the reward, we are effectively reducing the rewards for highly symmetric graphs by a factor of $1/|\mathrm{Aut}(G_n)|$. As a result, even if a model is fully trained, the likelihood of reaching the terminal state will not align with the desired distribution; instead, the model is penalized for generating symmetric graphs, following $\bar{p}_\mathcal{A}([G_n]) \propto R(G_n)/|\mathrm{Aut}(G_n)|$. This bias can be easily corrected by evaluating $|\mathrm{Aut}(G_n)|$ and scaling the reward accordingly.

We can also adjust the DB objective by multiplying the symmetry ratio $|\mathrm{Aut}(G')/|\mathrm{Aut}(G)|$ to the backward probability for each transition, though this requires multiple evaluations of automorphisms per trajectory. The next theorem states that, as in the TB correction, we can simply scale the rewards by $|\mathrm{Aut}(G)|$ without needing to count automorphisms at each transition.

Theorem 5.2 (DB correction)**.** *Consider a node-by-node graph generation scheme. We define the graph-level detailed balance condition, as opposed to the standard state-level condition, as follows:*

$$\tilde{F}(G)p_{\mathcal{E}}(G'|G) = \tilde{F}(G')q_{\mathcal{E}}(G|G'),$$

where $\tilde{F}$ denotes the graph-level flow function. If rewards are given by $\tilde{R}(G) = |\mathrm{Aut}(G)|R(G)$ and the graph-level detailed balance condition is satisfied for all transitions, then the forward policy samples terminal states proportionally to the given reward R.

Implication. Together with Corollary 5.1, we see that scaling the reward alone is sufficient for both TB and DB objectives. This suggests that other GFlowNet objectives, such as subtrajectory balance (Madan et al., 2023) and flow-matching (Bengio et al., 2021), can also be used with reward scaling (see the discussion on the flow-matching Appendix G). This provides a straightforward approach to implementing GFlowNet objectives while reducing the computational burden of counting automorphisms at each transition.

Finally, we provide the adjustment formula for fragment-based generation and defer the detailed discussion to Appendix H.

Theorem 5.3 (Fragment correction)**.** *Let G represents a terminal state ($[G] \in \mathcal{X}$) generated by connecting k fragments $\{C_1, \ldots, C_k\}$. Then, the scaled rewards to offset the effects of equivalent actions are given by:*

$$\tilde{R}(G) = \frac{|\mathrm{Aut}(G)|R(G)}{\prod_{i=1}^{k} |\mathrm{Aut}(C_i)|} \tag{3}$$

Intuitively, highly symmetric fragments contain many symmetric nodes available for connection, resulting in multiple forward equivalent actions, even though these actions do not lead to distinct outcomes. As a result, without correction, symmetric fragments are more likely to be sampled by the model. Equation (3) corrects this bias by penalizing symmetric fragments.

Estimating model likelihood. To address the intractability of marginalizing over all trajectories terminating at $x \in \mathcal{X}$, Zhang et al. (2022) proposed approximating the model likelihood using importance sampling with $q_{\mathcal{E}}$ as a variational distribution: $\bar{p}_{\mathcal{A}}(x) = \mathbb{E}_{\tau\sim q_{\mathcal{E}}(\tau|G_n)} \frac{p_{\mathcal{E}}(\tau)}{q_{\mathcal{E}}(\tau|G_n)}$, where $\tau = (G_0, \ldots, G_n)$. However, Zhang et al. (2022) worked with a restricted class of decision process where the equivalent action problem is not present. Instead, we estimate the probability of the terminal state as follows:

$$\begin{aligned} \bar{p}_{\mathcal{A}}(x) &= \mathbb{E}_{\tau\sim q_{\mathcal{E}}(\tau|G_n)} \left[\frac{p_{\mathcal{E}}(\tau)}{|\mathrm{Aut}(G_n)| q_{\mathcal{E}}(\tau|G_n)} \right] \\ &\approx \frac{1}{M|\mathrm{Aut}(G_n)|} \sum_{i=1}^{M} \frac{p_{\mathcal{E}}(\tau_i)}{q_{\mathcal{E}}(\tau_i|G_n)}. \end{aligned} \tag{4}$$

If we do not account for equivalent actions during both training and model likelihood estimation, the estimated model likelihood may still correlate with the rewards, but the actual sampling distribution will be biased. This happens because the policy is already biased towards generating samples with low $|\mathrm{Aut}(G_n)|$, leading to a spurious correlation.

Impact of GNN expressive power. Another source of inexact learning comes from the limited expressive power of GNNs used to parameterize the policy (Silva et al., 2025). The correction formula relies on the parameter-sharing property of GNNs for nodes within the same orbit. While this property is desirable, actions from different orbits may also collapse into identical representations, thereby reducing the network's representational power. As a result, the policy might assign equal probabilities to actions that lead to different rewards. Although this issue is not the primary focus of this paper, we provide additional analysis of its impact in Appendix E.2.

Computation. The main additional computation for reward scaling comes from evaluating $|\mathrm{Aut}(G)|$, which is necessary for each trajectory in both the TB and DB objectives. For fragment correction, we can pre-compute $|\mathrm{Aut}(C)|$ in our vocabulary set. While the fastest proven time complexity for computing $|\mathrm{Aut}(G)|$ has remained $\exp(\mathcal{O}(\sqrt{n \log n}))$ for decades (Babai et al., 1983), graphs with bounded degrees can be handled in polynomial time (Luks, 1982). In our experiments, we used the *bliss* algorithm (Junttila & Kaski, 2007), included in the `igraph` package (Csardi & Nepusz, 2006), and did not observe any significant delays in computation. In contrast, computing transition equivalent actions and summing their probabilities at each step involves several graph isomorphism tests. This process requires $K \times H$ more computations compared to the reward scaling, where K is the average number of actions per state, and H is the average trajectory length. We provide further analysis and comparisons on the computation time for each method in Appendix J.

Relation to distribution learning While our work focuses on improving the reward-matching capabilities of GFlowNets, many graph generation methods instead aim to learn the data distribution by maximizing a variational lower bound (VLB):

$$\log \bar{p}_{\mathcal{A}}(x) \geq \mathbb{E}_{\tau\sim q(\tau|x)} \left[\log p_{\mathcal{A}}(\tau) - \log q(\tau|x) \right]. \tag{5}$$

Here, $q(\tau|x)$ is a variational distribution that samples trajectories $\tau = (s_0, \ldots, s_n = x)$ conditioned on the final state x. There are two primary ways to parameterize q and sample trajectories τ: (1) q can be implemented as a backward policy $q(s_t|s_{t+1})$, as in GFlowNets; (2) $q(\pi|x)$ can first sample a node or edge ordering π, which then deterministically defines a trajectory τ. Our method suggest that when the backward policy is used to define $q(s_t|s_{t+1})$, the VLB can be computed via automorphism counting. In contrast, when $q(\pi|x)$ is used to sample orderings, no further correction is required. This is because forward equivalent actions split probabilities across different orderings that induce the same trajectory. Since this result differs from that of Chen et al. (2021), we elaborate on this point in Appendix I, where we interpret equivalent actions as those that correspond to different orderings but ultimately lead to the same state sequence.

6. Experiments

In this section, we conduct experiments to validate our theoretical results and demonstrate the effectiveness of our method. We use a uniform backward policy across all experiments. Details on hyperparameters and model configurations can be found in Appendix K. The experiments compare the following methods: (1) **Vanilla** GFlowNets, which do not incorporate graph symmetries. (2) **Transition Correction**, which identifies transition-equivalent actions by performing multiple isomorphism tests and sums their probabilities accordingly. (3) **PE**, the method proposed by Ma et al. (2024), which approximately identifies orbit-equivalent actions using positional encoding. (4) **Reward Scaling**, which achieves correction by modifying only the reward signals. (5) **Flow Scaling**, which multiplies symmetry ratio $|\mathrm{Aut}(G')|/|\mathrm{Aut}(G)|$ to backward probability at each transition. Note that methods 3-5 are orbit correction methods. Since Reward Scaling and Flow Scaling have the same effect under the TB objective, we only consider Flow Scaling when using the DB objective.[1]

6.1. Illustrative Example

We first conducted an illustrative experiment where the initial state consisted of six disconnected nodes, and only `AddEdge` and `Stop` actions were allowed. Terminal states correspond to connected graphs, with a uniform reward of 1 assigned to each. Theorem 4.6 predicts that the terminating probability of the vanilla GFlowNet will exhibit a bias proportional to $|\mathrm{Aut}(G_0)|/|\mathrm{Aut}(G_n)|$, where $|\mathrm{Aut}(G_0)| = 6!$. This corresponds to the number of graphs isomorphic to the terminal state x, meaning $\bar{p}_{\mathcal{A}}(x) \propto |x|$. Our method corrects this bias by multiplying the rewards by $1/|x|$.

We trained three policies using the TB objective and computed the exact terminating probabilities for all states ($|\mathcal{X}| = 112$). As shown in Figure 3 (a), the terminating probabilities of the vanilla model are clustered according to $|x|$. In contrast, when state transition probabilities $p_{\mathcal{A}}(s'|s)$ are computed exactly by summing over transition-equivalent actions, the terminating probabilities are uniform as desired. Notably, Reward Scaling achieves the same effect, validating Theorem 4.4. The PE method exhibits approximate bias correction, as indicated by the scatter in its results.

Upon inspecting the trained normalizing constant, we observed that with Reward Scaling, the estimated Z is 112, matching the true value. Without correction, however, Z is trained to be significantly larger, reaching 26706. This discrepancy arises because, in this example, the correction works by scaling down the rewards by $1/|x|$. Alternatively, since the 6! term in $|x|$ is constant, it can be absorbed into the normalizing constant Z. Therefore, the rewards could instead be scaled up by $|\mathrm{Aut}(x)|$ to achieve the same effect.

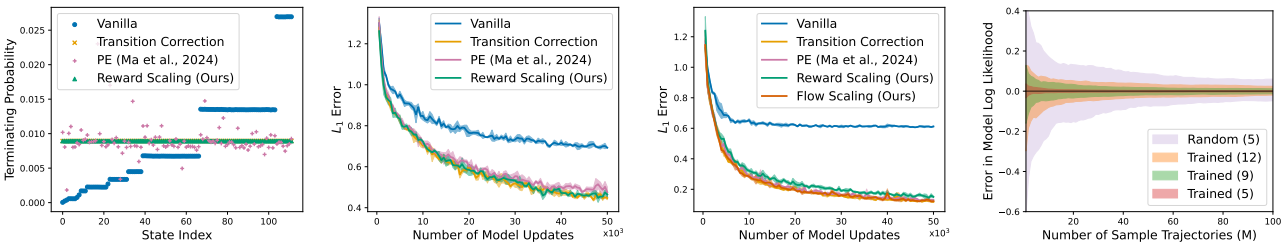
6.2. Synthetic Graphs

Following Ma et al. (2024), we set up a graph-building environment where nodes can be one of two types, and graphs can contain up to 7 nodes ($|\mathcal{X}| = 72296$). Rewards are assigned based on the number of 4-cliques that contain at least three nodes of the same type. For this relatively small environment, we compute exact terminating probabilities for all states without approximations for evaluation.

The results for the TB objective are presented in Figure 3 (b). The vanilla GFlowNet exhibits limited performance, as measured by L_1 errors between the target probabilities and the model's terminating probabilities. In contrast, our method (Reward Scaling) has substantially lower errors, producing results similar to those obtained by explicitly computing state transition probabilities at each step (Transition Correction). Although the PE method is an approximate solution and underperforms compared to ours, it still significantly outperforms the vanilla baseline, underscoring the importance of applying a correction.

For the DB objective shown in Figure 3 (c), we observe that Reward Scaling requires more training steps to converge than other correction methods. This is because the Reward Scaling corrects only at the end of trajectories, leaving intermediate probabilities remain inaccurate. This hinders training under the DB objective, which relies on intermediate probabilities. On the other hand, the per-transition correction can be interpreted as providing intermediate signals for the adjustment, similar to the idea of providing intermediate reward signals, as suggested by Pan et al. (2023). Reward Scaling achieves the same goal, but defers the adjustment

[1] Source code available at: `https://github.com/hohyun312/sagfn`



(a) Illustrative (b) Synthetic (TB) (c) Synthetic (DB) (d) Likelihood estimation

Figure 3: **(a)** Terminating probabilities of trained models in the uniform-reward environment. States are sorted according to the number of graphs in the state, $|x|$. **(b), (c)** L_1 errors between the target probabilities and the model's terminating probabilities during training in the synthetic environment. **(d)** Errors in the estimated model log-likelihood, defined as the difference between estimated and exact log-likelihood. "Random" denotes the errors of an initial random model, while "Trained" refers that of a trained model. Numbers in brackets indicate the number of edges in the terminal states used for estimation.

signal to the end of the trajectory.

To evaluate the effectiveness of the proposed model likelihood estimator, we sampled 100 terminal states for each category (5, 9, and 12 edges), resulting in a total of 300 states, and estimated their model likelihood using Equation (4). The Figure 3 (d) displays the estimation errors (computed as estimates minus exact values) for each category, with shaded bands representing one standard deviation. The estimates converge to the exact likelihood as M increases, with notably rapid convergence for small values of M.

However, the estimation error varies significantly depending on the task. For instance, terminal states with 5 edges can be estimated more accurately than those with 12 edges, as 12-edge states have substantially more trajectories leading to them in this environment, making the estimation problem more challenging. Additionally, a trained model's likelihood can be estimated more accurately than that of a random policy; in fact, for a fully trained model, a single sampled trajectory would be sufficient for an accurate estimation.

6.3. Molecule Generation

Task description. We investigate whether accurately modeling a given target distribution helps generate diverse and high-reward samples in practice. We examine the atom-based generation task from Jain et al. (2023b) and the fragment-based generation task from Bengio et al. (2021). In the atom-based task, the goal is to generate molecules by sequentially adding new atoms, edges, or setting their attributes. Rewards are provided by a proxy model, which predicts the HOMO-LUMO gap. In the fragment-based task, we use a predefined set of fragments, each with a predefined set of attachment points—nodes on the fragment where edges can connect. The task involves building a tree graph, where each node represents a fragment, and edges specify the attachment points on the two connected fragments. Rewards are determined by a proxy model that predicts the binding energy of a molecule to the sEH target.

For the atom-based task, we simply scale the final rewards by the order of the automorphism group. For the fragment-based task, we additionally correct for fragment automorphisms as described in Equation (3). We also explore an approximate correction scheme that offers computational benefits, as detailed in Appendix K.4. We sampled 5,000 molecules from each method and evaluated them using common metrics. The definitions of these metrics are provided in Appendix K.2.

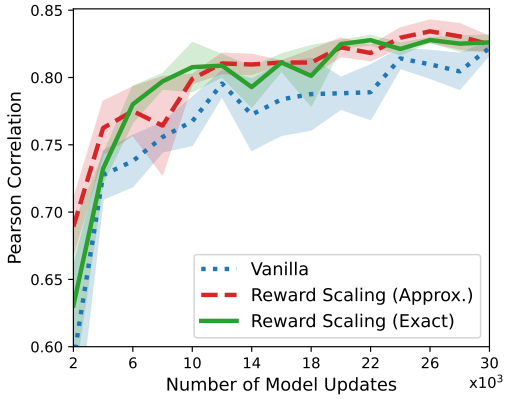


Figure 4: Correlations between log rewards and model log-likelihoods during training in fragment experiment.

Results. The results summarized in Table 1 show that accurately modeling the target distribution yields generally the better results in terms of generating diverse and high-reward samples for both molecule tasks. This result is noteworthy, as the vanilla model effectively optimizes for two objectives—the proxy and non-symmetries—which could enhance diversity. However, atom-based task exhibited limited improvement, possibly due to the reward structure of the task, where rewards are negatively correlated with the

Table 1: Results for molecule generation task. Highest scores are highlighted.

Task	Method	Diversity	Top K div.	Top K reward	Div. Top K	Uniq. Frac.
Atom	Vanilla	$0.929_{\pm 0.024}$	$0.077_{\pm 0.022}$	$1.09_{\pm 0.02}$	$1.09_{\pm 0.02}$	$0.93_{\pm 0.077}$
	Reward Scaling (Exact)	$0.959_{\pm 0.01}$	$0.046_{\pm 0.006}$	$1.091_{\pm 0.013}$	$1.091_{\pm 0.013}$	$1.0_{\pm 0.0}$
Fragment	Vanilla	$0.877_{\pm 0.001}$	$0.153_{\pm 0.003}$	$0.941_{\pm 0.002}$	$0.941_{\pm 0.002}$	$1.0_{\pm 0.0}$
	Reward Scaling (Approx.)	$0.88_{\pm 0.001}$	$0.164_{\pm 0.008}$	$0.949_{\pm 0.006}$	$0.949_{\pm 0.006}$	$1.0_{\pm 0.0}$
	Reward Scaling (Exact)	$0.879_{\pm 0.0}$	$0.151_{\pm 0.002}$	$0.952_{\pm 0.003}$	$0.952_{\pm 0.003}$	$1.0_{\pm 0.0}$

number of symmetries when measured in the QM9 test set.

For the fragment-based task, the sampled molecules show higher rewards with our method. We also observe that the approximate correction already enables the generation of high-reward samples, underscoring the effectiveness and importance of the correction. Without correction, the trained model tends to excessively favor components that incur multiple forward equivalent actions during generation. For example, among 5000 sampled molecules, the vanilla GFlowNet produced 5220 instances of cyclohexane (`C1CCCCC1`) as its fragments, whereas the corrected method produced only 1042.

In addition, we measured the Pearson correlation between the estimated model log-likelihood, $\log \bar{p}_{\mathcal{A}}(x)$, and the log rewards, $\log R(x)$, on the test set to validate the proposed fragment correction method. Figure 4 shows an overall high correlation for both exact and approximate corrections, emphasizing the impact of our methods.

7. Discussion and Conclusion

GFlowNets were first proposed as an alternative to previous methods, such as MaxEnt RL (Haarnoja et al., 2017), which are biased toward states with multiple action sequences leading to them. However, incorrect modeling of state transition probabilities introduces another type of bias in graph generation. Although we believe that the previous experimental results remain valid if interpreted carefully with the problem in mind, we recommend being explicit about the correction method used in all future work.

In this paper, we analyzed the properties of equivalent actions and proposed a simple correction method that allows for unbiased sampling from the target distribution. Our analysis shows that, without correction, highly symmetric graphs are less likely to be sampled, while symmetric fragments are more likely to be sampled, which is crucial for molecule discovery. We demonstrated that the reward-scaling technique works for both TB and DB objectives. Experimental results suggest that reward scaling and flow scaling effectively removes bias, allowing for accurate modeling of the target distribution, which is essential for sampling high-reward molecules. The exact effect, however, depends on the reward structure of the given task.

While our method is general and applicable to both node-by-node and fragment-based generation schemes, our theoretical guarantees rely on a specific set of predefined graph actions. Therefore, when designing a new set of graph actions, it is important to ensure that they share a similar structure, so that the theorems remain applicable. In most cases, however, the graph actions we introduced can be readily extended to incorporate additional actions. See Appendix C for further discussion.

A potential limitation of this paper is that the proposed correction method is demonstrated primarily on specific objectives (TB and DB) and datasets relevant to molecule discovery. Future work could explore applying the method to tasks with different symmetry patterns and reward structures.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2022-NR071853 and RS-2023-00222663), the Global-LAMP Program of the NRF grant funded by the Ministry of Education (No. RS-2023-00301976), Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2025-02263754), Brain Pool Plus (BP+, Brain Pool+) Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (2020H1D3A2A03100666), Korea Government Grant Program for Education and Research in Medical AI through the Korea Health Industry Development Institude (KHIDI) funded by the Korea government (MOE, MOHW), and AI-Bio Research Grant through Seoul National University.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Babai, L., Kantor, W. M., and Luks, E. M. Computational complexity and the classification of finite simple groups. In *24th Annual Symposium on Foundations of Computer Science (Sfcs 1983)*, pp. 162–171. IEEE, 1983.

Bengio, E., Jain, M., Korablyov, M., Precup, D., and Bengio, Y. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021.

Bengio, Y., Lahlou, S., Deleu, T., Hu, E. J., Tiwari, M., and Bengio, E. Gflownet foundations. *Journal of Machine Learning Research*, 24(210):1–55, 2023.

Chen, X., Han, X., Hu, J., Ruiz, F. J., and Liu, L. Order matters: Probabilistic modeling of node sequence for graph generation. *arXiv preprint arXiv:2106.06189*, 2021.

Csardi, G. and Nepusz, T. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL `https://igraph.org`.

Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021.

Flam-Shepherd, D., Zhu, K., and Aspuru-Guzik, A. Language models can learn complex molecular distributions. *Nature Communications*, 13(1):3293, 2022.

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

Jain, M., Deleu, T., Hartford, J., Liu, C.-H., Hernandez-Garcia, A., and Bengio, Y. Gflownets for ai-driven scientific discovery. *Digital Discovery*, 2(3):557–577, 2023a.

Jain, M., Raparthy, S. C., Hernández-Garcıa, A., Rector-Brooks, J., Bengio, Y., Miret, S., and Bengio, E. Multi-objective gflownets. In *International conference on machine learning*, pp. 14631–14653. PMLR, 2023b.

Junttila, T. and Kaski, P. Engineering an efficient canonical labeling tool for large and sparse graphs. In *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pp. 135–149. SIAM, 2007.

Kingma, D. P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kong, L., Cui, J., Sun, H., Zhuang, Y., Prakash, B. A., and Zhang, C. Autoregressive diffusion model for graph generation. In *International conference on machine learning*, pp. 17391–17408. PMLR, 2023.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W., Duvenaud, D. K., Urtasun, R., and Zemel, R. Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems*, 32, 2019.

Luks, E. M. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of computer and system sciences*, 25(1):42–65, 1982.

Ma, G., Bengio, E., Bengio, Y., and Zhang, D. Baking symmetry into gflownets. *arXiv preprint arXiv:2406.05426*, 2024.

Madan, K., Rector-Brooks, J., Korablyov, M., Bengio, E., Jain, M., Nica, A. C., Bosc, T., Bengio, Y., and Malkin, N. Learning gflownets from partial episodes for improved convergence and stability. In *International Conference on Machine Learning*, pp. 23467–23483. PMLR, 2023.

Malkin, N., Jain, M., Bengio, E., Sun, C., and Bengio, Y. Trajectory balance: Improved credit assignment in gflownets. *Advances in Neural Information Processing Systems*, 35:5955–5967, 2022.

McKay, B. D. and Piperno, A. Nauty and traces user's guide (version 2.5). *Computer Science Department, Australian National University, Canberra, Australia*, 2013.

Mohammadpour, S., Bengio, E., Frejinger, E., and Bacon, P.-L. Maximum entropy gflownets with soft q-learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 2593–2601. PMLR, 2024.

Pan, L., Malkin, N., Zhang, D., and Bengio, Y. Better training of gflownets with local credit and incomplete trajectories. In *International Conference on Machine Learning*, pp. 26878–26890. PMLR, 2023.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Popova, M., Shvets, M., Oliva, J., and Isayev, O. Molecularrnn: Generating realistic molecular graphs with optimized properties. *arXiv preprint arXiv:1905.13372*, 2019.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

Shen, M. W., Bengio, E., Hajiramezanali, E., Loukas, A., Cho, K., and Biancalani, T. Towards understanding and improving gflownet training. In *International Conference on Machine Learning*, pp. 30956–30975. PMLR, 2023.

Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., and Tang, J. Graphaf: a flow-based autoregressive model for molecular graph generation. *arXiv preprint arXiv:2001.09382*, 2020.

Silva, T., Alves, R. B., da Silva, E. d. S., Souza, A. H., Garg, V., Kaski, S., and Mesquita, D. When do gflownets learn the right distribution? In *The Thirteenth International Conference on Learning Representations*, 2025.

Tiapkin, D., Morozov, N., Naumov, A., and Vetrov, D. P. Generative flow networks as entropy-regularized rl. In *International Conference on Artificial Intelligence and Statistics*, pp. 4213–4221. PMLR, 2024.

Wang, Z., Shi, J., Heess, N., Gretton, A., and Titsias, M. K. Learning-order autoregressive models with application to molecular graph generation. *arXiv preprint arXiv:2503.05979*, 2025.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

You, J., Liu, B., Ying, Z., Pande, V., and Leskovec, J. Graph convolutional policy network for goal-directed molecular graph generation. *Advances in neural information processing systems*, 31, 2018a.

You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pp. 5708–5717. PMLR, 2018b.

Yun, S., Jeong, M., Kim, R., Kang, J., and Kim, H. J. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

Zhang, D., Malkin, N., Liu, Z., Volokhova, A., Courville, A., and Bengio, Y. Generative flow networks for discrete probabilistic modeling. In *International Conference on Machine Learning*, pp. 26412–26428. PMLR, 2022.

Zhang, M., Li, P., Xia, Y., Wang, K., and Jin, L. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34:9061–9073, 2021.

A. Notations

Table 2: Notation

Graph	Set of graphs	$\mathcal{G}$
	Set of graph transitions	$\mathcal{E}$
	Set of graph actions	$\bar{\mathcal{E}}$
	Set of vertices	V
	Set of edges	E
	Node labeling function	l_n
	Edge labeling function	l_e
	Graph labeling function	l_g
	Permutation	π
	Set of automorphisms of graph G	$\mathrm{Aut}(G)$
	Set of graphs isomorphic to graph G	$[G]$
	Orbit of a node u	$\mathrm{Orb}(G, u)$
	Stabilizer of a node u	$\mathrm{Stab}(G, u)$
	Forward policy over graphs	$p_{\mathcal{E}}$
	Backward policy over graphs	$q_{\mathcal{E}}$
	Set of next graphs from graph G	$\mathcal{E}(G)$
	Set of actions from graph G	$\bar{\mathcal{E}}(G)$
	Graph-action probability	$p_{\bar{\mathcal{E}}}$
	Graph-level flow function	$\tilde{F}$
State	Set of states	$\mathcal{S}$
	Set of state transitions	$\mathcal{A}$
	Set of actions	$\bar{\mathcal{A}}$
	Set of terminal states	$\mathcal{X}$
	Set of complete trajectories	$\mathcal{T}$
	Reward function	R
	Forward policy over states	$p_{\mathcal{A}}$
	Backward policy over states	$q_{\mathcal{A}}$
	Terminating probability induced by following $p_{\mathcal{A}}$	$\bar{p}_{\mathcal{A}}$
	State-action probability	$p_{\bar{\mathcal{A}}}$
	State-level flow function	F

B. Additional Comparison to Prior Work

To the best of our knowledge, Ma et al. (2024) is the only prior work addressing the equivalent action problem in GFlowNets. Their approach relies on approximate tests using positional encoding (PE) of nodes to identify nodes or edges within the same orbit. Once an orbit is identified, the probabilities of orbit-equivalent actions are summed. While they identified and partially addressed this issue, their discussion was limited to experimental validation. The primary motivation of Ma et al. (2024) was to highlight the existence of the problem and propose a partial solution. To this end, they conducted experiments in an offline, atom-based environment.

In contrast, our work provides the first rigorous theoretical foundation for the correction, demonstrating that this issue is not merely an experimental artifact but a fundamental and systematic challenge arising from graph symmetries in both atom-based and fragment-based generation. This insight is particularly significant given that GFlowNets were initially popularized for their reward-matching capabilities.

Our approach, based on reward/flow scaling, offers an exact and efficient solution. Unlike PE-based methods, which require adaptation for different action types (e.g., incorporating edge types and fragments), our method is straightforward to implement and easily generalizable across various action types. Our motivation is to thoroughly analyze the problem and present an efficient, scalable solution applicable to real-world setups. To validate this, we conducted experiments with online training for both atom- and fragment-based generation.

C. Definitions of Graph Actions

Here we provide a list of action types considered in the paper.

- `AddNode`(G, u) adds a new node to the existing node u.
- `AddEdge`(G, u, v) adds a new edge (u, v).
- `AddFragment`(G, C) adds a fragment C.
- `RemoveNode`(G, v) removes node v and its connecting edges.
- `RemoveEdge`(G, u, v) removes the edge (u, v).
- `RemoveFragment`(G, C) removes the subgraph C.
- `SetNodeAttribute`(G, u, t) sets the node-level attribute t for node u.
- `SetEdgeAttribute`(G, u, v, t) sets the edge-level attribute t for the edge (u, v).
- `SetGraphAttribute`(G, t) sets a graph-level attribute t.

The `Stop` action can be interpreted as setting a terminal flag, making `SetGraphAttribute` a viable replacement. Some actions may overlap in functionality. For instance, `AddNode` is equivalent to a sequence of two actions: adding a new node and connecting it to an existing node u. This can be achieved using a combination of `AddFragment` and `AddEdge`.

Likewise, the above graph actions can be easily extended to incorporate additional actions. For example, we can define `AddColoredNode`(G, u, t) as an action that adds a new node with node type t to the existing node u. In fact, we used `AddColoredNode`, instead of using `AddNode` and `SetNodeAttribute` for the Synthetic Graphs experiment.

From a practical perspective, defining `AddColoredNode` as a separate action type reduces the number of transitions per trajectory and improves convenience. For theorems, however, proving `AddColoredNode` as a distinct case may be redundant.

D. Example of Equivalent Actions

In Theorem 4.3, we established that orbit equivalence implies transition equivalence. However, the converse is not generally true, though such cases are rare. A counterexample is illustrated in Figure 5. While the two resulting graphs are isomorphic (by the permutation 1→4, 2→5, 3→1, 4→3, 5→6, 6→2), they are induced by actions that modify nodes belonging to different orbits. In other words, two actions `AddEdge`$(2, 3)$ and `AddEdge`$(4, 6)$ are transition-equivalent, but not orbit-equivalent. However, accounting for orbit-equivalent actions is sufficient, as the corresponding backward actions belong to two distinct orbits as well.

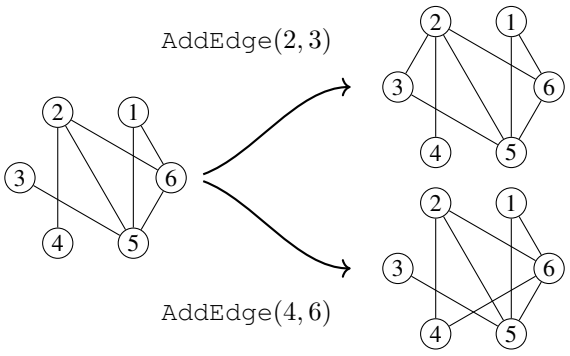


Figure 5: Two actions induce isomorphic graphs, making them transition-equivalent. However, they are not orbit-equivalent. This example was originally presented by (Ma et al., 2024).

E. Properties of Graph Neural Networks

E.1. Permutation Equivariance

The key design principle of GNNs is permutation equivariance, which ensures that the output remains consistent regardless of how the nodes in the input graph are ordered.

Definition E.1 (Permutation Equivariance)**.** A function f is permutation-equivariant if it satisfies $f(\pi(x)) = \pi(f(x))$ for any permutation π.

Specifically, let $\mathbf{A} \in \mathbb{R}^{n\times n\times d}$ be the adjacency tensor of a graph G with n nodes. The d dimensional node and edge features of G are represented in $\mathbf{A}$, where diagonal elements encode the node features. Let $\mathbf{A}[i,j]$ represent the (i,j)-th element of the tensor. We define the permutation of the tensor as $\pi(\mathbf{A})[i,j] = \mathbf{A}[\pi^{-1}(i), \pi^{-1}(j)]$. Automorphisms are permutations that preserve adjacency tensor. That is, for $\pi \in \mathrm{Aut}(G)$, we have $\pi(\mathbf{A}) = \mathbf{A}$. Since GNNs are permutation-equivariant, we can show that they produce identical node representations for nodes in the same orbit.

Theorem E.2. *Let $f : \mathbb{R}^{n\times n\times d} \times \mathbb{R}^{n\times n\times d}$ be a permutation-equivariant function. Then, for any $u, v, h, k \in V$, if there exists a permutation $\pi \in \mathrm{Aut}(G)$ such that $\pi(u) = h$ and $\pi(v) = k$, it follows that $f(\mathbf{A})[u,v] = f(\mathbf{A})[h,k]$.*

Proof.

$$\begin{aligned} f(\mathbf{A})[u,v] &= \pi^{-1}(f(\pi(\mathbf{A})))[u,v] \\ &= f(\pi(\mathbf{A}))[\pi(u), \pi(v)] \\ &= f(\mathbf{A})[h,k]. \end{aligned}$$

□

The theorem implies that nodes and edges within the same orbit are represented identically by GNNs.

E.2. Expressive Power

Another source of bias comes from the expressiveness of GNNs used to parameterize the policy. If actions from different orbits collapse into identical representations, reducing the network's representational power, the policy may be forced to assign equal probabilities to actions that produce different rewards. In Figure 6, the two actions $\texttt{AddEdge}(G, 2, 4)$ and $\texttt{AddEdge}(G, 2, 5)$ will have identical representations if they are aggregated from node representations. This occurs because nodes $2, 4$ and 5 all belong to the same orbit and, therefore, share identical representations.

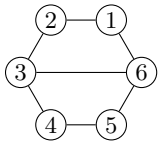


Figure 6: Two edges $(2,4)$ and $(2,5)$ belong to different orbits and will result in non-isomorphic graphs if added to the graph. However, their edge representations will be identical if aggregated from node representations.

Significant research has been conducted on techniques to enhance the expressive power of GNNs (Xu et al., 2018; Dwivedi et al., 2021). While much of this work has focused on improving graph-level representations, some methods have been proposed to enhance multi-node representations, such as edges (Zhang et al., 2021). In graph generation tasks, actions are often parameterized using all levels of representations—node-level, edge-level, and graph-level. This complexity makes designing expressive GNNs more challenging, emphasizing the need for architectures with enhanced expressive power when the task requires it.

In our synthetic experiments in the main text, we augmented edge-level representations from the GNN with shortest path lengths to distinguish edges in different orbits. For example, in Figure 6, shortest path length of the edge $(2,4)$ is 2, while the length of $(2,5)$ is 3. We conducted experiments without this feature augmentation, resulting in an inexact policy, as shown in Figure 11. Compare this to Figure 3 (a), where the terminating probabilities are more accurate. In Figure 11, terminating probabilities for certain states exhibit significantly larger errors, with those states marked by red circles. These states correspond to successor states derived from the state shown in Figure 6.

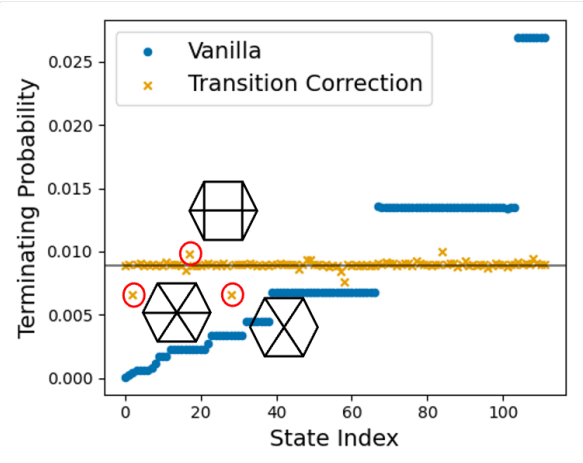


Figure 7: Terminating probabilities. States are sorted according to the number of graphs in the state, $|x|$. States with large errors are marked with red circles.

F. Proofs

F.1. Proof of Equation 6

While we did not present Equation (6) as a theorem, a formal derivation offers insights into the relationship between $(\mathcal{S}, \mathcal{A})$

and $(\mathcal{G}, \mathcal{E})$. Specifically, Equation (6) states that

$$p_{\mathcal{A}}(s'|s) = \sum_{G' \in \mathcal{E}(G) \cap s'} p_{\mathcal{E}}(G'|G). \tag{6}$$

for any $G \in s$.

Proof. We expand state transition probability $p_{\mathcal{A}}(s'|s)$ in terms of graph transitions as follows:

$$\begin{aligned} p_{\mathcal{A}}(s'|s) &= \frac{p_{\mathcal{A}}(s'|s)\bar{p}_{\mathcal{A}}(s)}{\bar{p}_{\mathcal{A}}(s)} \\ &= \frac{\sum_{G \in s} \sum_{G' \in \mathcal{E}(G) \cap s'} p_{\mathcal{E}}(G'|G)\bar{p}_{\mathcal{E}}(G)}{\sum_{G \in s} \bar{p}_{\mathcal{E}}(G)} \\ &= \frac{\sum_{G \in s} \bar{p}_{\mathcal{E}}(G) \sum_{G' \in \mathcal{E}(G) \cap s'} p_{\mathcal{E}}(G'|G)}{\sum_{G \in s} \bar{p}_{\mathcal{E}}(G)} \end{aligned}$$

If $\sum_{G' \in \mathcal{E}(G) \cap s'} p_{\mathcal{E}}(G'|G)$ is constant for all $G \in s$, we can factor it out, obtaining:

$$p_{\mathcal{A}}(s'|s) = \sum_{G' \in \mathcal{E}(G_0) \cap s'} p_{\mathcal{E}}(G'|G_0), \quad \text{for any } G_0 \in s.$$

This constantness follows from two assumptions: 1) $p_{\mathcal{E}}$ is permutation equivalent, and 2) $\mathcal{E}$ is structured, meaning for any isomorphic graphs $G, G' \in s$, the set of next graphs are matched by some permutation such that $\mathcal{E}(G) = \pi(\mathcal{E}(G'))$. In other words,

$$\sum_{G' \in \mathcal{E}(G) \cap s'} p_{\mathcal{E}}(G'|G) = \sum_{G' \in \mathcal{E}(\pi(G)) \cap s'} p_{\mathcal{E}}(G'|\pi(G))$$

holds for all $\pi \in \mathrm{Aut}(G)$. □

F.2. Proof of Theorem 4.3

The theorem simplifies to the assertion that, for a given graph G, graph actions of the same type applied within the same orbit are transition-equivalent. To establish this, we prove the theorem for each action type. The results are straightforward for attribute-level actions, and we provide a proof for the `SetNodeAttribute` action.

Lemma F.1 (SetNodeAttribute)**.** *Let $G[l_n(u) = t]$ denote the graph where the attribute of node u in graph G is changed to t. If $\mathrm{Orb}(G, u) = \mathrm{Orb}(G, v)$, then $G[l_n(u) = t] \cong G[l_n(v) = t]$.*

Proof. Let us denote the node labeling function of G, $G[l_n(u) = t]$ and $G[l_n(v) = t]$ as l_n, $l_{n[u=t]}$ and $l_{n[v=t]}$ respectively. Since u and v are in the same orbit, there exists $\pi \in \mathrm{Aut}(G)$ such that $\pi(u) = v$. Showing $l_{n[u=t]}(w) = l_{n[v=t]}(\pi(w))$ for all $w \in V$ is sufficient to establish the isomorphism. We prove for two cases. First, let $w = u$. Then, π satisfies $l_{n[u=t]}(w) = l_{n[u=t]}(u) = l_{n[v=t]}(v) = l_{n[v=t]}(\pi(u)) = l_{n[v=t]}(\pi(w))$. Secondly, let $w \neq u$. Then, $l_{n[u=t]}(w) = l_n(w) = l_n(\pi(w)) = l_{n[v=t]}(\pi(w))$. □

The proof for the `SetEdgeAttribute` action is nearly identical to that for `SetNodeAttribute`, with nodes replaced by edges. We now proceed to prove the cases for the `AddEdge`, `AddNode`, and their corresponding backward actions. We use the following two properties in our proofs.

$$\begin{aligned} \pi(E \cup E') &= \{(\pi(u), \pi(v)) : (u, v) \in E \cup E'\} \\ &= \{(\pi(u), \pi(v)) : (u, v) \in E \text{ or } (u, v) \in E'\} \\ &= \{(\pi(u), \pi(v)) : (u, v) \in E\} \cup \{(\pi(u), \pi(v)) : (u, v) \in E'\} \\ &= \pi(E) \cup \pi(E'), \end{aligned}$$

and similarly,

$$\begin{aligned}\pi(E \setminus E') &= \{(\pi(u), \pi(v)) : (u, v) \in E \setminus E'\} \\ &= \{(\pi(u), \pi(v)) : (u, v) \in E \text{ and } (u, v) \notin E'\} \\ &= \{(\pi(u), \pi(v)) : (u, v) \in E\} \setminus \{(\pi(u), \pi(v)) : (u, v) \in E'\} \\ &= \pi(E) \setminus \pi(E'),\end{aligned}$$

where E and E' are edge sets. We assume homogeneous graphs for simplicity.

Lemma F.2 (AddEdge)**.** *Let $G[E \cup (u, v)]$ and $G[E \cup (h, k)]$ denote the graphs induced by $E \cup \{(u, v)\}$ and $E \cup \{(h, k)\}$, respectively. If (u, v) and (h, k) are in the same orbit in G, then $G[E \cup (u, v)]$ and $G[E \cup (h, k)]$ are isomorphic. In other words, $\mathrm{Orb}(G, u, v) = \mathrm{Orb}(G, h, k)$ implies $G[E \cup (u, v)] \cong G[E \cup (h, k)]$.*

Proof. If (u, v) and (h, k) are in the same orbit, then there exists $\pi \in \mathrm{Aut}(G)$ such that $(\pi(u), \pi(v)) = (h, k)$. Since π is an automorphism, it also satisfies $\pi(E) = E$. Thus, $\pi(E \cup \{(u, v)\}) = \pi(E) \cup \{(\pi(u), \pi(v))\} = E \cup \{(h, k)\}$, indicating that π is an isomorphism between $G[E \cup (u, v)]$ and $G[E \cup (h, k)]$. □

Lemma F.3 (RemoveEdge)**.** *Let $G[E \setminus (u, v)]$ and $G[E \setminus (h, k)]$ denote graphs induced by $E \setminus \{(u, v)\}$ and $E \setminus \{(h, k)\}$ respectively. Then, $G[E \setminus (u, v)]$ and $G[E \setminus (h, k)]$ are isomorphic if (u, v) and (h, k) are in the same orbit in graph G.*

Proof. Let $\pi \in \mathrm{Orb}(G, u, v)$ such that $(\pi(u), \pi(v)) = (h, k)$. Then, $\pi(E \setminus \{(u, v)\}) = \pi(E) \setminus \{(\pi(u), \pi(v))\}) = E \setminus \{(h, k)\}$, completing the proof. □

Lemma F.4 (AddNode)**.** *Let $G[E \cup (u, w)]$ and $G[E \cup (v, w)]$ denote the graphs induced by attaching a new node w to the existing nodes u and v, respectively. Then, $\mathrm{Orb}(G, u) = \mathrm{Orb}(G, v)$ implies $G[E \cup (u, w)] \cong G[E \cup (v, w)]$.*

Proof. Let $\pi \in \mathrm{Orb}(G, u)$ such that $\pi(u) = v$, and let $\tilde{\pi} : V \cup \{w\} \to V \cup \{w\}$ be the extension of π such that $\tilde{\pi}(w) = w$ and $\tilde{\pi}(i) = \pi(i)$ for $i \neq w$. Then, $\tilde{\pi}(E \cup (u, w)) = \tilde{\pi}(E) \cup (\tilde{\pi}(u), \tilde{\pi}(w)) = \pi(E) \cup (\pi(u), \tilde{\pi}(w)) = E \cup (v, w)$. □

Corollary F.5 (RemoveNode)**.** *Let $G[V \setminus u]$ and $G[V \setminus v]$ denote the graphs induced by removing nodes u and v, respectively, where edges connected to u or v are also removed. Then, $\mathrm{Orb}(G, u) = \mathrm{Orb}(G, v)$ implies $G[V \setminus u] \cong G[V \setminus v]$.*

Proof. Let $\pi \in \mathrm{Aut}(G)$ be an automorphism of $G = (V, E)$ such that $\pi(u) = v$. Then:

$$\begin{aligned}\pi\big(E \setminus (\{(u, k) : k \in V\} \cup \{(k, u) : k \in V\})\big) &= \pi(E) \setminus (\{(\pi(u), \pi(k)) : k \in V\} \cup \{(\pi(k), \pi(u)) : k \in V\}) \\ &= E \setminus (\{(v, k') : k' \in V\} \cup \{(k', v) : k' \in V\})\end{aligned}$$

where $E \setminus (\{(u, k) : k \in V\} \cup \{(k, u) : k \in V\})$ and $E \setminus (\{(v, k') : k' \in V\} \cup \{(k', v) : k' \in V\})$ are edge sets of the induced subgraphs $G[V \setminus u]$ and $G[V \setminus v]$, respectively. Therefore, π, restricted to $V \setminus \{u\}$, is an isomorphism between $G[V \setminus u]$ and $G[V \setminus v]$. □

F.3. Proof of Theorem 4.4

Proof. Note that orbit equivalence implies transition equivalence by Theorem 4.3, meaning that each set of transition equivalent actions can be partitioned into subsets of orbit-equivalent actions. State transition probabilities can thus be computed by summing over state-action probabillities $p_{\bar{\mathcal{A}}}(a|s)$, where each $p_{\bar{\mathcal{A}}}$ is in turn defined by summing over orbit-equivalent graph actions in a.

Formally, let $\bar{\mathcal{A}}(s)$ be the set of actions available from state s, and let $a(s)$ represent the next state obtained by applying action a to s. Define $\bar{\mathcal{A}}(s, s') = \{a \in \bar{\mathcal{A}}(s) : a(s) = s'\}$ as the set of forward actions from state s that lead to state s'. Then, state transition probability can be expressed as:

$$p_{\mathcal{A}}(s'|s) = \sum_{a \in \bar{\mathcal{A}}(s,s')} p_{\bar{\mathcal{A}}}(a|s)$$
$$q_{\mathcal{A}}(s|s') = \sum_{a \in \bar{\mathcal{A}}(s,s')} q_{\bar{\mathcal{A}}}(a|s').$$

From the assumption, state-action flow constraints are satisfied, i.e.,

$$F(s)p_{\bar{\mathcal{A}}}(a|s) = F(s')q_{\bar{\mathcal{A}}}(a|s').$$

The assumption can be satisfied if the equivariant neural networks parameterizing $p_{\bar{\mathcal{A}}}$ and $q_{\bar{\mathcal{A}}}$ can distinguish between orbits differently, and the number of orbits (and hence the number of actions) is the same for both the forward and backward transitions.

Summing both side of the equations over $\bar{\mathcal{A}}(s, s')$, we have state transition flow constraints, $F(s)p_{\mathcal{A}}(s'|s) = F(s')q_{\mathcal{A}}(s|s')$.

□

F.4. Proof of Lemma 4.5 and Its Generalization

Lemma 4.5 relates the order of orbits to the order of automorphism group. The proof relies on orbit-stabilizer theorem, hence we introduce the following definition.

Definition F.6 (Stabilizer)**.** The stabilizer of a node $u \in V$ in graph G is the set of automorphisms that fix node u: $\mathrm{Stab}(G, u) = \{\pi \in \mathrm{Aut}(G) : \pi(u) = u\}$. The stabilizer of an edge (u, v) is defined as $\mathrm{Stab}(G, u, v) = \{\pi \in \mathrm{Aut}(G) : \pi(u) = u, \pi(v) = v\}$. Similarly, the stabilizer of a node set S is defined as $\mathrm{Stab}(G, S) = \{\pi \in \mathrm{Aut}(G) : \pi(S) = S\}$.

We restate Lemma 4.5 and provide its proof.

Lemma F.7 (AddEdge)**.** *Let $G = G'[E' \setminus (u, v)]$ and $G' = G[E \cup (u, v)]$ be two successive graphs induced be $E' \setminus (u, v)$ and $E \cup (u, v)$. Then the following equation holds:*

$$\frac{|\mathrm{Orb}(G, u, v)|}{|\mathrm{Orb}(G', u, v)|} = \frac{|\mathrm{Aut}(G)|}{|\mathrm{Aut}(G')|}.$$

Proof. Using the orbit-stabilizer theorem, we have

$$\begin{aligned}\frac{|\mathrm{Orb}(G, u, v)|}{|\mathrm{Orb}(G', u, v)|} = \frac{|\mathrm{Aut}(G)|}{|\mathrm{Aut}(G')|} &\iff \frac{|\mathrm{Aut}(G)|}{|\mathrm{Orb}(G, u, v)|} = \frac{|\mathrm{Aut}(G')|}{|\mathrm{Orb}(G', u, v)|} \\ &\iff |\mathrm{Stab}(G, u, v)| = |\mathrm{Stab}(G', u, v)|.\end{aligned}$$

Hence, we prove the lemma by showing $|\mathrm{Stab}(G, u, v)| = |\mathrm{Stab}(G', u, v)|$. It suffices to prove that $\mathrm{Stab}(G, u, v) = \mathrm{Stab}(G', u, v)$.

First, we show that $\mathrm{Stab}(G, u, v) \subseteq \mathrm{Stab}(G', u, v)$. Let $\pi \in \mathrm{Stab}(G, u, v)$. Then $\pi(u) = u$, $\pi(v) = v$, and $\pi(E \cup \{(u, v)\}) = \pi(E) \cup \{(\pi(u), \pi(v))\} = E \cup \{(u, v)\}$, which implies that $\pi \in \mathrm{Stab}(G', u, v)$.

Conversely, let $\pi' \in \mathrm{Stab}(G', u, v)$, so $\pi'(u) = u$, $\pi'(v) = v$, and $\pi'(E \cup \{(u, v)\}) = \pi'(E) \cup \{(u, v)\} = E \cup (u, v)$. Suppose for contradiction that $\pi'(E) \neq E$. Then some edge in E must be mapped to a different edge not in E, and to satisfy the equality $\pi'(E) \cup \{(u, v)\} = E \cup (u, v)$, the only possibility is that $\pi'(E) \setminus E = \{(u, v)\}$. But since $\pi'(\{(u, v)\}) = \{(u, v)\}$, this implies a duplication, contradicting the assumption that π' is a permutation. Therefore, $\pi'(E) = E$, and hence $\pi' \in \mathrm{Stab}(G, u, v)$.

Thus, $\mathrm{Stab}(G, u, v) = \mathrm{Stab}(G', u, v)$, completing the proof. □

For general statement of Lemma 4.5, we define the orbit of a graph action as the orbit of set of nodes or edges affected by the action. For example, the orbit of $e = \texttt{AddEdge}(G, u, v)$, denoted as $\mathrm{Orb}(G, e)$, corresponds to $\mathrm{Orb}(G, u, v)$. The backward action associated with e is $\texttt{RemoveEdge}(G', u, v)$, and the orbit of this action when applied to the next graph G' is denoted as $\mathrm{Orb}(G', e)$, which corresponds to $\mathrm{Orb}(G', u, v)$. Definition of orbits for each action type is provided in Table 3.

Table 3: Orbit of graph actions

$\texttt{AddNode}(G, u)$	$\mathrm{Orb}(G, u)$
$\texttt{RemoveNode}(G, v)$	$\mathrm{Orb}(G, v)$
$\texttt{AddEdge}(G, u, v)$	$\mathrm{Orb}(G, u, v)$
$\texttt{RemoveEdge}(G, u, v)$	$\mathrm{Orb}(G, u, v)$
$\texttt{SetNodeAttribute}(G, u, t)$	$\mathrm{Orb}(G, u)$
$\texttt{SetEdgeAttribute}(G, u, v, t)$	$\mathrm{Orb}(G, u, v)$
$\texttt{SetGraphAttribute}(G, t)$	$\mathrm{Orb}(G, V)$

Using these definitions, we generalize Lemma 4.5 as follows.

Lemma F.8. *Let $(G, G') \in \mathcal{E}$ be a graph transition induced by an action $e \in \bar{\mathcal{E}}$ that either adds a node, an edge, or modifies an attribute in the graph G. Then, the following relationship holds:*

$$\frac{\textit{Number of forward orbit-equivalent actions}}{\textit{Number of backward orbit-equivalent actions}} = \frac{|\mathrm{Orb}(G, e)|}{|\mathrm{Orb}(G', e)|} = \frac{|\mathrm{Aut}(G)|}{|\mathrm{Aut}(G')|}.$$

We already proved Lemma F.8 for $\texttt{AddEdge}$.

For $\texttt{AddNodeAttribute}(G, u, t)$, the proof is straightforward: the only difference between G and G' is the attribute assigned to node u, so it is immediate that $\mathrm{Stab}(G, u) = \mathrm{Stab}(G', u)$. The argument for $\texttt{AddEdgeAttribute}$ is nearly identical. Next, we prove the corresponding result for the $\texttt{AddNode}$.

Lemma F.9 (AddNode)**.** *Let $G' = (V', E')$ be the graph resulted by adding node v to node u in graph $G = (V, E)$. Then,*

$$\frac{|\mathrm{Orb}(G, u)|}{|\mathrm{Orb}(G', v)|} = \frac{|\mathrm{Aut}(G)|}{|\mathrm{Aut}(G')|}.$$

Proof. Using the orbit-stabilizer theorem, it suffices to show $|\mathrm{Stab}(G, u)| = |\mathrm{Stab}(G', v)|$. We do this by constructing a bijective map $f : \mathrm{Stab}(G, u) \to \mathrm{Stab}(G', v)$. Define $f(\pi) = \tilde{\pi}$ for $\pi \in \mathrm{Stab}(G, u)$, where $\tilde{\pi}$ is the extension of π defined on $V \cup \{v\}$ given by

$$\tilde{\pi}(w) = \begin{cases} \pi(w) & \text{if } w \in V \\ v & \text{if } w = v. \end{cases}$$

We claim that $\tilde{\pi} \in \mathrm{Stab}(G', v)$. Since $\pi \in \mathrm{Aut}(G)$ and $\pi(u) = u$, it follows that $\tilde{\pi} \in \mathrm{Aut}(G')$ and $\tilde{\pi}(v) = v$. Thus, f maps into $\mathrm{Stab}(G', v)$ and is injective by construction.

To show that f is surjective, let $\pi' \in \mathrm{Stab}(G', v)$. Then $\pi' \in \mathrm{Aut}(G')$ and $\pi'(v) = v$, and we have

$$\pi'(E \cup \{(u, v)\}) = \pi'(E) \cup \{(\pi'(u), v)\} = E \cup \{(u, v)\}.$$

This implies that $\pi'(E) = E$ and $(\pi'(u), v) = (u, v)$, so $\pi'(u) = u$, since $\pi'(E)$ cannot contain (u, v). Therefore, the restriction of π' on V, denoted π, belongs to $\mathrm{Stab}(G, u)$, and satisfies $f(\pi) = \pi'$.

Hence, f is bijective, and $|\mathrm{Stab}(G, u)| = |\mathrm{Stab}(G', v)|$, as claimed. □

F.5. Proof of Theorem 4.6

Proof. We first note that state-action probability can be computed by multiplying the number of orbit-equivalent actions when $p_{\mathcal{E}}$ is permutation-equivariant:

$$\begin{aligned} p_{\bar{\mathcal{A}}}(a|s) &= \sum_{e \in \bar{\mathcal{E}}(G) \cap a} p_{\bar{\mathcal{E}}}(e|G) \\ &= |\bar{\mathcal{E}}(G) \cap a| \cdot p_{\bar{\mathcal{E}}}(e|G) \end{aligned}$$

for any $G \in s$. Since $|\bar{\mathcal{E}}(G) \cap a|$ represents the number of orbit-equivalent actions from graph G, it is equal to $|\mathrm{Orb}(G, e)|$ for any $e \in \bar{\mathcal{E}}(G) \cap a$, where we defined the orbit of graph actions in Appendix F.4. Thus,

$$p_{\bar{\mathcal{A}}}(a|s) = |\mathrm{Orb}(G, e)| \cdot p_{\bar{\mathcal{E}}}(e|G).$$

Similarly, for the backward policy, we have

$$q_{\bar{\mathcal{A}}}(a|s') = |\mathrm{Orb}(G', e)| \cdot q_{\bar{\mathcal{E}}}(e|G').$$

We prove Theorem 4.6 using the following sequence of equations.

$$\begin{aligned} \frac{p_{\bar{\mathcal{A}}}(a|s)}{q_{\bar{\mathcal{A}}}(a|s')} &= \frac{|\mathrm{Orb}(G, e)| \cdot p_{\bar{\mathcal{E}}}(e|G)}{|\mathrm{Orb}(G', e)| \cdot q_{\bar{\mathcal{E}}}(e|G')} \\ &= \frac{|\mathrm{Aut}(G)|}{|\mathrm{Aut}(G')|} \cdot \frac{p_{\mathcal{E}}(e|G)}{q_{\mathcal{E}}(e|G')}, \end{aligned}$$

where we used Lemma F.8 for the last equation. □

F.6. Proof of Theorem 5.2

Before proving Theorem 5.2, we first prove the existence of a policy that satisfies graph-level DB constraints.

Lemma F.10. *For any given reward function R, there exist $p_{\mathcal{E}}$, $q_{\mathcal{E}}$, and $\tilde{F}$ that satisfy the graph-level detailed balance constraints for all transitions $(G, G') \in \mathcal{E}$, defined as follows:*

$$\tilde{F}(G) p_{\mathcal{E}}(G'|G) = \tilde{F}(G') q_{\mathcal{E}}(G|G') \tag{7}$$

Note that this differs from the usual state-level detailed balance condition:

$$F(s) p_{\bar{\mathcal{A}}}(a|s) = F(s') q_{\bar{\mathcal{A}}}(a|s'). \tag{8}$$

Proof. By Theorem 4.6, state-level detailed balance constraints can be rewritten as graph transition probabilities as follows:

$$|\mathrm{Aut}(G)| F(G) p_{\mathcal{E}}(G'|G) = |\mathrm{Aut}(G')| F(G') q_{\mathcal{E}}(G|G'). \tag{9}$$

Defining $\tilde{F}(G) = |\mathrm{Aut}(G)| F(G)$, the functions $\tilde{F}$, $p_{\mathcal{E}}$, and $q_{\mathcal{E}}$ satisfy the graph-level detailed balance constraints for a given R. □

Theorem F.11 (Restatement of Theorem 5.2)**.** *If the rewards are scaled by $|\mathrm{Aut}(G)|$ and the graph-level detailed balance constraints are satisfied for $p_{\mathcal{E}}$, $q_{\mathcal{E}}$, and $\tilde{F}$, then the corresponding forward policy will sample proportionally to the reward.*

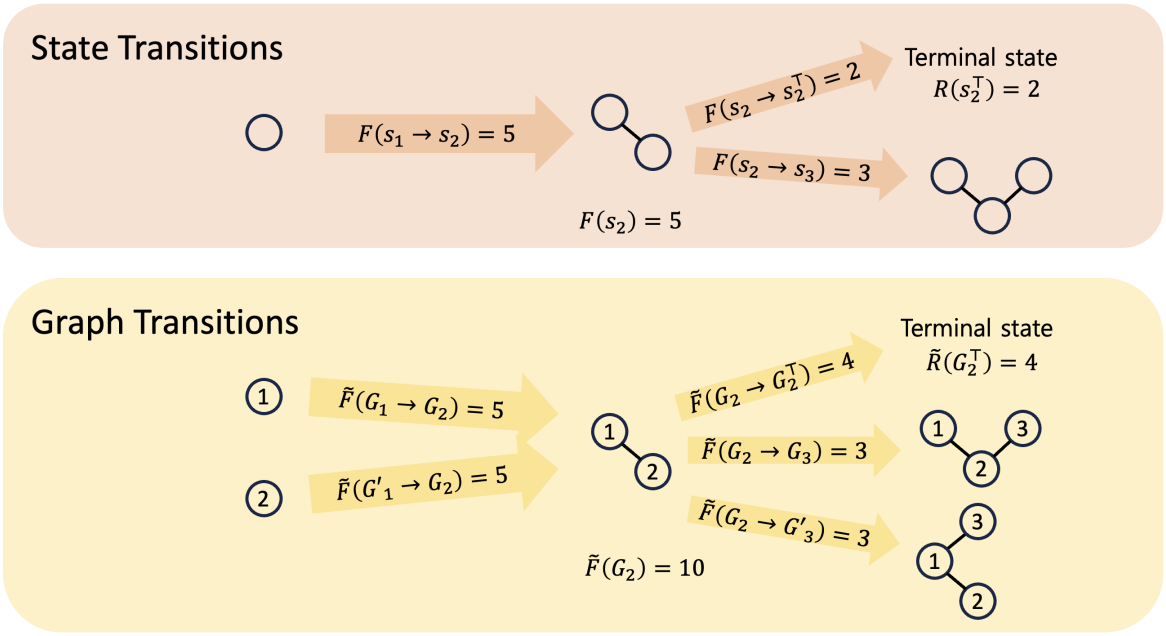


Figure 8: Illustration of the effect of reward adjustment. **Above**: State transitions from and to s_2. **Below**: Graph transitions from and to G_2. Due to the effect of the scaled reward $\tilde{R}$, state flows are also scaled by $|\mathrm{Aut}(G)|$, leading to $\tilde{F}(G) = F(s)|\mathrm{Aut}(G)|$. The edge flows remain unchanged in this figure. Note that the graph-level detailed balance condition holds, while the termination probability is proportional to $R(s)$.

Proof. For a given complete trajectory $G_0, \ldots, G_n$, we have:

$$\tilde{F}(G_0)p_{\mathcal{E}}(G_1|G_0) = \tilde{F}(G_1)q_{\mathcal{E}}(G_0|G_1),$$
$$\ldots$$
$$\tilde{F}(G_{n-1})p_{\mathcal{E}}(G_n|G_{n-1}) = |\mathrm{Aut}(G_n)|R(G_n)q_{\mathcal{E}}(G_{n-1}|G_n).$$

Multiplying the left- and right-hand sides of all the equations, we get:

$$\tilde{F}(G_0)\prod_{t=0}^{n-1} p_{\mathcal{E}}(G_{t+1}|G_t) = |\mathrm{Aut}(G_n)|R(G_n)\prod_{t=0}^{n-1} q_{\mathcal{E}}(G_t|G_{t+1}).$$

Defining $\tilde{F}(G_0) = Z$, this reduces to the state-level trajectory balance condition with corrections as in Corollary 5.1, which ensures $\bar{p}_{\mathcal{A}}(x) \propto R(x)$, as shown by Proposition 1 of Malkin et al. (2022). □

G. Discussion on the Flow-Matching Objective

The first GFlowNet training objective proposed by Bengio et al. (2021) is the Flow-Matching (FM) objective, which requires the flow-matching condition to hold at every state s':

$$\sum_{s:(s,s')\in\mathcal{A}} F(s \to s') = \sum_{s'':(s',s'')\in\mathcal{A}} F(s' \to s'') \tag{10}$$

where $F(s \to s')$ denotes the edge flow. As with the DB and TB objectives, it suffices to scale the final rewards to remove the bias introduced by action equivalence.

Corollary G.1 (FM correction)**.** *Consider the node-by-node generation. Specifically, we restrict the set of graph actions to*

those defined in Appendix C, except for fragment-based actions. Define the graph-level flow-matching condition as:

$$\sum_{G:(G,G')\in\mathcal{E}} \tilde{F}(G \to G') = \sum_{G'':(G',G'')\in\mathcal{E}} \tilde{F}(G' \to G''),$$

where $\tilde{F}(\cdot \to \cdot)$ denotes a permutation equivalent graph-level edge-flow function. If the rewards are given by $\tilde{R}(G) = |\mathrm{Aut}(G)|R(G)$ and the graph-level flow-matching condition holds for all states, then the forward policy induced by $\tilde{F}$ samples terminal states proportionally to the original reward R.

Proof. The detailed balance condition is satisfied whenever the flow-matching condition holds, via the relations:

$$\begin{aligned}
\tilde{F}(G) &= \sum_{G'\in\mathcal{E}(G)} \tilde{F}(G \to G'),\\
p_{\mathcal{E}}(G'|G) &= \frac{\tilde{F}(G \to G')}{\tilde{F}(G)},\\
q_{\mathcal{E}}(G|G') &= \frac{\tilde{F}(G \to G')}{\tilde{F}(G')}.
\end{aligned}$$

Applying Theorem 5.2, the result follows. □

As with the DB objective, FM can be corrected at each state without scaling the reward. Consider a neural network that parameterizes the edge flow function and outputs all possible outgoing edge flows $\tilde{F}(G \to G')$ from a given graph G. Outflows, defined as $\sum_{G'':(G',G'')\in\mathcal{E}} \tilde{F}(G' \to G'')$, can be computed easily by summing the outputs of the neural network. In contrast, computing the inflows for a given graph G' involves two considerations: (1) enumerating parents of G', which may include isomorphic duplicates. To account for this, the total inflow should be divided by the number of duplicates; (2) computing edge flows $\tilde{F}(G \to G')$ for each parent graph G, which may involve equivalence actions. To account for this, the total inflow should be multiplied by the number of equivalent actions. These considerations lead to the following result.

Theorem G.2. *Let G' be the given graph representing the current state. Assuming node-by-node generation, the state-level flow-matching condition can be expressed in terms of the graph-level flow-matching condition as follows:*

$$\sum_{G:(G,G')\in\mathcal{E}} |\mathrm{Aut}(G)|\tilde{F}(G \to G') = |\mathrm{Aut}(G')| \sum_{G'':(G',G'')\in\mathcal{E}(G)} \tilde{F}(G' \to G''),$$

where $\tilde{F}(\cdot \to \cdot)$ denotes a permutation-equivalent edge-flow function.

Proof. The inflows are computed as follows:

$$\begin{aligned}
\sum_{s:(s,s')\in\mathcal{A}} F(s \to s') &= \sum_{G:(G,G')\in\mathcal{E}} \frac{\text{Number of forward orbit-equivalent actions from } G \text{ to } G'}{\text{Number of backward orbit-equivalent actions from } G' \text{ to } G} \tilde{F}(G \to G')\\
&= \sum_{G:(G,G')\in\mathcal{E}} \frac{|\mathrm{Aut}(G)|}{|\mathrm{Aut}(G')|} \tilde{F}(G \to G'),
\end{aligned}$$

where the last equality holds in virtue of Lemma F.8. The first equality follows from the two considerations mentioned: (1) dividing by the number of backward orbit-equivalent actions to account for duplicate parents; (2) multiplying inflows by the number of forward orbit-equivalent actions to account for duplicate actions. On the other hand, the outflows are directly computed as:

$$\sum_{s'':(s',s'')\in\mathcal{A}} F(s' \to s'') = \sum_{G'':(G',G'')\in\mathcal{E}} \tilde{F}(G' \to G'').$$

Rearranging terms, we have the result. □

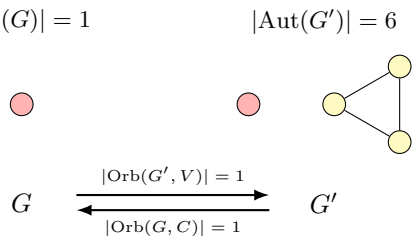


Figure 9: Transition representing the `AddFragment` action.

H. Discussion on the Fragment-based Generation

In the case the action `AddFragment(G, C)`, which adds a fragment C to the existing graph G resulting in $G' = G \cup C$, we must account for the additional symmetries introduced by the fragment. In Figure 9, there is only one way of adding and deleting the fragment from G to G', while the fragment induces 6 symmetries in G'.

We begin by defining the notion of orbits for actions `AddFragment(G, C)` and `RemoveFragment(G', C)`. The orbit of `AddFragment(G, C)` is defined as $\mathrm{Orb}(G, V)$, whose cardinality is 1. This is because each fragment C in the vocabulary leads to a unique next graph when added to G, so the number of forward transition-equivalent actions is 1. For the corresponding backward action `RemoveFragment(G', C)`, however, there may be multiple subgraphs of G' that are isomorphic to C. Hence, we define the orbit as `RemoveFragment(G', C)` is $\mathrm{Orb}(G', C) = \{C' : \exists \pi \in \mathrm{Aut}(G'), \pi(C) = C'\}$. This set captures all subgraphs of G' that are automorphic images of C, i.e., all valid candidates for removal that are equivalent under the symmetry of G'.

Next, we extend Lemma 4.5 to accommodate the fragment-level actions, which account for the symmetries of both the existing graph and the fragment.

Lemma H.1. *Let $G = (V_G, E_G)$ be a graph representing the current state. We consider augmenting the graph G by adding a fragment $C = (V_C, E_C)$. Let $G \cup C = (V_G \cup V_C, E_G \cup E_C)$ denote the union of the two graphs (without any edges connecting G and C). Then, we have:*

$$\frac{\textit{Number of forward orbit-equivalent actions}}{\textit{Number of backward orbit-equivalent actions}} = \frac{|\mathrm{Orb}(G, V)|}{|\mathrm{Orb}(G \cup C, C)|} = \frac{|\mathrm{Aut}(G)| \cdot |\mathrm{Aut}(C)|}{|\mathrm{Aut}(G \cup C)|}.$$

Proof. Since $|\mathrm{Orb}(G, V)| = 1$, we only need to consider $|\mathrm{Orb}(G \cup C, C)|$. The stabilizer $\mathrm{Stab}(G \cup C, C)$ is the set of automorphisms in $\mathrm{Aut}(G \cup C)$ that does not mix the labels of G and C; it acts independently on G and C. Therefore, the order of $\mathrm{Stab}(G \cup C, C)$ is $|\mathrm{Aut}(G)| \cdot |\mathrm{Aut}(C)|$. Using the orbit-stabilizer theorem, we obtain:

$$\begin{aligned} |\mathrm{Orb}(G \cup C, C)| &= \frac{|\mathrm{Aut}(G \cup C)|}{|\mathrm{Stab}(G \cup C, C)|} \\ &= \frac{|\mathrm{Aut}(G \cup C)|}{|\mathrm{Aut}(G)| \cdot |\mathrm{Aut}(C)|}. \end{aligned}$$

□

Using Lemma H.1, we obtain fragment correction formula in Theorem 5.3.

Theorem H.2 (Fragment correction)**.** *Let G represents a terminal state ($[G] \in \mathcal{X}$) generated by connecting k fragments $\{C_1, \ldots, C_k\}$. Then, the scaled rewards to offset the effects of equivalent actions are given by:*

$$\tilde{R}(G) = \frac{|\mathrm{Aut}(G)| R(G)}{\prod_{i=1}^{k} |\mathrm{Aut}(C_i)|}.$$

Proof. Using Lemma H.1, we first derive similar results for Theorem 4.6. If $p_{\mathcal{E}}$ and $q_{\mathcal{E}}$ are permutation-equivariant functions, we have

$$\frac{p_{\bar{\mathcal{A}}}(a|s)}{q_{\bar{\mathcal{A}}}(a|s')} = \frac{|\mathrm{Aut}(G)| \cdot |\mathrm{Aut}(C)|}{|\mathrm{Aut}(G')|} \cdot \frac{p_{\mathcal{E}}(G'|G)}{q_{\mathcal{E}}(G|G')}.$$

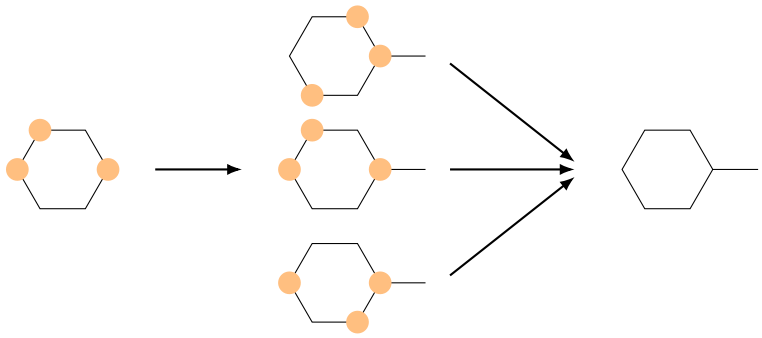


Figure 10: A fragment with attachment points highlighted. Attachment points are designed such that they break symmetries of the fragment. Rightmost graph represent the terminal state where attachment points are removed.

We defined `AddFragment(`G, C`)` as adding a fragment C, resulting in disconnected graph $G \cup C$. To connect between fragments, we use `AddEdge` and Lemma F.7. Then, the results follows from the TB objective written in terms of graph transitions $p_{\mathcal{E}}$ and $q_{\mathcal{E}}$, and the telescoping sum, as in Corollary 5.1. □

Unlike atom-based generation, the fragment terms $|\mathrm{Aut}(C)|$ do not cancel out through a telescoping sum. Therefore, these terms must be explicitly accounted for in the correction, both for reward scaling and estimating the model likelihood.

We need to exercise caution when applying Lemma H.2, as the validity of the result depends on the specifics of the action design. A common practice in the GFlowNet literature is to predefine a set of attachment points for each fragment, where attachment points refer to nodes where new edges can connect to other fragments. These attachment points should be treated as node attributes, even if they are artifacts of the generation process rather than intrinsic properties of the graph. The reason for this treatment is that attachment points constrain the set of allowable actions, including the set of equivalent actions. For example, even if two nodes u and v belong to the same orbit, they should be considered distinct if only one of them is marked as an attachment point.

The complication arises when terminal states are considered. Since attachment points are not intrinsic properties of the graph, conceptually, a graph receives its reward after the attachment points are removed as shown in Figure 10. However, this removal introduces three distinct backward actions from the terminal state in the figure, which complicates the calculation of the backward probabilities.

This issue does not arise, however, if we arrange attachment points in a fragment such that nodes in different orbits (i.e., orbits that consider attachment points as node attributes) remain different even after the attachment points not present. We observe that this holds for the fragments used in Bengio et al. (2021).

I. Relation to Node Orderings

Some previous work on graph generation uses a distribution over permutations (or node orderings) π, treating it as a random variable (Chen et al., 2021; Kong et al., 2023; Wang et al., 2025). If the node (or edge) ordering π is given, the generation path $s_{0:n} = (s_0, \ldots, s_n)$ can be determined. However, the converse is not generally true: for each state sequence $s_{0:n}$, there can be multiple compatible orderings π, complicating the computation of variational lower bound.

Chen et al. (2021) derived an exact formula for the number of node orderings consistent with a given trajectory $s_{0:n}$: it is given by the product of the sizes of the orbits encountered during the generative process, $\prod_{t=0}^{n} |\mathrm{Orb}(s_t, a_t)|$. To make this computation tractable, they approximate the orbit sizes using the color refinement algorithm. This estimate is then used to compute the joint probability $p(s_{0:n}, \pi)$.

However, the joint probability can be computed easily if we consider the corresponding graph sequence $G_{0:n}$, as $p(s_{0:n}, \pi) = \prod_{t=0}^{n-1} p_{\mathcal{E}}(G_{t+1}|G_t)$. This holds because the number of different orderings that induce the same state sequence $s_{0:n}$ corresponds to the number of distinct paths generated by following transition-equivalent actions. In this view, different actions that are transition-equivalent can be interpreted as actions that induce different orderings while resulting in the same sequence of graph states. See Chen et al. (2021) for more details on using node orderings as a random variable.

J. Computational Cost

J.1. Computation Time for Counting Automorphisms

While computing the exact $|\mathrm{Aut}(G)|$ has inherent complexity, this complexity is unavoidable for exact GFlowNets. In practice, fast heuristic algorithms computing $|\mathrm{Aut}(G)|$ often perform well, particularly for relatively small graphs. We provide computation time of $|\mathrm{Aut}(G)|$ for several molecular dataset.

Table 4: Computational cost. "Large" dataset refers to the largest molecules in PubChem, which is used in the paper Flam-Shepherd et al. (2022). Experiments were conducted on an Apple M1 processor.

Dataset	Sample Size	Num Atoms	Compute time (*bliss*)	Compute time (*nauty*)
QM9	133,885	8.8 ± 0.5	0.010 ms ± 0.008	0.019 ms ± 0.079
ZINC250k	249,455	23.2 ± 4.5	0.022 ms ± 0.010	0.042 ms ± 0.032
CEP	29,978	27.7 ± 3.4	0.025 ms ± 0.014	0.050 ms ± 0.076
Large	304,414	140.1 ± 49.4	-	0.483 ms ± 12.600

Compared to sampling trajectories, which involves multiple forward passes through a neural network, the compute time for $|\mathrm{Aut}(G)|$ is negligible. For comparison, we report the speed of molecular parsing algorithms measured using ZINC250k dataset: 0.06 ms ± 0.70 (SMILES $\rightarrow$ molecule) and 0.04 ms ± 0.05 (molecule $\rightarrow$ SMILES). The combination of two parsing steps is often used to check the validity of a given molecule in various prior works. In words, computing $|\mathrm{Aut}(G)|$ is in an order of magnitude faster than validity checking algorithm.

We used the *bliss* algorithm for our experiment. It is easy to use as it is included in the `igraph` package and is fast enough for our purposes. For large molecules, we can still count automorphisms in few milliseconds using the *nauty* package (McKay & Piperno, 2013) as can be seen in the table. We observed that the `pynauty` package does not natively support distinguishing between different edge types, requiring us to transform the input graphs by attaching virtual nodes to handle this. The reported time in the table reflects these preprocessing steps.

While we believe the computation time is already negligible for current applications, we outline two additional strategies to further reduce runtime:

1. **Parallelization.** Data processing tasks can be parallelized across multiple CPUs. Since GFlowNet is an off-policy algorithm, $|\mathrm{Aut}(G)|$ can be computed concurrently with the policy learning.
2. **Approximate correction for large graphs.** For large graphs, fragment-based generation is highly likely to be employed. In such cases, we can apply an approximate correction scheme, as outlined in Appendix H.

J.2. Training Time Comparison

To evaluate the training time, we ran three separate training sessions for each method. Table 5 reports the total training time for each method in the synthetic environment.

Method	1000 Steps (s)	3000 Steps (s)	5000 Steps (s)
Transition Correction	1338 ± 79	4122 ± 168	6859 ± 80
PE (Ma et al., 2024)	1322 ± 44	4001 ± 131	6604 ± 152
Reward Scaling (Ours)	1178 ± 31	3584 ± 97	5999 ± 146
Flow Scaling (Ours)	1176 ± 32	3577 ± 100	5987 ± 168

Table 5: Wall-clock runtime (in seconds) for different methods over increasing training steps. Mean and standard deviation are computed over 3 runs.

While PE is faster than Transition Correction—which performs multiple isomorphism tests per transition—our proposed methods, *Reward Scaling* and *Flow Scaling*, are even more efficient. All timing experiments were conducted using a single processor with a TITAN RTX GPU (24GB) and an Intel Xeon Silver 4216 CPU. These results demonstrate that our methods incur significantly less computational overhead while preserving correctness.

J.3. Scalability Comparison

We evaluated the computation time for each method, varying the number of transitions per trajectory. We sampled 100 random graphs for each horizon length category. We measured the time spent on only the major components of each method. This includes: 1) multiple isomorphism tests for **Transition Correction**; 2) computing positional encodings and matching encodings to identify isomorphic graphs for **PE**; 3) computing automorphisms of the final graph for **Reward Scaling**; 4) computing automorphisms of all intermediate graphs for **Flow Scaling**.

When computing PEs, we used $k = 8$. Note that methods 1) and 2) must be applied for both forward and backward actions. The table below reports the additional computational cost incurred by each method per trajectory, where Vanilla GFlowNets is considered to incur zero additional cost.

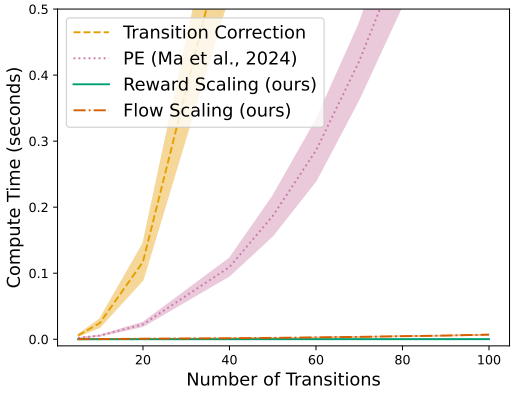


Figure 11

Method	10 Transitions (ms)	50 Transitions (ms)	100 Transitions (ms)
Transition Correction	24.32 ± 6.28	1148 ± 240.3	7354 ± 1288
PE (Ma et al., 2024)	5.49 ± 0.58	186.7 ± 29.87	997.5 ± 133.9
Reward Scaling (Ours)	0.024 ± 0.002	0.063 ± 0.004	0.111 ± 0.008
Flow Scaling (Ours)	0.215 ± 0.025	2.106 ± 0.116	6.975 ± 0.421

Table 6: Runtime comparison for computing correction terms under different methods.

While the cost increases for all methods as the number of transitions grows, our method clearly scales better. It is important to note that the time differences accumulate over the entire training duration. Experiments were performed using Intel Xeon Silver 4216 CPU.

K. Experimental Details

K.1. Graphs-building Environments

For graph-building environments, both for illustrative example and synthetic graphs, we stacked 5 GPS layers with 256 embedding dimensions (Rampášek et al., 2022). To increase representation power, we augmented node features to increase representation power. We augmented node features with one-hot node degree, clustering coefficient, and 8 dimensions of random-walk positional encoding (Dwivedi et al., 2021). Importantly, edge features were computed by summing node features outputted from GNN layers, which, in turn, concatenated with shortest path length between two nodes connected by the edge. This is to prevent representation collapse between different orbits (see Appendix E.2).

Illustrative Example. For the illustrative experiment, homogeneous graphs were constructed edge by edge, allowing only `AddEdge` and `Stop` actions. The initial state consisted of six disconnected nodes, and terminal states corresponded to connected graphs without isolated nodes. The number of terminal states, $|\mathcal{X}|$, is 112.

We trained the models for 30,000 updates using the TB objective. During the first 16,000 steps, each update used a batch of 128 trajectories, comprising 32 samples from the current policy and 96 samples drawn from the replay buffer. We used the Adam optimizer (Kingma, 2014) with the default parameters from `PyTorch` (Paszke et al., 2019) settings, except for the learning rates: 0.0001 for GNN layers and 0.01 for the normalizing constant Z. For the remaining steps, we increased batch size to 256 and annealed the learning rate to 0.00001. To encourage exploration, the policy selected actions uniformly with a probability of 0.9.

Synthetic Graphs. For the Synthetic Graphs experiment, we followed the environment setup described in (Ma et al., 2024). Each node can be one of two types, and graphs can contain up to 7 nodes, resulting in a total of 72,296 terminal

states. The generation process starts from the empty graph, from which the policy incrementally adds nodes and edges. The reward function was also adopted from (Ma et al., 2024).

Models were updated 50,000 times using both the TB and DB objectives. For the TB objective, 64 trajectories were used for updates, with 16 sampled from the behavior policy. For the DB objective, 16 trajectories were sampled per step and converted into transitions, with each update utilizing 512 transitions. We used the Adam optimizer with a learning rate 0.0001 for GNN layers and 0.01 for the normalizing constant Z.

PE Implementation Details. Following Ma et al. (2024), we implemented the positional embedding (PE) method as follows: First, the random walk matrix is computed as $(AD^{-1})^k$, where A is the adjacency matrix, and D is the degree matrix with node degrees on the diagonal. Next, we multiply the matrix by a vector c representing node types. We constructed the vector c by setting each element to $\log(t+2)$, where $t \in \{0, 1, \dots\}$ denotes the node type ID. By varying k from 0 to 7, we obtained 8-dimensional PEs for each node. Edge-level PEs were then computed by summing the embeddings of the corresponding node pairs. Finally, actions sampled from the policy were compared with other candidate actions based on their PEs to identify equivalence.

K.2. Molecule Generation

We conducted experiments on small molecule generation tasks following (Bengio et al., 2021; Jain et al., 2023b). More detailed task descriptions can be found in these previous works. We used a open-source code for tasks.[2] We used a graph transformer architecture (Yun et al., 2019) with the hyperparameters summarized in Table 7 and Table 8. In GFlowNets, the reward exponent β is used to focus sampling on high-reward regions in the state space. The correction is applied after rewards are exponentiated: $C(x)R(x)^\beta$, where $C(x)$ is the correction term.

Table 7: Hyperparameters for atom-based experiments

	Hyperparameters	**Values**
Training	Learning Rate ($p_\mathcal{E}$, Z)	0.0005
	Batch Size (Online)	32
	Batch Size (Buffer)	32
	Uniform Exploration ϵ	0.1
	Gradient Clipping (Layer-wise Norm)	10.0
	Reward Exponent β	1
	Number of Updates	30,000
Model	Architecture	Graph Transformer
	Number of Layers	4
	Number of Heads	4
	Number of Embeddings	128
	Number of Final MLP Layers	1

Evaluation Metrics. Evaluation metrics we presented in Table 1 are defined as follows:

- **Diversity.** The average pairwise Tanimoto distance between molecules sampled from the trained policy.
- **Top K diverse.** Diversity among the top K reward molecules.
- **Top K reward.** The average reward of the top K molecules.
- **Diverse top K.** The average reward of the top K molecules, ensuring that each pair has a Tanimoto distance greater than 0.7.
- **Unique fraction.** The fraction of unique molecules in the generated samples.

[2] `https://github.com/recursionpharma/gflownet`

Table 8: Hyperparameters for fragment-based experiments

	Hyperparameters	Values
Training	Learning Rate ($p_{\mathcal{E}}$)	0.0001
	Learning Rate (Z)	0.001
	Batch Size (Online)	32
	Batch Size (Buffer)	32
	Exploration ϵ	0.1
	Gradient Clipping (Layer-wise Norm)	10.0
	Reward Exponent β	16
	Number of Updates	30,000
Model	Architecture	Graph Transformer
	Number of Layers	5
	Number of Heads	4
	Number of Embeddings	256
	Number of Final MLP Layers	2

We selected $K = 50$, which corresponds to the top 10% of molecules for our evaluation. When reporting rewards, we adjust them to remove the effects of reward scaling and reward exponents.

For the Pearson correlation evaluation presented in Figure 4, terminal states were sampled by uniformly selecting random actions. The model likelihood was computed using Equation (4), with a modified correction term in Theorem 5.3. We set $M = 5$ and used 2,048 samples for the test set.

K.3. Fragment Correction Method

For fragment-based molecule generation, we used a predefined set of fragments and attachment points provided by Bengio et al. (2021). There are a total of 72 fragments, each with a varying number of attachment points. Our method requires pre-computing the number of automorphisms for each fragment. In Figure 12, we present the number of automorphisms for each fragment used in our experiment. As discussed in Appendix H, attachment points were treated as distinct attributes when counting automorphisms.

K.4. Approximate Correction Method

Additionally, we experimented with a simplified version where the correction is applied approximately for the fragment-based task. While we can compute exact correction term as in Equation (3), this approximation provides computational benefits, as it avoids counting automorphisms. Moreover, similar approximations can be easily implemented even for more complex generation schemes that do not fit into Equation (3). The approximation works as follows: we assign a number to each fragment based on how many equivalent actions it is likely to incur during generation. We adjust the final rewards by dividing them by the product of the assigned numbers N for the constituent fragments: $R(G)/\prod_{i=1}^{k} N(C_i)$.

We assigned the number N to each fragment based on how likely it is to incur forward equivalent actions. This is because fragments that incur multiple forward equivalent actions are more likely to be selected if no adjustment is applied. For example, cyclohexane (`C1CCCCC1`) has six attachment points, all in the same orbit, so it will always incur at least six forward equivalent actions in subsequent steps. In contrast, even if a fragment is highly symmetric, if it has only one attachment point, it will incur no equivalent actions. We assigned $N = 1$ to such fragments.

Assuming backward equivalent actions are relatively rare in fragment-based generation, this approximation should closely match the unbiased correction. These numbers were assigned through visual inspection of the fragments. The full set of fragments and their assigned numbers for the approximate correction is provided in Figure 13.

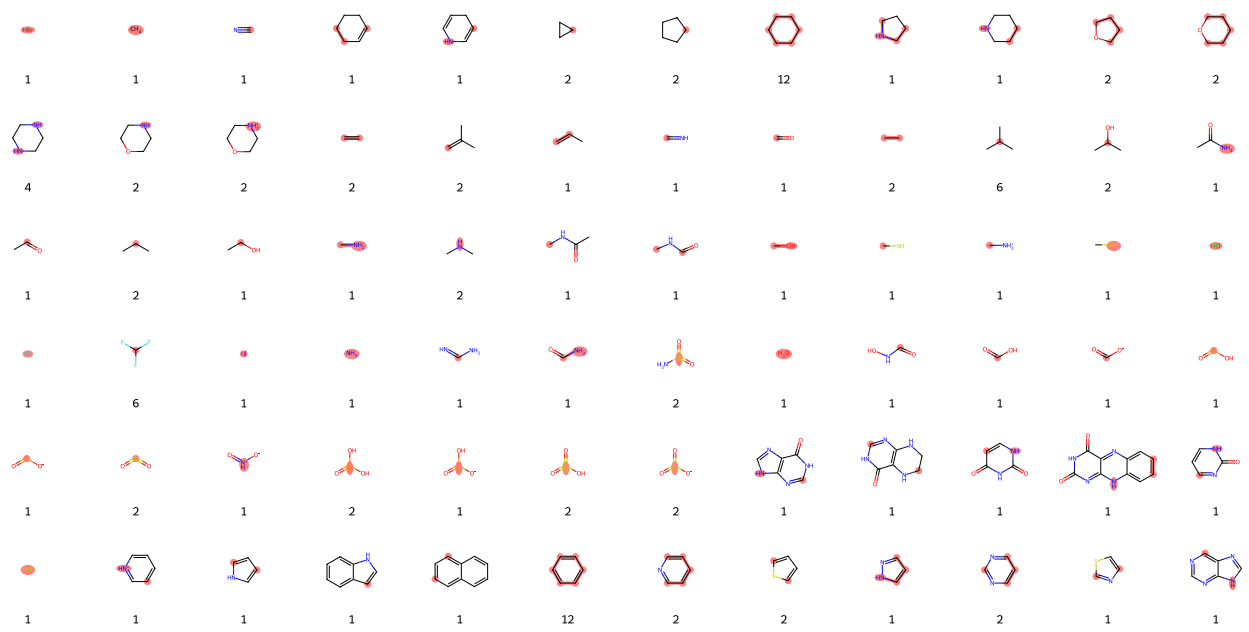


Figure 12: Predefined fragment set used for the fragment-based task. Attachment points, where a single bond can connect to another fragment, are highlighted in red. The numbers indicate the order of the automorphism group of each fragment, $|\mathrm{Aut}(C)|$.

L. Additional Experimental Results

L.1. Synthetic Graphs

In addition to the clique environment presented in the main text, we conducted similar experiments under a different reward structure. We restricted the maximum number of nodes and edges to 10, and limited the maximum node degree to 4, thereby constraining the state space to $|\mathcal{X}| = 2,999$. Rewards were defined as 1 + (number of cycles). The hyperparameters are not tuned and remain the same as those explained in Appendix K.

For the TB objective shown in Figure 14 (a), the PE method slightly underperforms compared to Reward Scaling, which is consistent with the results in the clique environment. In contrast, for the DB objective shown in Figure 14 (b), where intermediate probabilities play a crucial role, the PE method performs comparably to Reward Scaling. In all cases, Transition Correction and Flow Scaling perform the best.

L.2. Molecule Generation

We investigated which fragments were sampled by each method. We sampled 5,000 terminal states from trained models, resulting in 44,974 and 44,978 fragments sampled from vanilla and corrected model, respectively. Symmetric fragments were found to be sampled more frequently in vanilla model, which aligns with our projection, as the fragment correction in Equation (3) penalizes symmetric components. However, the proportions of fragments between the two methods are not exactly proportional to the magnitude of the corrections, as some fragments are more likely to occur together (they are not independent).

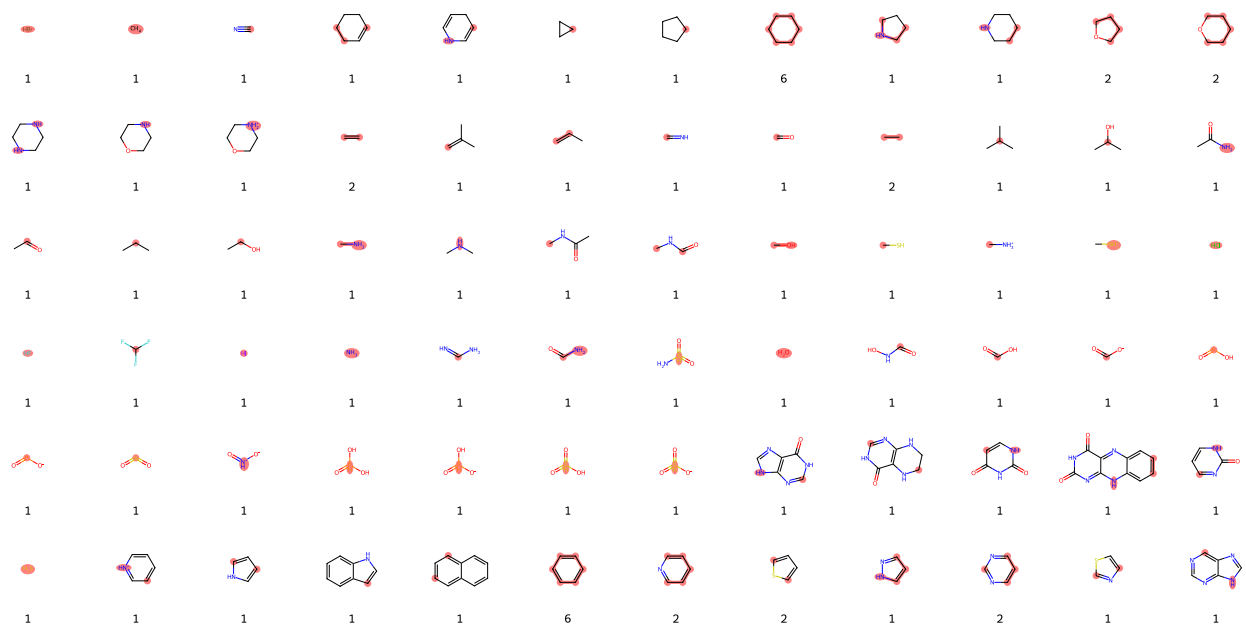
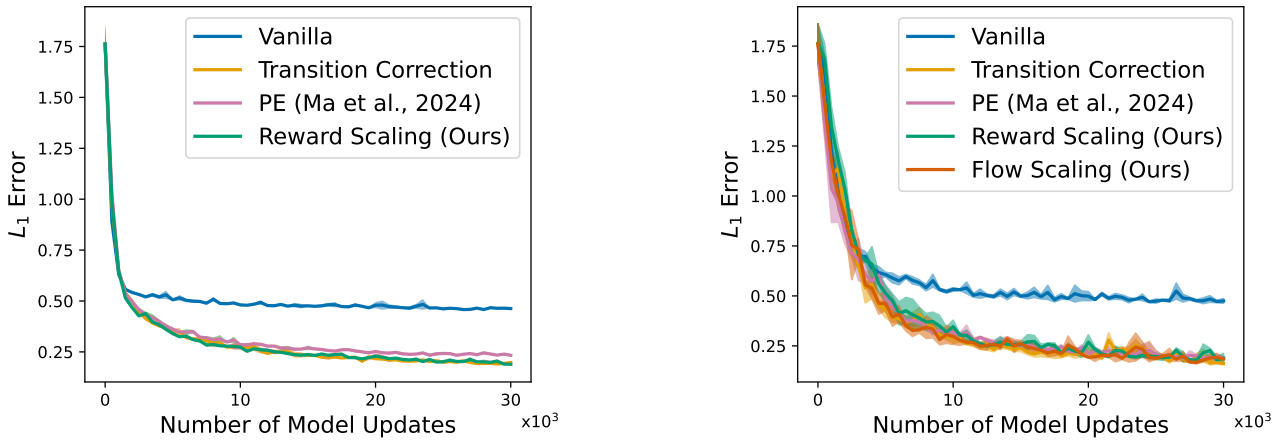


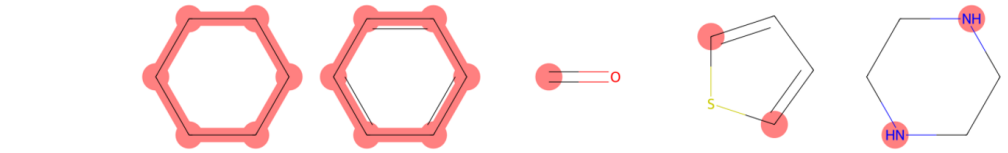
Figure 13: Predefined fragment set used for the fragment-based task. Attachment points are highlighted in red. The numbers below each molecule are used for approximate correction.



(a) Synthetic (TB)

(b) Synthetic (DB)

Figure 14: Results on synthetic experiments, where rewards are given based on the number of cycles in the graph.



Vanilla	5220	4821	101	531	744
Reward Scaling	1042	1038	63	349	495

Figure 15: The number of sampled fragments from 5,000 terminal states for vanilla model and corrected model. We display the 5 fragments that were sampled most disproportionately. Attachment points are highlighted in red.