NOTE:

- Use consistent mathematical notation throughout
- Include clear figures and diagrams
- Provide code snippets for key implementations
- Reference related work appropriately
- Writing style: maintain an academic tone while ensuring readability. Use precise technical language but explain complex concepts clearly. Include examples and visualizations to aid understanding.

TODO: Add title page.	*TODO:* Add an abstract covering the thesis.

Contents

1 Introduction

Many real-world applications present a fundamental challenge that current reinforcement learning (RL) methods struggle to address effectively: the problem of delayed and sparse rewards.

> **Delayed and Sparse Rewards:** Learning scenarios where meaningful feedback signals (rewards) are provided only far after a long sequence of actions, and where most actions yeild no immediate feedback.
>
> *Example: In drug discovery, the effectiveness of a designed molecule can only be evaluated after its complete synthesis, with no intermediate feedback during the design process.*

Consider, for instance, the process of drug design, where a reinforcement learning agent must make a series of molecular modifications to create an effective compound. The value of these decisions — the drug's efficacy — can only be assessed once the entire molecule is complete. Similarly, in robotics tasks like assembly or navigation, success often depends on precise sequences of actions where feedback is only available upon having completed the entire task.

Traditional reinforcement learning algorithms face two critical limitations in such environments:

1. **Credit Assignment:** When rewards are delayed, the algorithm struggles to correctly attribute success or failure to specific actions in a long sequence. This is analogous to trying to improve a

chess strategy when only knowing the game's outcome, without understanding which moves were actually decisive.

2. **Exploration Efficiency:** With sparse rewards, random exploration becomes highly inefficient. An agent might need to execute precisely the right sequence of actions to receive any feedback at all, making random exploration about as effective as searching for a needle in a haystack.

This thesis investigates a novel approach to addressing these challenges through the comparison of two promising methodologies: **Generative Flow Networks** (GFlowNets) [1] and **Bayesian Exploration Networks** (BEN) [CITATION NEEDED]. These approaches represent fundamentally different perspectives on handling uncertainty and exploration in reinforcement learning:

1. GFlowNets frame the learning process as a flow network, potentially offering more robust learning in situations with multiple viable solutions.
2. BENs leverages Bayesian uncertainty estimation to guide exploration more efficiently, potentially making better use of limited feedback.

By comparing these approaches, we aim to understand their relative strengths and limitations in environments with delayed and sparse rewards, ultimately contributing to the development of more efficient and practical reinforcement learning algorithms. Our investigation focuses specifically on examining these methods in carefully designed environments that capture the essential characteristics of delayed and sparse reward scenarios while remaining tractable for systematic analysis.

1.1 Research Objectives and Contributions

This thesis aims to advance our understanding of efficient learning in sparse reward environments through three primary objectives:

1. **Comparative Analysis:** Conduct a rigorous empirical comparison between GFlowNets and Bayesian Exploration Networks in standardized environments with delayed rewards.
2. **Hypothesis Testing:** Investigate whether BEN's Bayesian exploration strategy leads to more efficient learning compared to GFlowNets in highly delayed reward scenarios, particularly during early training stages.
3. **Algorithmic Understanding:** Analyze the underlying mechanisms that drive performance differences between these approaches, focusing on their handling of uncertainty and exploration.

The contributions of this work include:

- A comprehensive empirical evaluation using the n-chain environment with varying degrees of reward delay.
- Insights into the relative strengths and limitations of Bayesian and flow-based approaches to exploration.
- Implementation and analysis of both algorithms with comparisons.

1.2 Thesis and Structure

The remainder of this thesis is structured as follows:

NOTE:

Check that these titles correctly correspond to their reference.

Section 2: Preliminaries provides the theoretical foundations of reinforcement learning and explores existing approaches to handling sparse rewards. This chapter establishes the mathematical framework and notation used throughout the thesis.

Section 3: Theoretical Framework presents our hypothesis and analytical approach. We develop the mathematical foundations for comparing GFlowNets and BEN, with particular attention to their theoretical guarantees and limitations.

Section 4: Experimental Design details our testing methodology, including environment specifications, evaluation metrics, and implementation details. This chapter ensures reproducibility and clarity in our experimental approach.

Section 5: Results and Analysis presents our findings, including both quantitative performance metrics and qualitative analysis of learning behaviors. We examine how each algorithm handles the exploration-exploitation trade-off and adapts to varying levels of reward sparsity.

Section 6: Future Research ...

Section 7: Conclusion summarizes our findings, discusses their implications for the field, and suggests directions for future research.

2 Preliminaries

NOTE:

- Fundamentals of reinforcement learning
  - Contextual reinforcement learning
  - Bayesian reinforcement learning
  - Markov Decision Processes
  - Q-learning and temporal difference methods
- Sparse reward challenges
- Survey of existing approaches
  - GFlowNets
  - Deep exploration networks (BEN)
  - Comparison of methodologies

2.1 Flow Networks

GFlowNets [1] rely on the concept of flow networks. A flow network is represented as a directed acyclic graph $G = (\mathcal{S}, \mathcal{A})$, where $\mathcal{S}$ represents the state space and $\mathcal{A}$ represents the action space.

> **Flow Network:** A directed acyclic graph with a single source node (initial state) and one or more sink nodes (terminal states), where flow is conserved at each intermediate node.
>
> *Example: In molecular design, states represent partial molecules and actions represent adding molecular fragments.*

2.1.1 States and Trajectories

We distinguish severel types of states:

- An initial state $s_0 \in \mathcal{S}$ (the source);
- Terminal states $x \in \mathcal{X} \subset \mathcal{S}$ (sinks);
- Intermediate states that form the pathways from source to sinks.

A trajectory τ represents a complete path through the network, starting at s_0 and ending at some terminal state x. Formally, we write a trajectory as an ordered sequence $\tau = (s_0 \to s_1 \to \ldots \to s_n = x)$, where each transition $(s_t \to s_{t+1})$ corresponds to an action in $\mathcal{A}$.

2.1.2 Flow Function and Conservation

The *trajectory flow function* $F : \mathcal{T} \to \mathbb{R}_{\geq 0}$ assigns a non-negative value to each possible trajectory. From this flow function, two important quantities are derived:

1. **State flow**: For any state s, its flow is the sum of flows through all trajectories passing through it:

$$F(s) = \sum_{s \in \tau} F(\tau). \tag{1}$$

2. **Edge flow**: For any action (edge) $s \to s'$, its flow is the sum of flows through all trajectories using that edge:

$$F(s \to s') = \sum_{\tau = (\ldots \to s \to s' \to \ldots)} F(\tau). \tag{2}$$

These flows must satisfy a conservation principle known as the *flow matching constraint*:

> **Flow Matching:** For any non-terminal state s, the total incoming flow must equal the total outgoing flow:
>
> $$F(s) = \sum_{(s'' \to s) \in \mathcal{A}} F(s'' \to s) = \sum_{(s \to s') \in \mathcal{A}} F(s \to s'). \tag{3}$$

2.1.3 Markovian Flow

The flow function induces a probability distribution over trajectories. Given a flow function F, we define $P(\tau) = \frac{1}{Z}F(\tau)$, where $Z = F(s_0) = \sum_{\tau \in \mathcal{T}} F(\tau)$ is the *partition function* — i.e., the total flow through the network.

> **Markovian Flow:** A flow is *Markovian* when it can be factored into local decisions at each state. This occurs when the following exist:
>
> 1. Forward policies $P_F(- \mid s)$ over children of each non-terminal state s.t.
>
> $$P(\tau = (s_0 \to \ldots \to s_n)) = \prod_{t=1}^{n} P_F(s_t \mid s_{t-1}). \tag{4}$$
>
> 2. Backward policies $P_B(- \mid s)$ over parents of each non-initial state s.t.
>
> $$P(\tau = (s_0 \to \ldots \to s_n) \mid s_n = x) = \prod_{t=1}^{n} P_B(s_{t-1} \mid s_t). \tag{5}$$

The Markovian property allows us to decompose complex trajectory distributions into simple local decisions, making learning tractable while maintaining the global flow constraints [CITATION NEEDED].

2.2 GFlowNets

GFlowNets [1] are a novel approach to learning policies that sample from desired probability distributions. They frame the learning process as the discovering of a flow function that makes the probability of generating any particular object proportional to its reward.

Given a reward function $R : \mathcal{X} \to \mathbb{R}_{\geq 0}$ defined over the set of terminal states $\mathcal{X}$, GFlowNets aim to approximate a Markovian flow F on the graph G s.t. $F(x) = R(x)$ for all $x \in \mathcal{X}$.

> **GFlowNet:** [2] defines a GFlowNet as any learning algorithm that discovers flow functions matching terminal state rewards, consisting of:
>
> 1. A model that outputs:
>   - Initial state flow $Z = F(s_0)$;
>   - Forward action distributions $P_F(- \mid s)$ for non-terminal states.
> 2. An objective function that, when globally minimized, guarantees $F(x) = R(x)$ for all terminal states.
>
> *Example: In molecular design, this ensures that high-reward molecules are genered more frequently, while maintaining diversity through exploration of multiple pathways.*

The power of GFlowNets lies in their ability to handle situations where multiple action sequences can lead to the same terminal state — a common scenario in real-world applications like molecular design or image synthesis. Unlike traditional RL methods that focus on finding a single optimal path, GFlowNets learn a distribution over all possible paths proportional to their rewards.

2.2.1 Learning Process

The learning process of GFlowNets involves iteratively improving both flow estimates and the policies. The forward policy of a GFlowNet can sample trajectories from the learned Markovian flow F by sequentially selecting actions according to $P_F(- \mid s)$. When the training converges to a global minimum of the objective function, this sampling process guarantees that $P(x) \propto R(x)$. That is, the probability of generating any terminal state x is proportional to its reward $R(x)$. This property makes GFlowNets particularly well-suited for:

1. **Diverse Candidate Generation:** Rather than converging to a single solution, GFlowNets maintain a distribution over solutions weighted by their rewards.
2. **Multi-Modal Exploration:** The flow-based approach naturally handles problems with multiple distinct solutions of similar quality.
3. **Compositional-Structure Learning:** By learning flows over sequences of actions, GFlowNets can capture and generalize compositional patterns in the solution space.

To achieve this, GFlowNets employ various training objectives, with *trajectory balance* [2] being one such particularly effective objective.

2.2.2 Trajectory Balance

Trajectory balance focuses on ensuring consistency across entire trajectories, instead of matching flows at every state (which can be computationally expensive).

> **Trajectory Balance:** A principle that ensures the probability of generating a trajectory matches its reward by maintaining consistency between forward generation and backward reconstruction probabilities.

Consider a Markovian flow F that induces a distribution P over trajectories according to $P(\tau) = \frac{1}{Z}F(\tau)$. The forward policy P_F and backward policy P_B must satisfy the following *trajectory balance constraint* [2]

$$Z \prod_{t=1}^{n} P_F(s_t \mid s_{t-1}) = F(x) \prod_{t=1}^{n} P_B(s_{t-1} \mid s_t). \tag{6}$$

That is to say, the probability of constructing a trajectory forward should match the probability of reconstructing it backward, scaled by the appropriate rewards.

2.2.3 Trajectory Balance as an Objective

To convert the trajectory balance function into a training objective, we introduce a parametrized model with parameters θ that outputs:

1. A forward policy $P_F(- \mid s; \theta)$;
2. A backward policy $P_B(- \mid s; \theta)$;
3. A scalar estimate Z_θ of the partition function.

For any complete trajectory $\tau = (s_0 \to \ldots \to s_n = x)$, we define the *trajectory balance loss* as

$$\mathcal{L}_{\text{TB}}(\tau) = \left(\log \frac{Z_\theta \prod_{t=1}^{n} P_F(s_t \mid s_{t-1}; \theta)}{R(x) \prod_{t=1}^{n} P_B(s_{t-1} \mid s_t; \theta)} \right)^2. \tag{7}$$

This loss captures how well our model satisfies the trajectory balance constraint. When the loss approaches zero, our model has learned to generate samples proportional to their rewards. In practice, we compute this loss in the log domain to avoid numerical stability, as suggested by [2]:

$$\mathcal{L}_{\text{TB}}(\tau) = \left(\log Z_\theta + \log \sum_{t=1}^{n} P_F(s_t \mid s_{t-1}; \theta) - \log R(x) - \log \sum_{t=1}^{n} P_B(s_{t-1} \mid s_t; \theta) \right)^2. \tag{8}$$

[2] also remarks that a simplificatoin of Equation 7 occurs in tree-structured state spaces (when G is a directed tree), where each state has exactly one parent. In such cases, the backward policy becomes deterministic ($P_B = 1$), reducing the loss function to

$$\mathcal{L}_{\text{TB}}(\tau) = \left(\log \frac{Z_\theta \prod_{t=1}^{n} P_F(s_t \mid s_{t-1}; \theta)}{R(x)} \right)^2, \tag{9}$$

which can be exploited for the n-chain environment.

The model is trained by sampling trajectories from a training policy π_θ — typically a tempered version of $P_F(- \mid -; \theta)$ to encourage exploration — and updating parameters using stochastic gradient descent: $\theta \leftarrow \theta - \alpha \mathbb{E}_{\tau \sim \pi_\theta} \nabla_\theta \mathcal{L}_{\text{TB}}(\tau)$.

3 Theoretical Framework

NOTE:

- Hypothesis development
- Problem formulation
  - Mathematical notation and definitions
  - Assumptions and constraints
- Proposed solution approach

4 Experimental Design

NOTE:

- Test environments
  - N-Chain implementation
- Evaluation metrics
  - Sample efficiency (steps needed to reach optimal policy) - measure by objective loss over training?
  - Final performance (average success/reward rate) - measure by difference between sample distribution and true distributions? Otherwise, this probably doesn't make sense for the semi-deterministic n-chain environment, as we know we'll succeed in n steps.
  - Exploration behavior (state coverage over time) - maybe not so interesting for the simple n-chain environment, but not entirely uninteresting either.
- Implementation details
  - Network architectures
  - Training procedures
    - For GFlowNets, mention tempered exploration during training (off-policy training)
  - Hyperparameter selection

4.1 Test Environment

TODO:

- Describe the n-chain environment:
  - a chain of length n
  - the chain has a branch point from which two branches emerge
  - last state is always a terminal state
  - terminal states have a disignated reward
- Describe the action space:
  - actions include a *forward* action;
  - a *left branch* or *right branch* choice at branch point.
- *NOTE:*

  The *terminal stay* action is probably inconsequensial.

4.2 Evaluation Metrics

TODO:

- Describe how the sample efficiency is measured.
  - Maybe the objective loss over training (how many steps to convergence)
- Describe the final performance:
  - In my n-chain implementation, a terminal state will always be reached, so average success rate wouldn't make sense as a metric (as success will always be 100%).
  - Instead, for GFlowNets, this should probably be evaluated as the difference from the true distribution (the expected distribution).
  - I.e., with rewards 10 and 5 for two terminal states, the true distribution would be $\frac{1}{3}$ samples with reward 5 and $\frac{2}{3}$ samples with reward 10.
- Describe exploration behavior:
  - State coverage over time.
  - This will probably require a higher number of branches for any interesting metrics.

4.3 Implementation Details

TODO:

- Network architectures
  - GFlowNets:
    - Multi-layer perceptron for state encoding
      - Input: State tensor of shape [batch_size, state_dim]
      - Output: State encoding of of shape [batch_size, hidden_dim]
    - Single layer for forward policy
      - Input: State encoding of shape [batch_size, hidden_dim]
      - Output: Forward policy [batch_size, num_actions]
    - Single layer for backward policy
      - Input: State encoding of shape [batch_size, hidden_dim]
      - Output: Backward policy [batch_size, num_actions]
        - Note: for the n-chain environment, the graph is a directed tree and so each state can have at most one parent, meaning that only one action is possible for each state in the backward policy, as only one action could have lead from the previous state to this state (as actions are represented by edges)
    - Parameter (scalar) for log Z function approximation
- Training procedures
  - For GFlowNets, mention tempered exploration during training (i.e., off-policy training)
    - Mention epsilon parameter for temperature control in guided exploration
- Hyperparameter selection

5 Results and Analysis

NOTE:

- Quantitative results
  - Performance comparisons
  - Statistical analysis
- Qualitative analysis
  - Exploration patterns
  - Learning behavior
- Discussion of findings

6 Future Research

NOTE:

Everything I think I could have done better essentially.

7 Conclusion

NOTE:

- Summary of contributions
- Key insights
- Future work directions

Bibliography

[1] E. Bengio, M. Jain, M. Korablyov, D. Precup, and Y. Bengio, "Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation." [Online]. Available: https://arxiv.org/abs/2106.04399

[2] N. Malkin, M. Jain, E. Bengio, C. Sun, and Y. Bengio, "Trajectory balance: Improved credit assignment in GFlowNets." [Online]. Available: https://arxiv.org/abs/2201.13259