Flow with Mario: Using GFlowNets to Generate Mario Levels

Nisarg Parikh
parikh.nis@northeastern.edu
Northeastern University
Boston, Massachussets, USA

ABSTRACT

Generative Flow Networks(or GFlowNets) are a novel architecture introduced to solve the problem of sampling diverse candidates for a generative process [1]. They are used for many generative applications where we want to generate multitudes of diverse samples. They have been used for generating molecules for drug discovery and generating graphs for solving combinatorial problems. Another natural extention for GFlowNets is Proceedural Content Generation, more specifically Games. For this project we focus on generating Super Mario Bros levels. A lot of work has been done with Mario levels using supervised learning, but most of them require a lot of training data, which there is not much available of with the original Super Mario Bros levels. Hence we begin a line of work using a Reinforcement Learning-like objective to generate Mario Levels.

KEYWORDS

Generative, Flow, Network, GFlowNet, Sampling

ACM Reference Format:
Nisarg Parikh. 2018. Flow with Mario: Using GFlowNets to Generate Mario Levels. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Creating challenging and exciting game levels procedurally or automatically is a difficult endeavour as it requires us to either have a way to model human enjoyment or have a metric for fun, both of which are difficult. But suppose we did have that, we still do not have a great way to make use of that metric. Most generative processes have big action spaces and state spaces. And the generative process for games itself is in active development, as the way and environment humans develop games in allows for a lot of flexibility and freedom in their conception.

With regular RL methods we can generate levels, but one issue with a regular RL objective is it aims to maximize observed rewards, which is an issue because we do not have an exact reward function and even if we do, we don't want a really good level or a few really good levels. We want to be able to generate all possible good levels, which is not necessarily possible, as RL based methods seem to stagnate to the same set of high reward levels [7] it finds initially unless we reward level diversity or add a regularization penalty for similar levels to currently generated levels.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

Hence we need a way to explore the subdomain of high reward levels for diverse sampling and generate a bigger set of candidates for manual evaluation by domain experts.

A recent way to do this came up in a recent paper by Bengio et.al. [1] which introduced Generative Flow Networks(GFlowNets) which introduces a new way to sample composite objects. GFlowNets describe a new formulation of a Markov Descision Process converted into a Flow Network which gets us a Markovian Flow Network. They also describe a way to learn how to sample from the GFlowNet as calculating the flow and and storing the network itself can be computationally expensive.

Generative Flow Networks sample trajectories to complete levels proportional to the reward, this motivates them to find many high reward levels. And hence in a way explore in high reward states.

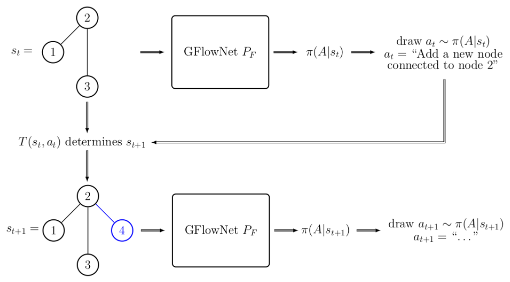
$s_t =$
1
2
3
GFlowNet P_F
$\pi(A|s_t)$
draw $a_t \sim \pi(A|s_t)$
$a_t =$ "Add a new node connected to node 2"
$T(s_t, a_t)$ determines s_{t+1}
$s_{t+1} =$
1
2
3
4
GFlowNet P_F
$\pi(A|s_{t+1})$
draw $a_{t+1} \sim \pi(A|s_{t+1})$
$a_{t+1} =$ "…"

Figure 1: A step in the generative process with GFlowNets

2 RELATED WORK

A Flow network is a directed graph where each edge has a capacity and a flow. They are typically used to model problems involving the transport of items between locations, using a network of routes with limited capacity. Flow functions model the net flow of units between pairs of nodes, and are useful when asking questions such as what is the maximum number of units that can be transferred from the source node s to the sink node t? The amount of flow between two nodes is used to represent the net amount of units being transferred from one node to the other. Here they have specific properties which benefits converting an MDP into a GFlowNet:

- Traditional MDPs can struggle with situations where multiple actions can lead to the same state (non-injective cases). Flow networks don't require unique transitions for each state. The focus on flow allows for multiple incoming edges to a single node, naturally accommodating non-injective cases [1].

- MDPs typically require on-policy learning, meaning the data used to train the model comes from the same policy the agent will follow during generation. GFlowNets can work with data generated from different policies (off-policy learning) [6]. This flexibility allows for leveraging existing datasets or exploring diverse policy options during training.
- MDPs primarily focus on finding the optimal policy that maximizes long-term rewards. While valuable, this can limit exploration of alternative solutions that might also be good. GFlowNets[1], by focusing on matching the flow between states, can explore a wider range of possibilities and uncover diverse good solutions.
- MDPs can become computationally expensive for problems with large or continuous state spaces. While not always the case, flow networks might offer some computational advantages in specific scenarios. By focusing on flow between states rather than transition probabilities for every action[2], flow networks might require less memory in some situations.

The way we convert a MDP into a Flow Network is by "injecting" flow from the terminal states.

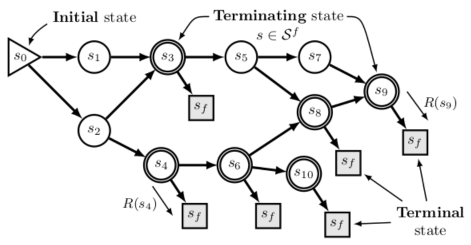

Figure 2: Construction of a GFlowNet

We use the following objectives to train a 2 layer CNN:

- Flow Matching [1]
- Stable Flow Matching [3]
- Trajectory Balance [5]

Generative Flow Networks have been used for tasks such as:

- Combinatorial Optimization[10]
- Molecule Generation [1]
- Scheduling [9]
- N-Gram Sequence Design [4]

3 SYSTEM OVERVIEW

My contribution to the PCGML space was as follows:

- The input representation of Mario Levels
- The sampling mechanism which allows us to begin with empty levels and place tiles
- An implementation of GFlowNets which can be used to learn how to make levels

For both input representations, the tiles are mapped to a token number and we convert each tile from a string representation to a number, so we can input them to a CNN. The input representations are as follows:

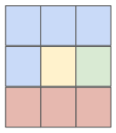

Figure 3: Naive Feature Representation

- Naive
  With the naive input representation for a level of NxM dimensions, each tile space has the numeric representation of the tile and then we normalize them.
- Feature Maps

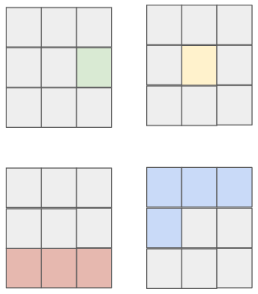

Figure 4: Feature Maps Representation

For the feature maps, we convert a NxM map into a NxMxK map, where we have K total type of tiles we can place. Instead of normalizing, we place a 1 wherever a tile of that type exists in the actual NxM map.

To get the sampling mechanism I added 2 tile types:

- Empty Tile: The sampling process always starts with the level full of empty tiles. They are a placeholder such that we always have a strict stopping criteria of having placed all tiles, instead of stopping arbitrarily by adding a "stop" action or placing an pre-determined number of tiles.
- Next Tile: This tile specifies where we want to place the next tile, the "empty" level always starts with atlast 1 Next tile.

The sampling mechanism is also augmented by a seed level:

- Floor on the bottom
- Sky on the left
- Sky on the right

The top was left empty as if we put sky on the top, for some level heights, it is always playable, as the agent can just jump above the placed tiles and if its all floors, it is just that much harder to generate levels.

There are 3 sampling mechanisms:

- From the center: The sampling happens starting a random empty tile from the center. Where we replace an empty tile with a next tile. And each subsequent next tile is placed around a non-empty tile randomly.
- From the bottom left: This sampling mechanism behaves similarly to the previous one, except the starting next tile is

placed by replacing the bottom left empty tile. And the next subsequent next tiles are placed by replacing any empty tile above or to the right to a currently placed non-empty tile.
- From the left: Following the trend from before, we start by placing a next tile on the left-most wall. And then continue the following sampling by replacing empty tiles by next tiles which are next to non-empty tiles.

The playability metric from the Summerville A* level solver is used as the reward signal. This metric ranges from 0 to 1, where 1 means the level is fully playable.

4 TRAINING DATA

This is a case of PCGRL, hecne the training data was in a sense the levels generated by the model itself.

The tiles were gathered from the VGCL level database [8].

5 TRAINING THE MODEL

Generative Flow Networks can be trained using any architecture which can allow for Autoregressive generation. In this case, as the input is an image, we use a CNN to "Autoregressively" generate Mario Levels. This is done by letting a CNN output the next tile to place in the place of the "next" tile.

The training loop for GFlowNets is as follows:

(1) Sample flow for the level
(2) Sample parent states and actions
(3) Calculate flow coming into a state(i.e. from parent states with parent actions)
(4) Calculate flow going out from a state
(5) Repeat till full level is generated
(6) Calculate the loss and update the agent policy every few steps

The generation process is similar to any other PCGRL method. But the major difference comes in the objective we try to optimize.

Training the model included generating levels and optimizing it according to the following objective functions:

(1) Flow Matching
This objective converts the constraint:

$$\sum_{s,a:T(s,a)=s'} F(s,a) = R(s') + \sum_{a' \in \mathcal{A}(s')} F(s',a') \tag{1}$$

Into a Temporal Difference-like loss objective for each state the agent obseves in their trajectory to a complete level.

$$L_{FM}(\hat{F}, s') = \log \left(\frac{\delta + \sum_{s \in Par(s')} \hat{F}(s \rightarrow s')}{\delta + R(s') + \sum_{s'' \in Child(s') - s_f} \hat{F}(s' \rightarrow s'')} \right)$$

And we additively aggregate this observed loss objective to calculate the complete loss function.

$$\mathcal{L}_{FM} = \sum_{s \in \mathcal{S}} L_{FM}(\hat{F}, s) \tag{2}$$

This loss function considers all injected flow from the rewards we do not observe at the end of the trajectory as well. Which is not the case for Trajectory balance.

(2) Stable Flow Matching
This loss function uses a divergence based objective definition to derive a stable loss in presence of cycles.

$$\mathcal{L} = \mathbb{E} \sum_{t=1}^{\tau} \left\{ \log \left[1 + \epsilon |F_{in}(s) - F_{out}(s)|^{\alpha} \right] \times (1 + \eta(F_{in}(s) + F_{out}(s)))^{\beta} \right\}$$

This is an extension to the Flow Matching loss function.

(3) Trajectory Balance
The trajectory balance takes the constraint:

$$\hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t|s_{t-1}) = R(s_n) \prod_{t=1}^{n+1} \hat{P}_B(s_{t-1}|s_t)$$

It is a constraint over the entire trajectory, and here we multiplicatively aggregate the probability of arriving at a specific terminal state with the specific trajectory which was observed, and only it considers the flow that would be observed in the trajectory due to the flow injected by the reward observed in the observed terminal state, ignoring the flow injected by the other states as opposed to the way Flow Matching is formulated. Here $\hat{Z}$ is the predicted flow across that trajectory.
We convert this constraint into a loss function as follows:

$$L_{TB}(\hat{Z}, \hat{P}_F, \hat{P}_B, \tau) = \left(\log \frac{\hat{Z} \prod_{t=1}^{n+1} \hat{P}_F(s_t|s_{t-1})}{R(s_n) \prod_{t=1}^{n+1} \hat{P}_B(s_{t-1}|s_t)} \right)^2$$

6 GENERATING CONTENT

Generating content was straight forward:

(1) We start with a grid of "empty" tiles/ or a seed level.
(2) Placing a "next" tile somewhere in the level according to one of the following sampling mechanisms:
a. From the center
b. From the bottom left
c. From the left
(3) Place a tile in place of the "next tile" proportional to the flow described by the GFlowNet.
(4) Sample another "next" tile according to the chosen sampling mechanism till no "empty" tiles exist.

7 EVALUATION

Evaluation was done by the metrics of loss and rewards. There were a total of 22 experimental setups with the following hyperparameters:

- Input Representation(Naive vs Feature Maps)
- Objective function used(Flow Matching, Stable Flow Matching or Trajectory Balance)
- Sampling mechanism(From center, from the bottom left or from the left)
- Tile set(Full tile set or limited tile set)

Due to the dearth of results and only unique results shall be included in this paper, the rest being included as artefacts. All plots are a moving average of the metric in question rather than the exact values for ease of analysis. And all losses are on the log scale.

All setups converge to a loss something similar to looking like the following case with Flow Matching with the full tileset:

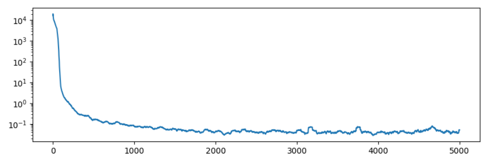

Figure 5: Flow Matching Loss with full tile set

This setting is with the from center sampling mechanism which is considered the default for our project. There isn't much interesting happening here other than the GFlowNet learning something over time. Which is better shown with the reward plot:

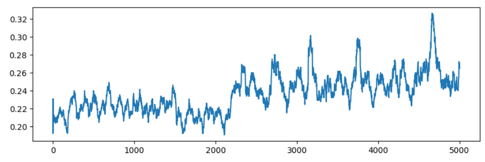

Figure 6: Flow Matching Reward with full tile set

Here we see the average reward around 0.22 till around 2000 steps after which the rewards rise to 0.25, which is a marginal difference, but definitive nonetheless.

A sample playable level generated by this set up is as follows:

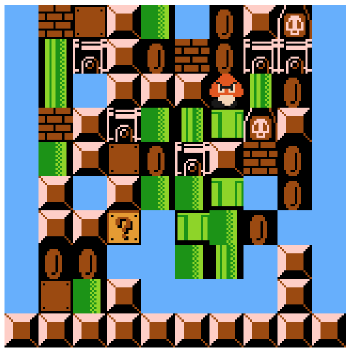

Figure 7: Sample Level for Flow Matching with full tile set

The learning is more apparent with the limited tile set, where the loss function is similar to the previously seen plot, but the increase observed reward is more apparent:

A generated level in this setup is as follows:

Here we can obviously see that the observed reward goes from an average of 0.5 to an averge of 0.9 around the end of the training process.

The other divergence in observed loss values happens with the from bottom left sampling mechanism, from left sampling mechanism, the Stable Flow Matching objecting and the Trajectory Balance objective. With the Stable Flow Matching objective with from

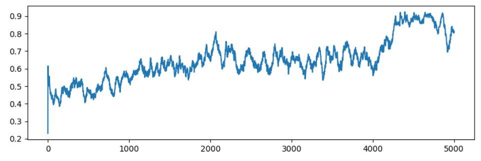

Figure 8: Flow Matching rewards with limited tile set

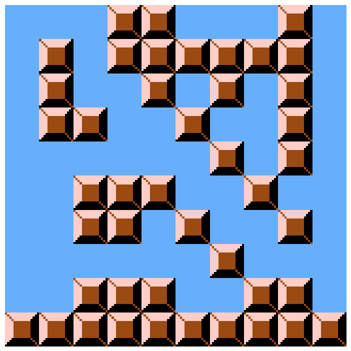

Figure 9: Sample Level for Flow Matching with limited tile set

bottom left sampling mechanism with the limited tileset we see that the loss we observe is as we expect with a "correctly" learning GFlowNet.

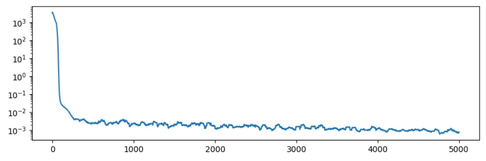

Figure 10: Stable Flow Matching from bottom left sampling losses with Limited tile set

But this is one of the cases where the setup is unable to learn how to obtain higher rewards even the limited tile set. With the full tile set we can make the argument that due to the large action space the agent does not see enough high reward states in the duration of training to learn how to generate high reward trajectories, but with this setup, with the limited tile set it generates some high reward levels and then eventually generates lower reward levels.

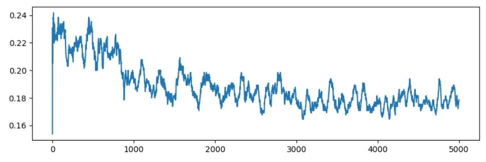

Figure 11: Stable Flow Matching from bottom left sampling rewards with Limited tile set

This set up was unable to generate any playable levels.

All setups with the Trajectory Balance objective had a very different loss function as it is optimizing the agent by a different constraint and assumptions as stated previously. On the log scale it is almost linear, which means it shows approximately exponential on the linear scale. And the Trajectory Balance has more variance as apparent with it showing variance even with a moving average visualisation.

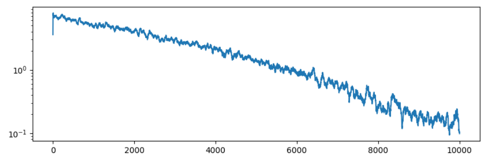

Figure 12: Trajectory Balance losses with full tile set

The reward also shows this variance where the full tile set does still generate higher reward levels, but it shows high variance with the moving average, similarly to the loss plot. This setup generates 5 fully playable levels after 5000 total full level generations. This is good as it can go from generating completely unplayable levels to generating levels which can generate somewhat playable levels.

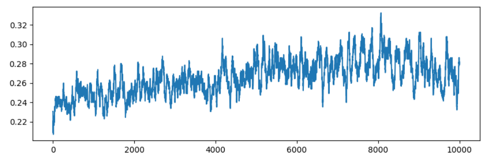

Figure 13: Trajectory Balance rewards with full tile set

With the Trajectory Balance we were able to generate 13 playable levels with the defined playability metric, one of which is as follows:

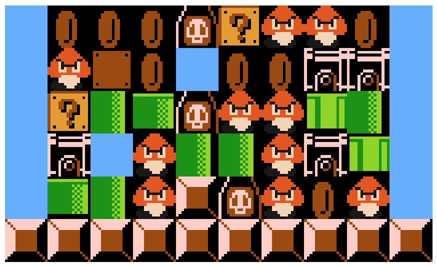

Figure 14: Sample Level for Trajectory Balance with full tile set

The limited tile set with Trajectory Balance learns faster than the full tile set, as we would expect.

It also learns to generate better levels earlier in the training process than setups with the Flow Matching objective.

A generated level with this setup is as follows:

The Flow Matching with Feature Map setup and the Stable Flow Matching with Feature Map setup are unable to learn to generate

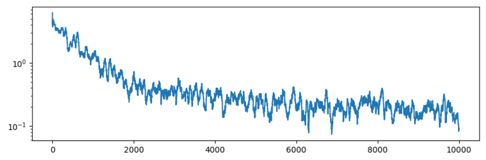

Figure 15: Trajectory Balance losses with limited tile set

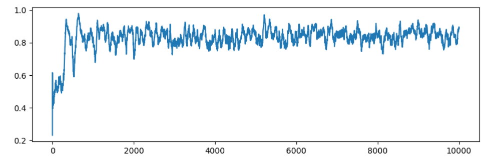

Figure 16: Trajectory Balance rewards with limited tile set

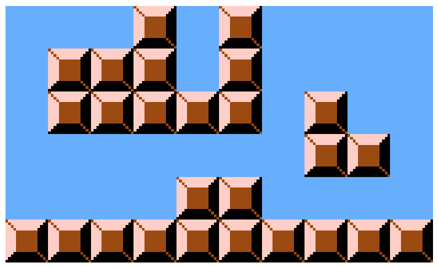

Figure 17: Sample Level for Trajectory Balance with limited tile set

any reasonable levels with high rewards as seen in the following plots.

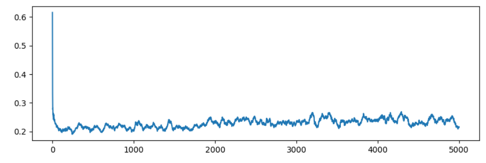

Figure 18: Flow Matching with Feature Maps rewards with full tile set

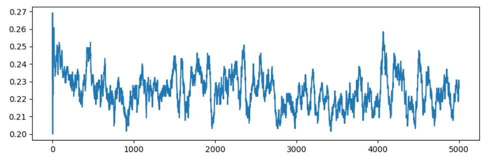

Figure 19: Stable Flow Matching with Feature Maps rewards with full tile set

Other rewards of notable behaviour are as follows:

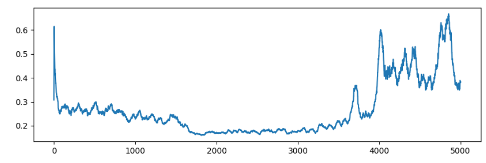

Figure 20: Flow Matching from bottom left sampling rewards with limited tile set

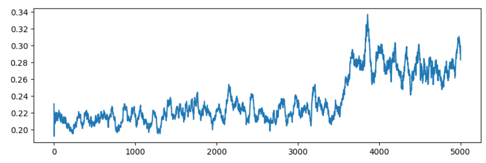

Figure 21: Stable Flow Matching rewards with full tile set

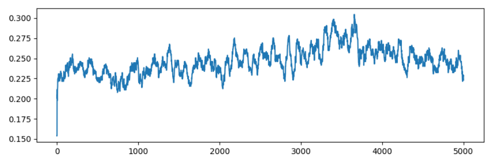

Figure 22: Stable Flow Matching from left sampling rewards with full tile set

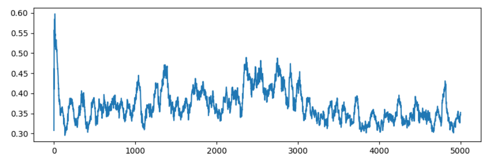

Figure 23: Stable Flow Matching from left sampling rewards with limited tile set

8 DISCUSSION AND FUTURE DIRECTIONS

The results presented in this paper demonstrate the feasibility of using Generative Flow Networks (GFlowNets) for Mario level generation. The model successfully learns to generate playable levels, as evidenced by the increasing reward observed during training.

One potential direction is to incorporate a human-in-the-loop approach where human feedback can be used to guide the training process. This could involve allowing humans to curate the generated levels and provide feedback on their quality and playability. The GFlowNet could then be retrained using this feedback, potentially leading to the generation of even better levels.

Another interesting direction is to explore different GFlowNet architectures and training objectives. The current work utilizes a basic CNN architecture and a limited set of objective functions. Experimenting with more sophisticated architectures and objectives specifically designed for level generation tasks could yield further improvements.

This can be done a few other training models ways, namely:

- Vectorize the 2D level and input it to either an RNN or a Transformer.
- Use a Box Transformer or a Visual Transformer with the same sampling mechanism.

Finally, it is important to consider the creative aspect of level design. While GFlowNets can generate playable levels, they may lack the creativity and surprise elements often found in human-designed levels. Future work could explore ways to incorporate these creative elements into the generation process, perhaps by leveraging a multi-pass sampling mechanism or a better reward signal.

We can also attempt to perform the following modifications to the current method:

- A Flood Fill like sampling mechanism would be more deterministic
- A simpler sampling mechanism, i.e. sampling next tiles from bottom to top, and left to right
- A learnable sampling mechanism for the next tile, as we are only learning how to sample what to place in the next tile, but not where to sample the next tile
- A simpler game, a maze building task
- Pre-Training a GFlowNet by Behaviour Cloning

9 CONCLUSION

This project presented a novel approach to Mario level generation using Generative Flow Networks (GFlowNets). The model demonstrates the ability to generate playable levels, opening doors for further exploration in this domain. Future work can focus on incorporating human feedback, experimenting with different architectures and objectives, and exploring methods to introduce creativity into the generated levels and different sampling mechanisms. These advancements hold promise for creating even more engaging and diverse Mario level generation experiences.

REFERENCES

[1] Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *CoRR*, abs/2106.04399, 2021.

[2] Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J. Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. *Journal of Machine Learning Research*, 24(210):1–55, 2023.

[3] Leo Maxime Brunswic, Yinchuan Li, Yushun Xu, Shangling Jui, and Lizhuang Ma. A theory of non-acyclic generative flow networks, 2023.

[4] Moksh Jain, Sharath Chandra Raparthy, Alex Hernández-García, Jarrid Rector-Brooks, Yoshua Bengio, Santiago Miret, and Emmanuel Bengio. Multi-objective GFlownets, 2023.

[5] Nikolay Malkin, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. Trajectory balance: Improved credit assignment in GFlownets. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[6] Nikolay Malkin, Salem Lahlou, Tristan Deleu, Xu Ji, Edward J Hu, Katie E Everett, Dinghuai Zhang, and Yoshua Bengio. GFlownets and variational inference. In *The Eleventh International Conference on Learning Representations*, 2023.

[7] Ishita Mediratta, Minqi Jiang, Jack Parker-Holder, Michael Dennis, Eugene Vinitsky, and Tim Rocktäschel. Stabilizing unsupervised environment design with a learned adversary. In Sarath Chandar, Razvan Pascanu, Hanie Sedghi, and Doina Precup, editors, *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pages 270–291. PMLR, 22–25 Aug 2023.

[8] Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontañón. The vglc: The video game level corpus, 2016.

[9] David W Zhang, Corrado Rainone, Markus Peschl, and Roberto Bondesan. Robust scheduling with GFlownets. In *The Eleventh International Conference on Learning Representations*, 2023.

[10] Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial problems with GFlownets. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.