Adaptive Exploration for Deep Reinforcement Learning

Geoffrey H. Harrison
Department of Computer Science
University of Toronto
Toronto, ON M5S 1A1, Canada
ghharrison@cs.toronto.edu

Sarah Hindawi
Department of Computer Science
University of Toronto
Toronto, ON M5C G01, Canada
shindawi@cs.toronto.edu

Abstract

Good exploration is important to the sample efficiency of reinforcement learning methods, as well as the strength of the resulting learned policy. Exploring meaningful states is especially important in problems with sparse rewards. Exploration is sometimes guided using an intrinsic reward, which measures the novelty of experienced states and provides signal even in the absence of a frequently updating extrinsic reward, but the strength of learned policies using intrinsic-extrinsic score decompositions depends strongly on factors such as the emphasis of exploration over exploitation and the discount factor used for each reward head. We propose a method to adaptively adjust these factors to improve sample efficiency and exploration quality, validating the performance of the learned agent in an OpenAI Gym environment.

1 Introduction

The reinforcement learning problem seeks to maximize *expected trajectory return*, or the expected cumulative reward gathered by an agent as it takes actions and receives rewards from the environment [1]. To determine what sequences of actions are truly optimal, and what states must be encountered at which to take those actions, requires adequate exploration of the environment state-action space. Exploring this space in a meaningful, informative way remains a persistent challenge in the domain of reinforcement learning; finding an optimal policy is only theoretically guaranteed once the number of visits to each state-action pair approaches infinity [2]. Ensuring that all such pairs may be visited, while still acquiring a learned agent that maximizes trajectory returns, is in a nutshell the challenge of exploration.

The exploration process is additionally influenced by a discount factor (often denoted γ), a value typically between 0 and 1 that decides how strongly to weight future as opposed to instantaneous rewards when estimating the value function [1]. Choosing a good value for γ is often an important factor in the success or failure of the reinforcement learning process; a γ of 0, for instance, makes the agent act in a purely greedy fashion, only selecting actions to maximize the instantaneous reward at each step, whereas a high value of $\gamma = 0.999$ might be used in difficult, sparse-reward tasks where long-term planning is critical. There are cases to be made for low or high discount factors; small factors have been suggested to generalize better [3], but large factors handle reward sparsity and long-term credit assignment better [4]. Manually tuning an algorithm, however, to find the ideal discount factor for a particular task can be a slow, time-consuming process.

We propose a modification to Never Give Up (NGU), a deep reinforcement learning algorithm that has shown success on the Atari 2600 suite of video games (a common reinforcement learning benchmark) [5], wherein we adaptively update the discount factor as training proceeds in order to determine the best value of γ automatically, adjusting the rate at which the agent explores the state-action space

Preprint. Under review.

accordingly. We demonstrate the method on Pymaze [6], a simple yet informative reinforcement learning environment, performing an ablation study to validate the improvement brought on by our method.

2 Related Work

Reinforcement learning algorithms rely on a combination of *exploration* (exploring the state-action space) and *exploitation* (taking actions to maximize expected return) in order to achieve good results [1]. However, standard methods of exploration, such as random actions under an ϵ-greedy policy, struggle to explore the state space in a meaningful way and often encounter the same handful of states repeatedly. The issue is compounded in sparse-reward games, where the reward signal does not often update and is of limited help in exploration, or hard-exploration games where achieving non-trivial returns requires long and complex sequences of precise actions. A common solution in recent literature is the *intrinsic reward*, wherein the “extrinsic" reward provided by the environment (e.g. game score) is augmented with an “exploration bonus" that rewards the agent for encountering novel states [2]. The Never Give Up (NGU) reinforcement learning algorithm seeks to overcome the problem of never returning to well-trodden states by introducing an *episodic* novelty score alongside the lifelong intrinsic reward, such that even a state that has been seen many times in prior episodes may still be explored so that any downstream opportunities may still be discovered [5].

Training is informed by the choice of discount factor and exploration rate. However, it is challenging to know what the optimal discount factor for a particular problem is *a priori*, and it is troublesome to update it manually only after many episodes of training. In NGU, this dilemma is addressed by jointly learning a family of policies, some of which are more exploration-oriented and some more exploitation-oriented [5]. The policies are trained using a Universal Value Function Approximator framework with some sharing between policies, in essence training all such policies simultaneously. However, Agent57, a successor to NGU, modifies this framework by using a multi-armed bandit “metacontroller” to select at the start of each episode the policy that is assessed to make the most progress [7].

An alternative to the “policy family” approach is to explore adaptively, using a single policy but changing its hyperparameters over time in response to developments in training. This is often motivated by a desire to avoid the problem of value function overestimation, which tends to bias Q-learning methods towards out-of-distribution actions with erroneously high estimated values, while still maintaining a high enough discount factor to encourage long-term reward-seeking from the agent. A simple solution is to use two value heads and simply select at each step whichever one produces lower value estimates [8]. Another common approach is to adjust the learning rate over time to a fixed maximum [9][10]. Kim et al. introduce a method for adjusting the discount factor adaptively in response to over- or under-estimation of the advantage function, gradually tightening a bound on the optimal discount factor [11]. We use this method as the basis for our modified NGU, as it does not fix an arbitrary initial value for discount factor or adjust it over time according to a deterministic rule, but can instead find the *optimal* discount factor for the task at hand without overshooting.

3 Method

We begin by building our method atop an open-source implementation of Never Give Up [12]. However, this open-source implementation only includes the episodic novelty module, so we additionally must re-implement the Random Network Distillation module before we achieve a true reconstruction of NGU.

Our method builds upon the notions of intrinsic reward introduced by prior work, specifically the *episodic* novelty reward developed in NGU and the *lifelong* novelty reward from RND. As in NGU, these two notions of novelty are combined by computing episodic intrinsic reward, which stores experienced states in an episodic memory buffer and compares the current state to the contents of the buffer at each step, and scaling it by a life-long curiosity factor which is computed in terms of error in predicting the output of a randomly initialized neural network that takes the current state as an input [2].

The NGU agent is trained using transformed Retrace double-Q loss applied to Reccurent Replay Distributed DQN (R2D2) [5][13]. As is common in reinforcement learning algorithms, computing the temporal difference error involves the use of a discount factor, denoted γ, which affects the relative impact on the agent's learning of instantaneous reward versus future reward, the latter as embodied by an estimated state-value (Q) function. However, in R2D2 and similar RL algorithms, this discount remains fixed. We are interested in the possibility of making this discount factor *adaptive,* rather than initializing it to a fixed arbitrary value or attempting to find an optimal value through grid search. NGU and Agent57 make their RL agents adaptive by considering a wide range of possible tuples of exploration rate β and discount factor γ; NGU attempts to learn the optimal value function $Q(x, a, \beta_i)$ with respect to all such β simultaneously, in effect training the family of policies according to a uniform selection from the set of (β_i, γ_i) tuples [5], whereas Agent57 instead selects which policy to train at the beginning of each episode according to a multi-armed bandit algorithm [7].

We propose instead to adapt a technique devised by Kim et al. to train a single policy while scaling the discount factor according to developments in the training process. Kim et al. train by using *two* discount factors γ_1 and γ_2 initialized to 0.5 and 0.99 respectively, with γ_1 used for action selection and γ_2 an upper bound on eventual values of γ_1 [11]. The advantage function is expressed as $A_t = \frac{1}{M}\sum^{M}(Q(s_t, a_t) - V(s_t))$, with $A_t(\gamma)$ denoting A_t computed with discount factor γ. and compute $A_t(\gamma_i)$ for $i = 1, 2$. Comparing them, the discount factors are updated according to

$$(\gamma_1, \gamma_2) = \begin{cases} \gamma_1 \leftarrow \gamma_1 + c \cdot (1/\sigma_t) & \text{if } A_t(\gamma_1) < A_t(\gamma_2) \\ \gamma_2 \leftarrow \gamma_2 - c \cdot (1/\sigma_t) & \text{if } A_t(\gamma_1) > A_t(\gamma_2) \\ (\gamma_1, \gamma_2) & \text{otherwise} \end{cases} \tag{1}$$

where

$$\sigma_t = 1 + \frac{|\delta_{t-1} - \delta_t|}{\delta_t} \tag{2}$$

$$\delta_t = (r_t + \gamma(V(s_{t+1}) - V(s_t))^2 \tag{3}$$

σ_t is a correction function that depends on the difference at each step in δ_t, δ_t is the squared TD error of the preceding timestep, and c is a hyperparameter controlling the step size. If the higher value of γ results in overestimation of the advantage, the upper bound on the discount factor, is reduced. Otherwise, it is assumed safe to increase γ_1. δ_t is defined so as to be large at the beginning of training and decrease over time as the value function becomes more reliable. In time, the bound tightens, and the agent is able to automatically determine the best value of the discount function.

The advantage function is difficult to compute exactly, but fortunately NGU provides a convenient way to compute the advantage function using the two values of γ. As it is built on R2D2, which as a Q-learning method is trained using temporal difference error, and as advantage is simply the expectation of the temporal difference error, we can use the TD error of each sample from the memory buffer as an approximation of advantage, and adjust γ_1 and γ_2 accordingly.

We combine Kim's adaptive discount factor with the intrinsic reward used by Never Give Up, favoring a simpler implementation that avoids the "family" of policies used by NGU and Agent57 and dispensing with the need for a multi-armed bandit metacontroller to select the optimal hyperparameters to train at each step. Additionally, inspired by the ability of Agent57 to adapt the exploration rate β over time in addition to γ, we initialize β to a starter value β_{max} and scale the exploration rate in keeping with updates to γ_1:

$$\beta_{t+1} = \begin{cases} \beta_t - \frac{\Delta\gamma \cdot (\beta_{max} - \beta_{min})}{\gamma_{max} - \gamma_{min}} \text{ if } A_t(\gamma_1) < A_t(\gamma_2) \\ \beta_t \text{ otherwise} \end{cases} \tag{4}$$

$$\Delta\gamma = c \cdot (1/\sigma_t) \tag{5}$$

This means that, as γ_1 is the discount factor that the policy actually uses at a given moment, the exploration rate remains at a high value as long as the discount factor does not increase, but should γ_1 be increased to better approximate the true value function, then the exploration rate will decrease by an amount proportional to the change in discount factor. We do this as one would expect policies with high β and low γ to make progress early in training, whereas later on a low β and higher γ becomes preferable. This is the same intuition underlying the use of the metacontroller in Agent57 [7].

4 Evaluation and Results

We use the OpenAI Gym Pymaze environment in which to test our reinforcement learning agent. Specifically, we use the “Maze Plus” environments, in which the mazes also have open spaces as well as portals that teleport the player to a matching location when used [6]. This ensures that there are multiple solutions to the maze (in contrast to ordinary mazes where there is only one “correct" solution that does not involve retracing one’s steps.) This ensures that our agent, which is discouraged by the episodic curiosity module from returning to earlier states, must find the *best* path to the goal rather than *any* path to obtain the maximal reward. We test our agent in the 10x10, 20x20, and 30x30 Random Maze Plus environments, fixing the seed on the random maze generator to ensure that all agents are evaluated on the same problem. We test our results over the course of 2500 training episodes.

We observe that, while our method initially struggles to make as much progress as standard NGU earlier in training, with a greater number of episodes, our agent is able to surpass NGU’s performance and achieve an average reward of close to 10, the maximum possible value in Pymaze environments. (See **Appendix B** for more detailed results.) Ablating the agent by iteratively removing the lifelong curiosity (RND) module and the episodic novelty module, we observe that the performance degrades steadily; the NGU agent (with RND but without adaptive discount factors) quickly learns to gain an initial reward, but in both 10×10 and 20×20 cases suffers from “catastrophic forgetting,” seeing average returns decrease without ever reaching the maximum value. (All three agents failed to achieve any positive reward in the 30×30 case within 2500 training episodes.) Without RND, the agent learns yet slower and suffers an equally drastic loss in return as NGU, and again fails to achieve the maximum return possible. These results suggest that our method is a viable improvement over NGU, achieving higher trajectory returns at the potential cost of some sample efficiency as the agent “tunes” itself to find the optimal discount factor.

5 Conclusion

We introduced a method to adaptively adjust the discount factor and exploration rate in a reinforcement learning algorithm based on Never Give Up (NGU). Our method combines the episodic novelty and lifelong curiosity modules from NGU with an adaptive update rule for the discount factor and exploration rate. The results of our experiments demonstrate that our proposed algorithm is able to outperform the baseline NGU algorithm in terms of exploration quality and sample efficiency, especially in the later stages of training.

While our method does not achieve positive trajectory returns as quickly as the baseline NGU, it quickly surpasses it as training progresses, achieving near-optimal performance in the tested Pymaze environments. These results also suggest that our method is a viable improvement over the NGU framework, as it allows the agent to automatically adapt the discount factor and exploration rate during training, leading to more efficient exploration and learning. However, a proper baseline is still needed to compare our modified algorithm to the state-of-the-art; testing on Atari games such as *Q-Bert* or *Montezuma’s Revenge*, for instance, would allow us to directly compare against the benchmarks set by contemporary RL algorithms such as Agent57, Ape-X, and the like.

An area of future investigation to consider is an adaptive discount factor that is “non-monotonic”; while our method assumes that the given task has a single optimal discount factor (and exploration rate) that can be found during training, it relies on the assumption that a greater discount factor is strictly better as training progresses. Conceptually, if training is found to lead to “dead end” periods in which no further improvement on return can be found, it may be best for the agent to encourage more exploration rather than exploitation even late in training, especially in large state-action spaces that are highly time-consuming to fully explore. Establishing conditions under which the discount factor (and not just its upper bound, as for γ_2) may be *decreased,* and the exploration rate increased, could help to break such a stalemate.

Additional future work could focus on investigating the interplay between intrinsic and extrinsic rewards, as well as other methods to encourage exploration, such as hierarchical reinforcement learning or information-theoretic approaches. In addition, the generalizability of the learned policy across different maze configurations and the robustness of the agent to changes in the environment, such as the addition of obstacles or non-stationary rewards, could be explored.

References

[1] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[2] Burda, Y., Edwards, H., Storkey, A., & Klimov, O. (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894.*

[3] Van Seijen, H., Fatemi, M., Tavakoli, A. (2019). Using a logarithmic mapping to enable lower discount factors in reinforcement learning. Advances in Neural Information Processing Systems, 32.

[4] Amit, R., Meir, R., & Ciosek, K. (2020, November). Discount factor as a regularizer in reinforcement learning. In International conference on machine learning (pp. 269-278). PMLR.

[5] Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Olivier, L., Arjovsky, M. & Blundell, C. (2020). Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038.*

[6] Chan, M. (2020). gym-maze. GitHub repository, https://github.com/MattChanTK/gym-maze

[7] Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., & Blundell, C. (2020, November). Agent57: Outperforming the atari human benchmark. *In International conference on machine learning* (pp. 507-517). PMLR.

[8] Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence (Vol. 30, No. 1).

[9] François-Lavet, V., Fonteneau, R., & Ernst, D. (2015). How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies. *In NIPS 2015 Workshop on Deep Reinforcement Learning.*

[10] Fedus, W., Gelada, C., Bengio, Y., Bellemare, M. G., & Larochelle, H. (2019). Hyperbolic discounting and learning over multiple horizons. arXiv preprint arXiv:1902.06865.

[11] Kim, M., Kim, J. S., Choi, M. S., & Park, J. H. (2022). Adaptive Discount Factor for Deep Reinforcement Learning in Continuing Tasks with Uncertainty. *Sensors*, 22(19), 7266.

[12] Le, V. (2021). PyTorch implementation of Never Give Up: Learning Directed Exploration Strategies. GitHub repository, https://github.com/Coac/never-give-up

[13] Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., Dabney, W. (2019, May). Recurrent experience replay in distributed reinforcement learning. In International conference on learning representations.

A Evaluation setup

The setup we use is the same as that used in Never Give Up [5], which itself is the same as that used by R2D2 [13].

Learner

- Sample from the replay buffer a batch of augmented (scaled sum of intrinsic/extrinsic) rewards r_t, intrinsic rewards r_t^i, state observations x, actions a, and discount factor $\gamma_1^{(t)}$.
- Perform Q-learning on (r_t, a, x) as in RND using Retrace.
- Use the batch to train the Random Network Distillation (RND) network as well.

Actor

- Obtain state x_t, intrinsic and extrinsic rewards r_t^e, r_{t-1}^i, and discount γ_i.
- Compute the forward pass of R2D2 to obtain the selected action a.
- Using RND and the embedding network as described in NGU, obtain the intrinsic reward r^i and compute augmented reward r using the current values of γ_1 and γ_2.
- Using the TD error, update $\gamma_1 \leftarrow \gamma_1 + c \cdot (1/\sigma_t)$ or $\gamma_2 \leftarrow \gamma_2 - c \cdot (1/\sigma_t)$ according to the rule described in Section 3.
- Insert state, action, augmented reward, intrinsic reward, and the current γ_1 into the replay buffer.
- Step in environment using action a.

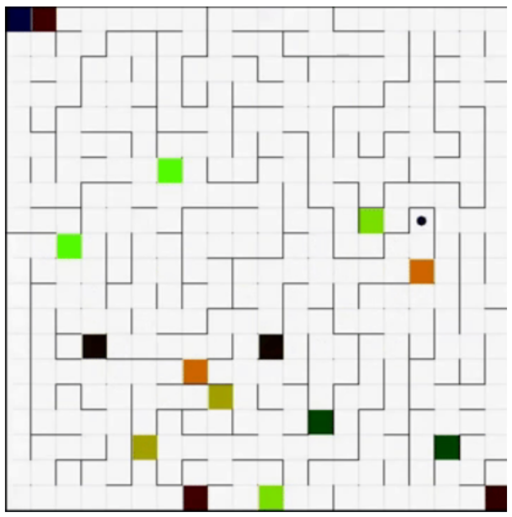

Figure 1: An example of a 20 $\times$ 20 PyMaze environment. Same-colored squares are "portals" that the player can instantaneously move between. The maze also has more valid solutions than a typical maze.

B Detailed Results

We include charts of mean reward and discount factor incurred by our modified NGU agent, the unmodified NGU agent (labelled as "No Adaptive Discount"), and an NGU agent with only the episodic curiosity module ("No AD or RND"). The agents were trained on PyMaze "Random Maze Plus" environments with a random seed initialized to a fixed value to ensure consistency between runs. Run data was recorded using Weights & Biases.

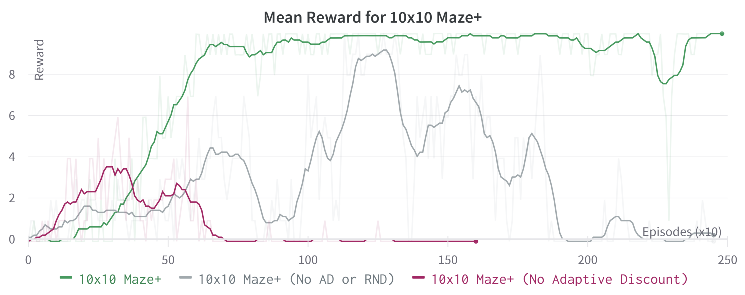

Figure 2: Mean reward on the 10x10 Maze+ environment. Our method is compared over 2500 training episodes with the base NGU algorithm with no adaptive discount factor (in red) and the algorithm with no adaptive discount factor or RND (in blue.) Reward is shown as a running average over the preceding 100 episodes.

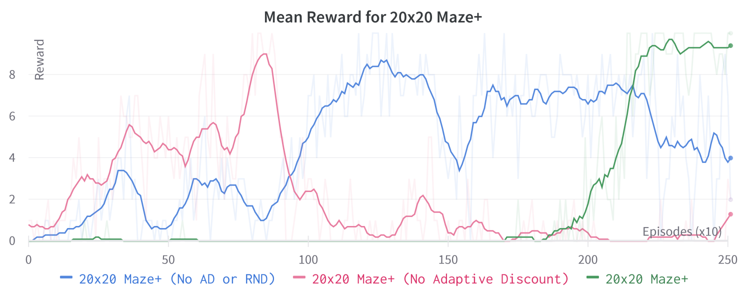

Figure 3: Mean reward on the 20x20 Maze+ environment. Our method is compared over 2500 training episodes with the base NGU algorithm with no adaptive discount factor (in red) and the algorithm with no adaptive discount factor or RND (in blue.) Reward is shown as a running average over the preceding 100 episodes.

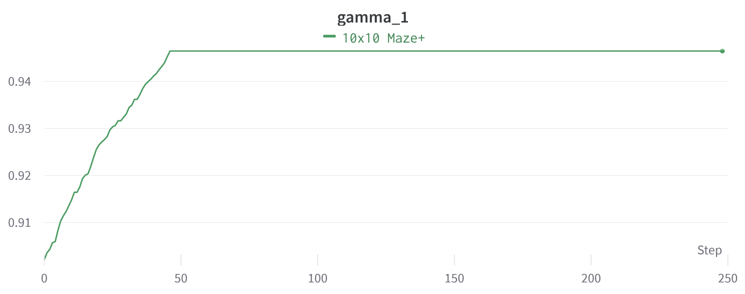

Figure 4: Value of γ_1 over time for the 10x10 maze with adaptive discount factor.

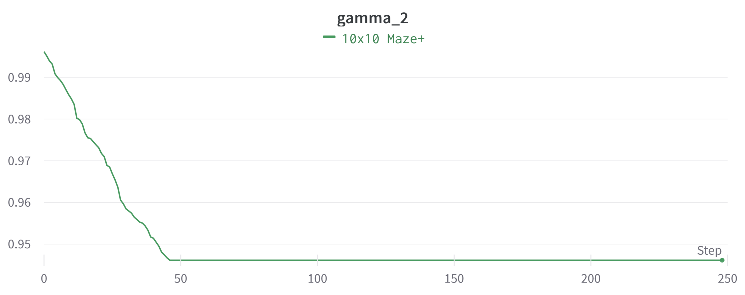

Figure 5: Value of γ_2 over time for the 10x10 maze with adaptive discount factor.

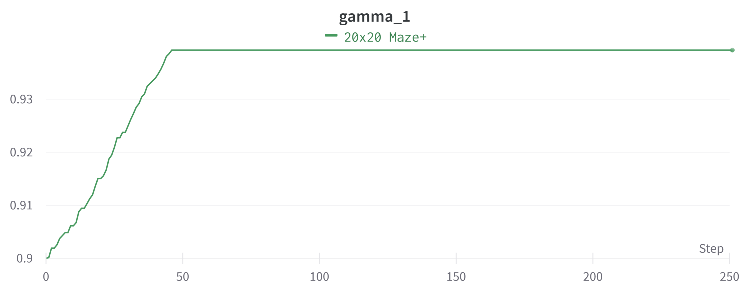

Figure 6: Value of γ_1 over time for the 20x20 maze with adaptive discount factor.

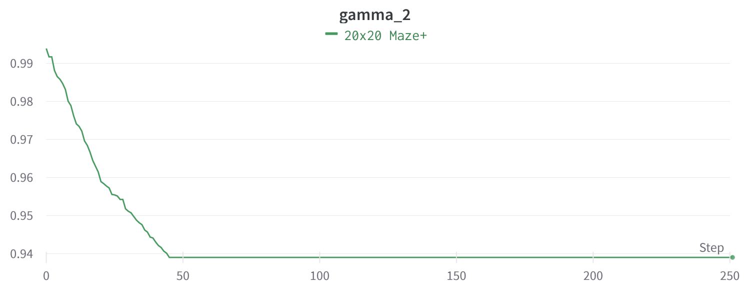

Figure 7: Value of γ_2 over time for the 20x20 maze with adaptive discount factor.

C Hyperparameters

Hyperparameter	Value
γ_1 (initial)	0.90
γ_2 (initial)	0.999
β_{max}	0.3
β_{min}	0
c	1e-3
Replay memory size	1000
Optimizer	Adam
learning rate	0.001
ϵ	0.1
ϵ decay rate	1e-5
Random seed (maze generation)	9001
Random seed (action selection)	42
Number of training episodes	2500
Epsilon decay rate	0.00001
Hidden state size	16
Local minibatch size	8

D Group Contributions

Geoffrey H. Harrison was responsible for the idea of combining NGU with an adaptive discount factor, as well as implementing the new algorithm, performing evaluation runs, and contributing to the literature review.

Sarah Hindawi was responsible for helping to develop the idea of the new algorithm, as well as writing the final report and analysis and contributing to the literature review.