MAXIMUM REWARD FORMULATION IN REINFORCEMENT LEARNING

Sai Krishna Gottipati[1], Yashaswi Pathak*[1,2], Rohan Nuttall[3], Sahir[3], Raviteja Chunduru[6], Ahmed Touati[6], Sriram Ganapathi Subramanian[5], Matthew E. Taylor[3,4], and Sarath Chandar[6,7,8]

[1]99andBeyond
[2]International Institute of Information Technology, Hyderabad
[3]Department of Computing Science, University of Alberta
[4]Alberta Machine Intelligence institute
[5]University of Waterloo
[6]Mila - Quebec AI Institute
[7]Canada CIFAR AI Chair
[8]Ecole Polytechnique Montréal

ABSTRACT

Reinforcement learning (RL) algorithms typically deal with maximizing the expected cumulative return (discounted or undiscounted, finite or infinite horizon). However, several crucial applications in the real world, such as drug discovery, do not fit within this framework because an RL agent only needs to identify states (molecules) that achieve the highest reward within a trajectory and does not need to optimize for the expected cumulative return. In this work, we formulate an objective function to maximize the expected maximum reward along a trajectory, propose a novel functional form of the Bellman equation, introduce the corresponding Bellman operators, and provide a proof of convergence. Using this formulation, we achieve state-of-the-art results on the task of synthesizable molecule generation that mimics a real-world drug discovery pipeline.

1 INTRODUCTION

Reinforcement learning (RL) algorithms typically try to maximize the cumulative finite horizon undiscounted return, $R(\tau) = \sum_{t=0}^{T} r_t$, or the infinite horizon discounted return, $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$. r_t is the reward obtained at time step t, γ is the discount factor in the range $[0, 1)$, and τ is the agent's trajectory. τ consists of actions (a) sampled from the policy ($\pi(\cdot \mid s)$) and states (s') sampled from the probability transition function $P(s'|s, a)$ of the underlying Markov Decision Process (MDP).

The action-value function $Q^\pi(s, a)$ for a policy π is given by

$$Q^\pi(s, a) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau)|(s_0, a_0) = (s, a)]$$

The corresponding Bellman equation for $Q^\pi(s, a)$ with the expected return defined as $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$ is

$$Q^\pi(s_t, a_t) = \mathop{\mathbb{E}}_{\substack{s_{t+1} \sim P(\cdot|s_t, a_t) \\ a_{t+1} \sim \pi(\cdot|s_{t+1})}} [r(s_t, a_t) + \gamma Q^\pi (s_{t+1}, a_{t+1})]$$

This Bellman equation has formed the foundation of RL. However, we argue that optimizing for only the maximum reward achieved in an episode is also an important goal. Reformulating the RL problem to achieve the largest reward in an episode is the focus of this paper, along with empirical demonstrations in one toy and one real-world domain.

In the *de novo* drug design pipeline, molecule generation tries to maximize a given reward function. Existing methods either optimize for the expected cumulative return, or for the reward at the end of

*Work done during internship at 99andBeyond.

the episode, and thus fail to optimize for the very high reward molecules that may be encountered in the middle of an episode. This limits the potential of several of these reinforcement learning based drug design algorithms. We overcome this limitation by proposing a novel functional formulation of the Bellman equation:

$$Q^{\pi}_{\max}(s_t, a_t) = \mathbb{E}_{\substack{s_{t+1}\sim P(\cdot|s_t,a_t) \\ a_{t+1}\sim\pi(\cdot|s_{t+1})}} \left[\max\left(r(s_t, a_t), \gamma Q^{\pi}_{\max}(s_{t+1}, a_{t+1})\right)\right] \tag{1}$$

Other use cases of this formulation (i.e, situations where the single best reward found, rather than the total rewards, are important) are - symbolic regression (Petersen (2020), Udrescu & Tegmark (2020)) which is interested in finding the single best model, active localization (Chaplot et al. (2018)) must find the robot's one most likely pose, green chemistry (Koch et al. (2019)) wants to identify the one best product formulation, and other domains that use RL for generative purposes.

This paper's contributions are to:

- Propose a novel functional form of the Bellman equation, called max-Bellman, to optimize for the maximum reward in an episode.
- Introduce the corresponding evaluation and optimality operators, and prove the convergence of Q-learning with the max-Bellman formulation.
- Test on a toy environment and draw further insights with a comparison between Q-learning and Q-learning with our max-Bellman formulation.
- Use this max-Bellman formulation to generate synthesizable molecules in an environment that mimics the real drug discovery pipeline, and demonstrate significant improvements over the existing state-of-the-art methods.

2 Related work

This section briefly introduces fundamental RL concepts and the paper's main application domain.

2.1 Reinforcement Learning

Bellman's dynamic programming paper (Bellman, 1954) introduced the notions of optimality and convergence of functional equations. This has been applied in many domains, from control theory to economics. The concept of an MDP was proposed in the book Dynamic Programming and Markov Processes (Howard, 1960) (although some variants of this formulation already existed in the 1950s). These two concepts of Bellman equation and MDP are the foundations of modern RL. Q-learning was formally introduced in (Watkins & Dayan, 1992) and different convergence guarantees were further developed in (Jaakkola et al., 1993) and (Szepesvári, 1997). Q-learning convergence to the optimal Q-value ($Q^{\star}$) has been proved under several important assumptions. One fundamental assumption is that the environment has finite (and discrete) state and action spaces and each of the states and actions can be visited infinitely often. The learning rate assumption is the second important assumption, where the sum of learning rates over infinite episodes is assumed to go to infinity in the limit, whereas the sum of squares of the learning rates are assumed to be a finite value (Tsitsiklis, 1994; Kamihigashi & Le Van, 2015). Under similar sets of assumptions, the on-policy version of Q-learning, known as Sarsa, has also been proven to converge to the optimal Q-value in the limit (Singh et al., 2000).

Recently, RL algorithms have seen large empirical successes as neural networks started being used as function approximators (Mnih et al., 2016). Tabular methods cannot be applied to large state and action spaces as these methods are linear in the state space and polynomial in the action spaces in both time and memory. Deep reinforcement learning (DRL) methods on the other hand, can approximate the Q-function or the policy using neural networks, parameterized by the weights of the corresponding neural networks. In this case, RL algorithms easily generalize across states, which improves the learning speed (time complexity) and sample efficiency of the algorithm. Some popular Deep RL algorithms include DQN (Mnih et al., 2015), PPO (Schulman et al., 2017), A2C (Mnih et al., 2016), SAC (Haarnoja et al., 2018), TD3 (Fujimoto et al., 2018b), etc.

2.2 DE NOVO DRUG DESIGN

De novo drug design is a well-studied problem and has been tackled by several methods, including evolutionary algorithms (Brown et al. (2004); Jensen (2019); Ahn et al. (2020)), generative models (Simonovsky & Komodakis (2018); Gómez-Bombarelli et al. (2018); Winter et al. (2019); Jin et al. (2018); Popova et al. (2018); Griffiths & Hernández-Lobato (2020); Olivecrona et al. (2017)), and reinforcement learning based approaches (You et al. (2018a); Zhou et al. (2018)). While the effectiveness of the generated molecules using these approaches has been demonstrated on standard benchmarks such as Guacamol (Brown et al. (2019)), the issue of synthesizability remains a problem. While all the above approaches generate molecules that optimize a given reward function, they do not account for whether the molecules can actually be effectively synthesized, an important practical consideration. Gao & Coley (2020) further highlighted this issue of synthesizability by using a synthesis planning program to quantify how often the molecules generated using these existing approaches can be readily synthesized. To attempt to solve this issue, Bradshaw et al. (2019) used a variational auto-encoders based approach to optimize the reward function with single-step reactions. Korovina et al. (2019) employed a random selection of reactants and reaction conditions at every time step of a multi-step process. PGFS (policy gradient for forward synthesis) from Gottipati et al. (2020) generates molecules via multi-step chemical synthesis and simultaneously optimized for the given reward function. PGFS leveraged TD3 algorithm (Fujimoto et al. (2018a)), and like existing approaches, optimizes for the usual objective of total expected discounted return.

3 METHOD

This section formally defines the novel objective function that optimizes the maximum reward in an episode, defines the corresponding novel functional form of Bellman equation, and proves its convergence properties.

Since the objective is to optimize for maximum reward achieved in a trajectory, the overall expected return can be re-formulated as:

$$R(\tau) = \max_{t \geq 0} \gamma^t r_t$$

where τ is the trajectory in which the actions a are sampled using policy network π and next states s' are sampled from the MDP's probability transition function $P(s'|s, a)$. $\gamma \in [0, 1)$ allows the agent to prefer rewards sooner rather than later.

The action-value function for a stationary policy π, denoted $Q^{\pi}_{\max} : \mathcal{S} \times A \to \mathbb{R}$, represents the expected maximum of discounted rewards along the trajectories induced by the policy in the MDP. The new functional form of the Bellman equation (i.e, max-Bellman equation) is given by:

$$Q^{\pi}_{\max}(s_t, a_t) = \mathbb{E}_{\substack{s_{t+1} \sim P(\cdot|s_t, a_t) \\ a_{t+1} \sim \pi(\cdot|s_{t+1})}} \left[\max \left(r(s_t, a_t), \gamma Q^{\pi}_{\max}(s_{t+1}, a_{t+1}) \right) \right] \tag{2}$$

Based on Equation 2, we can now define the *max-Bellman* evaluation operator and the *max-Bellman* optimality operator. For any function $Q : \mathcal{S} \times A \to \mathbb{R}$ and for any state-action pair (s, a),

$$(\mathcal{M}^{\pi} Q)(s, a) \triangleq \max \left(r(s, a), \gamma \mathbb{E}_{\substack{s' \sim P(\cdot|s,a) \\ a' \sim \pi(\cdot|s')}} \left[Q(s', a') \right] \right)$$

$$(\mathcal{M}^{\star} Q)(s, a) \triangleq \max \left(r(s, a), \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q(s', a') \right] \right)$$

$\mathcal{M}^{\star}$ and $\mathcal{M}^{\pi} : (\mathcal{S} \times \mathcal{A} \to \mathbb{R}) \to (\mathcal{S} \times \mathcal{A} \to \mathbb{R})$ are operators that takes in a Q function and returns another modified Q function by assigning the Q value of the state s, action a, to be the maximum of the reward obtained at the same state and action (s, a) and the discounted future expected Q value.

Proposition 1. *The operators have the following properties.*

- *Monotonicity: let $Q_1, Q_2 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ such that $Q_1 \geq Q_2$ element-wise. Then,*

$$\mathcal{M}^{\pi} Q_1 \geq \mathcal{M}^{\pi} Q_2 \text{ and } \mathcal{M}^{\star} Q_1 \geq \mathcal{M}^{\star} Q_2$$

- *Contraction: both operators are γ-contraction in supremum norm i.e, for any $Q_1, Q_2 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$,*

$$\|\mathcal{M}^{\pi} Q_1 - \mathcal{M}^{\pi} Q_2\|_{\infty} \leq \gamma \|Q_1 - Q_2\|_{\infty} \tag{3}$$
$$\|\mathcal{M}^{\star} Q_1 - \mathcal{M}^{\star} Q_2\|_{\infty} \leq \gamma \|Q_1 - Q_2\|_{\infty} \tag{4}$$

Proof. We will provide a proof only for $\mathcal{M}^{\star}$. The proof of $\mathcal{M}^{\pi}$ is similar and is provided in Section A of the Appendix.

Monotonicity: Let Q_1 and Q_2 be two functions such that $Q_1(s,a) \geq Q_2(s,a)$ for any state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$. We then have:

$$\max_{a' \in \mathcal{A}} Q_1(s,a') \geq Q_2(s,a), \forall (s,a)$$

By the definition of $\max$, we obtain:

$$\max_{a' \in \mathcal{A}} Q_1(s,a') \geq \max_{a' \in \mathcal{A}} Q_2(s,a'), \forall s$$

and by linearity of expectation, we have:

$$\gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q_1(s',a') \right] \geq \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q_2(s',a') \right], \forall (s,a)$$

Since,

$$\mathcal{M}^{\star} Q_1(s,a) \geq \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q_1(s',a') \right]$$

we get:

$$\mathcal{M}^{\star} Q_1(s,a) \geq \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q_2(s',a') \right]$$

Moreover, because $\mathcal{M}^{\star} Q_1(s,a) \geq r(s,a)$, we obtain :

$$\mathcal{M}^{\star} Q_1(s,a) \geq \max \left(r(s,a), \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q_2(s',a') \right] \right) = \mathcal{M}^{\star} Q_2(s,a)$$

which is the desired result.

Contraction: Denote $f_i(s,a) = \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q_i(s',a') \right]$, the expected action-value function of the next state. By using the fact that $\max(x,y) = 0.5(x + y + |x - y|), \forall (x,y) \in \mathbb{R}^2$, we obtain:

$$\begin{aligned}
\max(r, f_1) - \max(r, f_2) &= 0.5 \left(r + f_1 + |r - f_1| \right) - 0.5 \left(r + f_2 + |r - f_2| \right) \\
&= 0.5 \left(f_1 - f_2 + |r - f_1| - |r - f_2| \right) \\
&\leq 0.5 \left(f_1 - f_2 + |r - f_1 - (r - f_2)| \right) \\
&= 0.5 \left(f_1 - f_2 + |f_1 - f_2| \right) \\
&\leq |f_1 - f_2|
\end{aligned}$$

Therefore

$$\begin{aligned}
\|\mathcal{M}^{\star} Q_1 - \mathcal{M}^{\star} Q_2\|_{\infty} &= \| \max(r, f_1) - \max(r, f_2) \|_{\infty} \\
&\leq \|f_1 - f_2\|_{\infty} \\
&\leq \gamma \| \max_{a'} Q_1(\cdot, a') - \max_{a'} Q_2(\cdot, a') \|_{\infty} \\
&\leq \gamma \|Q_1 - Q_2\|_{\infty} \qquad (\max \text{ is a non-expansion})
\end{aligned}$$

□

The left hand side of this equation represents the largest difference between the two Q-functions. Recall that γ lies in the range $[0, 1)$ and hence all differences are guaranteed go to zero in the limit. The Banach fixed point theorem then lets us conclude that the operator $\mathcal{M}^\star$ admits a fixed point.

We denote the fixed point of $\mathcal{M}^\star$ by ${Q^\star}_{\max}$. Based on equation **??**, we see that $Q^\pi_{\max}$ is the fixed point of $\mathcal{M}^\pi$. In the next proposition, we will prove that $Q^\star_{\max}$ corresponds to the optimal action-value function in the sense that $Q^\star_{\max} = \max_\pi Q^\pi_{\max}$.

Proposition 2 (Optimal policy)**.** *The deterministic policy $\pi^\star$ is defined as $\pi^\star(s) = \arg\max_{a\in\mathcal{A}} Q^\star_{\max}(s, a)$. $\pi^\star$, the greedy policy with respect to $Q^\star_{\max}$, is the optimal policy and for any stationary policy π, $Q^{\pi^\star}_{\max} = Q^\star_{\max} \geq Q^\pi_{\max}$.*

Proof. By the definition of greediness and the fact that $Q^\star$ is the fixed point of $\mathcal{M}^\star$, we have: $\pi^\star(s) = \arg\max_{a\in\mathcal{A}} Q^\star_{\max}(s, a) \Rightarrow \mathcal{M}^{\pi^\star} Q^\star = \mathcal{M}^\star Q^\star_{\max} = Q^\star_{\max}$. This proves that $Q^\star_{\max}$ is the fixed point of the evaluation operator $\mathcal{M}^{\pi^\star}$, which implies that $Q^\star_{\max}$ is the action-value function of $\pi^\star$.

For any function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and any policy π, we have $\mathcal{M}^\star Q \geq \mathcal{M}^\pi Q$. Using monotonicity, we have

$$(\mathcal{M}^\star)^2 Q = \mathcal{M}^\star(\mathcal{M}^\star Q) \geq \mathcal{M}^\star(\mathcal{M}^\pi Q) \geq \mathcal{M}^\pi(\mathcal{M}^\pi Q) = (\mathcal{M}^\pi)^2 Q. \tag{5}$$

We then use induction to show that for any $n \geq 1$, $(\mathcal{M}^\star)^n Q \geq (\mathcal{M}^\pi)^n Q$. As both operators are contractions, by the fixed-point theorem, $(\mathcal{M}^\star)^n Q$ and $(\mathcal{M}^\pi)^n Q$ converge to $Q^\star_{\max}$ and $Q^\pi_{\max}$, respectively, when n goes to infinity. We conclude then that $Q^\star_{\max} \geq Q^\pi_{\max}$. □

Thus, we have shown that and optimality operator defined based on our novel formulation of Bellman equation converges to a fixed point (Proposition 1) and that fixed point is the optimal policy (Proposition 2).

4 Experiments

This section shows the benefits of using max-Bellman in a simple grid world and in the real-world domain of drug discovery.

4.1 Toy example - Gold mining environment

Our first demonstration is on a toy domain with multiple goldmines in a 3×12 grid [*]. The agent starts in the bottom left corner and at each time step, can choose from among the four cardinal actions: up, down, left and right. The environment is deterministic with respect to the action transitions and the agent cannot leave the grid. All states in the grid that are labeled with values other than -1 represent goldmines and each value represents the reward that the agent will collect upon reaching that particular state. Transitions into a non-goldmine state results in a reward of -1. A goldmine's reward can be accessed only once and after it has been mined, its goldmine status is revoked, and the reward received upon further visitation is -1. The episode terminates after 11 time steps and the discount factor is $\gamma = 0.99$. The observation is a single integer denoting the state number of the agent. Thus, this is a non-Markovian setting since the environment observation does not communicate which goldmines have already been visited, and also because the timestep information is not included in the state observation.

The environment is shown in Figure 1. If the agent goes up to the top row and continues to move right, it will only get a cumulative return of 26.8. If it instead traverses the bottom row to the right, it can receive a cumulative return of 27.5, which is the highest cumulative return possible in this environment.

We test both Q-learning and Max-Q (i.e. Q-learning based on our proposed max-Bellman formulation), on this environment. As usual, the one step TD update rule for Q-learning is:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

[*]The environment and algorithm code is submitted as supplementary material and will be open sourced after acceptance for replication purposes.

1.0	1.1	1.2	1.3	1.4	1.5	2.0	2.1	7.2	9.0	-1.0	-1.0
-1.0	-8.0	-8.0	-8.0	-8.0	-8.0	-8.0	-8.0	-8.0	-8.0	-8.0	-8.0
	-1.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	6.0

Figure 1: A visualization of the gold-mining toy example. The bottom left state, shown in green, denotes the starting state. States with values other than -1 denote the goldmines and the values denote their respective rewards.

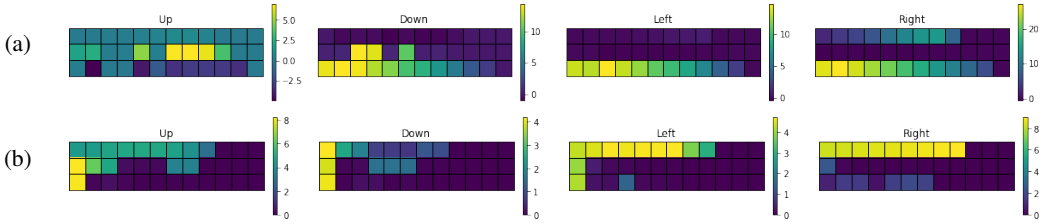


Figure 2: The learned q-values for **(a)** Q-learning **(b)** Max-Q

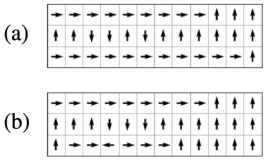
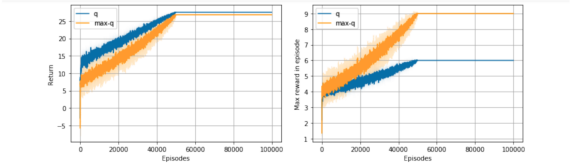


Figure 3: Learned policies for **(a)** Q-learning **(b)** Max-Q



(a) Average return (b) Maximum reward per episode

Figure 4: Comparison between Q-learning and Max-Q

The one step TD update rule for Max-Q is:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(\max(r_t, \gamma \max_a Q(s_{t+1}, a)) - Q(s_t, a_t))$$

For both algorithms, we use learning rate $\alpha = 0.001$, and decay epsilon from 0.2 to 0.0 linearly over 50000 episodes, after which ϵ remains fixed at 0. Figure 2 shows the learned Q-values and Figure 3 shows a difference in the final behavior learned by the two algorithms. Q-learning seems to prefer the path with highest cumulative return (along the bottom row) while Max-Q prefers the path with highest maximum reward (reward of $+9$ in the top row). The learned policies consistently reflect this behavior. Over 10 independent runs of each algorithm, Q-learning's policy always converges to moving along the bottom row and achieves expected cumulative return of 27.5. On the other hand, Max-Q always prefers the top row and achieves expected cumulative return of 26.8, but accomplishes its goal of maximizing the expected maximum reward in an episode (i.e, reaching the highest rewarding state). Also, Max-Q has worse initial learning performance in terms of the cumulative return, which can be explained by the agent wanting to move from the bottom row to the top row, despite the -8 penalty. This desire to move upwards at any cost is because the agent is pulled towards the $+9$, and does not care about any intermediate negative rewards that it may encounter.

Figure 4 shows a quantitative comparison between Q-learning and Max-Q. Figure 4a shows a comparison in terms of the average episodic return. Q-learning achieved optimal performance in terms of cumulative return and therefore has no incentive to direct the agent towards the maximum reward of $+9$. Max-Q on the other hand, converges to the path of following the top row to reach the highest reward of $+9$. This can be seen more clearly in Figure 4b, which shows a comparison of the maximum reward obtained in each episode. Each curve is averaged over 10 runs, and the shaded region represents 1 standard deviation. We also perform experiments in a fully Markovian setting for a similar environment. These results are shown in the Appendix.

4.2 DRUG DISCOVERY

We give a brief summary of PGFS (Gottipati et al. (2020)) for *de novo* drug design here and then incorporate the max-Bellman formulation derived above. PGFS operates in the realm of off-policy continuous action space algorithms. The actor module Π that consists of f and π networks predicts a continuous action a (that is in the space defined by the feature representations of all second reactants). Specifically, the f network takes in the current state s (reactant-1 $R^{(1)}$) as input and outputs the best reaction template T. The π network takes in both $R^{(1)}$ and T as inputs and outputs the continuous action a. The environment then takes in a and computes k closest valid second reactants ($R^{(2)}$). For each of these $R^{(2)}$s, we compute the corresponding product of the chemical reaction between $R^{(1)}$ and $R^{(2)}$, compute the reward for the obtained produce and choose the product (next state, s_{t+1}) that corresponds to the highest reward. All these quantities are stored in the replay buffer. The authors leveraged TD3 (Fujimoto et al., 2018a) algorithm for updating actor (f, π) and critic (Q) networks. More specifically, the following steps are followed after sampling a random minibatch from the buffer (replay memory):

First, the actions for the next time step (T_{i+1} and a_{i+1}) are computed by passing the state input ($R^{(1)}_{i+1}$) through the target actor network (i.e, the parameters of the actor networks are not updated in this process). Then, the one step TD target is computed:

$$y_i = r_i + \gamma \min_{j=1,2} \text{Critic-target}(\{R^{(1)}_{i+1}, T_{i+1}\}, a_{i+1})$$

In the proposed approach "MB (max-Bellman) + PGFS", we compute the one-step TD target as

$$y_i = \max[r_i, \gamma \min_{j=1,2} \text{Critic-target}(\{R^{(1)}_{i+1}, T_{i+1}\}, a_{i+1})]$$

The critic loss and the policy loss are defined as:

$$\mathcal{L}_{\textit{critic}} = \sum_i |y_i - Q(\{R^{(1)}_i, T_i\}, a_i)|^2$$

$$\mathcal{L}_{\textit{policy}} = -\sum_i \text{Critic}(R^{(1)}_i, \text{Actor}(R^{(1)}_i))$$

Additionally, like the PGFS algorithm, we also minimize an auxiliary loss to enable stronger gradient updates during the initial phases of training.

$$\mathcal{L}_{\textit{auxil}} = -\sum_i (T^{(1)}_{ic}, \log(f(R^{(1)}_{ic})))$$

and, the total actor loss $\mathcal{L}_{\textit{actor}}$ is a summation of the policy loss $\mathcal{L}_{\textit{policy}}$ and auxiliary loss $\mathcal{L}_{\textit{auxil}}$.

$$\mathcal{L}_{\textit{actor}} = \mathcal{L}_{\textit{policy}} + \mathcal{L}_{\textit{auxil}}$$

The parameters θ^Q of the Q-network are updated by minimizing the critic loss $\mathcal{L}_{\textit{critic}}$, and the parameters θ^f and θ^π of the actor networks f and π are updated my minimizing the actor loss $\mathcal{L}_{\textit{actor}}$. A more detailed description of the algorithm, pseudo code and hyper parameters used in given in Section C of the Appendix.

We compared the performance of the proposed formulation "MB (max-Bellman) + PGFS" with PGFS, and random search (RS) where reaction templates and reactants are chosen randomly at every time step, starting from initial reactant randomly sampled from ENAMINE dataset (which contains a set of roughly 150,000 reactants). We evaluated the approach on five rewards: QED (that measures the drug like-ness: Bickerton et al. (2012)), clogP (that measures lipophilicity: You et al. (2018b)) and activity predictions against three targets (Gottipati et al. (2020)): HIV-RT, HIV-INT, HIV-CCR5. While PGFS demonstrated state-of-the-art performance on all these rewards across different metrics (maximum reward achieved, mean of top-100 rewards, performance on validation set, etc.) when compared to the existing *de novo* drug design approaches (including the ones that do not implicitly or explicitly account for synthesizability and just optimize directly for the given reward function), we show that the proposed approach (PGFS+MB) performed better than PGFS (i.e, better than the existing state-of-the-art methods) on all the five rewards across all the metrics. For a fairness in comparison, like PGFS, we only performed hyper parameter tuning over policy noise and noise clip

and trained only for 400,000 time steps. However, in the proposed formulation, we noticed that the performance is sensitive to the discount factor γ and the optimal γ is different for each reward.

Table 1 compares the maximum reward achieved during the entire course of training of 400,000 time steps. We notice that the proposed approach PGFS+MB achieved highest reward compared to existing state-of-the-art approaches. Table 2 compares the mean (and standard deviation) of top 100 rewards (i.e., molecules) achieved by each of the methods over the entire course of training with and without applicability domain (AD) (Tropsha (2010), Gottipati et al. (2020)). We again note that the proposed formulation performed better than existing methods on all three rewards both with and without AD. In Figure 5, we compare the performance based on the rewards achieved starting from a fixed validation set of 2000 initial reactants. For all the three HIV reward functions, we notice that PGFS+MB performed better than existing reaction-based RL approaches (i.e, PGFS and RS) in terms of reward achieved at every time step of the episode (Figure 5a), and in terms of maximum reward achieved in each episode (Figure 5b).

Table 1: Performance comparison of reaction based *de novo* drug design algorithms in terms of maximum reward achieved

Method	QED	clogP	RT	INT	CCR5
ENAMINEBB	**0.948**	5.51	7.49	6.71	8.63
RS	**0.948**	8.86	7.65	7.25	8.79 (8.86)
PGFS	**0.948**	27.22	7.89	7.55	9.05
PGFS+MB	**0.948**	**27.60**	**7.97**	**7.67**	**9.20 (9.26)**

Table 2: Statistics of the top-100 produced molecules with highest predicted HIV scores for every reaction-based method used and Enamine's building blocks

	NO AD			AD		
Scoring	RT	INT	CCR5	RT	INT	CCR5
ENAMINEBB	6.87 ± 0.11	6.32 ± 0.12	7.10 ± 0.27	6.87 ± 0.11	6.32 ± 0.12	6.89 ± 0.32
RS	7.39 ± 0.10	6.87 ± 0.13	8.65 ± 0.06	7.31 ± 0.11	6.87 ± 0.13	8.56 ± 0.08
PGFS	**7.81** ± 0.03	7.16 ± 0.09	8.96 ± 0.04	7.63 ± 0.09	7.15 ± 0.08	8.93 ± 0.05
PGFS+MB	**7.81** ± 0.01	**7.51** ± 0.02	**9.06** ± 0.01	**7.75** ± 0.04	**7.51** ± 0.04	**9.01** ± 0.05

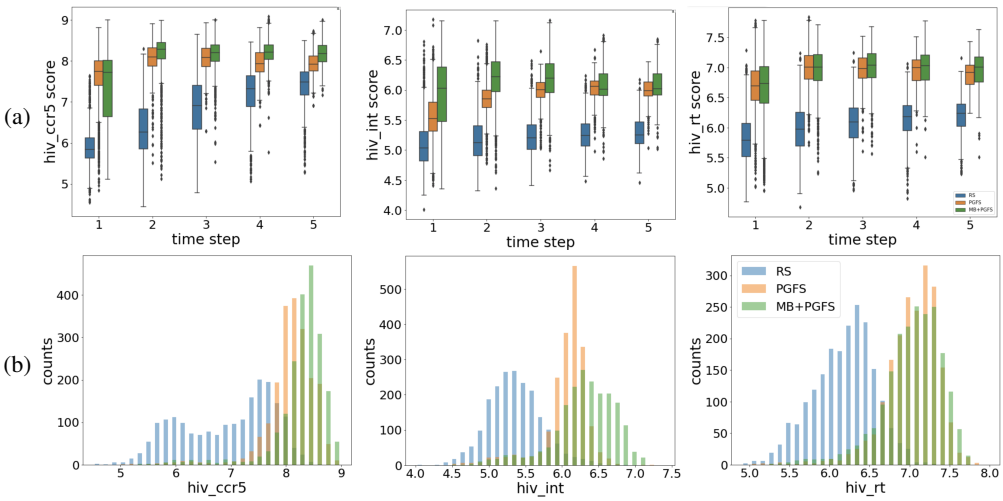


Figure 5: Comparison of the three methods: RS (blue), PGFS (orange) and PGFS+MB (green) based on the rewards achieved starting with the 2000 initial reactants from a fixed validation set.

REFERENCES

Sungsoo Ahn, Junsu Kim, Hankook Lee, and Jinwoo Shin. Guiding deep molecular optimization with genetic exploration, 2020.

Richard Bellman. Some applications of the theory of dynamic programming - A review. *Oper. Res.*, 2(3):275–288, 1954.

G Richard Bickerton, Gaia V Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90, 2012.

John Bradshaw, Brooks Paige, Matt J. Kusner, Marwin H. S. Segler, and José Miguel Hernández-Lobato. A model to search for synthesizable molecules. *CoRR*, abs/1906.05221, 2019.

Nathan Brown, Ben McKay, François Gilardoni, and Johann Gasteiger. A graph-based genetic algorithm and its application to the multiobjective evolution of median molecules. *Journal of chemical information and computer sciences*, 44(3):1079–1087, 2004.

Nathan Brown, Marco Fiscato, Marwin HS Segler, and Alain C Vaucher. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59(3): 1096–1108, 2019.

Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. Active neural localization. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=ry6-G_66b`.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018a.

Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018b.

Wenhao Gao and Connor W. Coley. The synthesizability of molecules proposed by generative models. *Journal of Chemical Information and Modeling*, Apr 2020. ISSN 1549-960X. doi: 10.1021/acs.jcim.0c00174.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Sai Krishna Gottipati, Boris Sattarov, Sufeng Niu, Yashaswi Pathak, Haoran Wei, Shengchao Liu, Karam M. J. Thomas, Simon Blackburn, Connor W. Coley, Jian Tang, Sarath Chandar, and Yoshua Bengio. Learning to navigate the synthetically accessible chemical space using reinforcement learning. In *Proceedings of the 37th International Conference on International Conference on Machine Learning*, ICML'20, 2020.

Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. Constrained bayesian optimization for automatic chemical design using variational autoencoders. *Chem. Sci.*, 11:577–586, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. Convergence of stochastic iterative dynamic programming algorithms. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pp. 703–710, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

Jan H Jensen. A graph-based genetic algorithm and generative model/monte carlo tree search for the exploration of chemical space. *Chemical science*, 10(12):3567–3572, 2019.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2323–2332, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Takashi Kamihigashi and Cuong Le Van. Necessary and sufficient conditions for a solution of the bellman equation to be the value function: A general principle. 2015.

Mathilde Koch, Thomas Duigou, and Jean-Loup Faulon. Reinforcement learning for bio-retrosynthesis. *bioRxiv*, 2019.

Ksenia Korovina, Sailun Xu, Kirthevasan Kandasamy, Willie Neiswanger, Barnabas Poczos, Jeff Schneider, and Eric P. Xing. ChemBO: Bayesian Optimization of Small Organic Molecules with Synthesizable Recommendations. *arXiv:1908.01425 [physics, stat]*, August 2019. arXiv: 1908.01425.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9(1):48, 2017.

Brenden K. Petersen. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients, 2020.

Mariya Popova, Olexandr Isayev, and Alexander Tropsha. Deep reinforcement learning for de novo drug design. *Science advances*, 4(7):eaap7885, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pp. 412–422. Springer, 2018.

Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38(3):287–308, 2000.

Csaba Szepesvári. The asymptotic convergence-rate of q-learning. In *NIPS*, pp. 1064–1070, 1997.

Alexander Tropsha. Best practices for qsar model development, validation, and exploitation. *Molecular informatics*, 29(6-7):476–488, 2010.

John N Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 16(3): 185–202, 1994.

Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16), 2020.

Christopher J. C. H. Watkins and Peter Dayan. Technical note: q -learning. *Mach. Learn.*, 8(3–4): 279–292, May 1992.

Robin Winter, Floriane Montanari, Andreas Steffen, Hans Briem, Frank Noé, and Djork-Arné Clevert. Efficient multi-objective molecular optimization in a continuous latent space. *Chemical science*, 10(34):8016–8024, 2019.

Jiaxuan You, Bowen Liu, Rex Ying, Vijay S. Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *CoRR*, abs/1806.02473, 2018a.

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in neural information processing systems*, pp. 6410–6421, 2018b.

Zhenpeng Zhou, Steven M. Kearnes, Li Li, Richard N. Zare, and Patrick Riley. Optimization of molecules via deep reinforcement learning. *CoRR*, abs/1810.08678, 2018.

A PROOF - MONOTONICITY AND CONTRACTION PROPERTIES OF $\mathcal{M}^\pi$

Proof. **Monotonicity**: Let Q_1 and Q_2 be two functions such that $Q_1(s,a) \geq Q_2(s,a)$ for any state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$. By linearity of expectation, we have $\gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} [Q_1(s',a')] \geq \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} [Q_2(s',a')], \forall (s,a)$. As $\mathcal{M}^\pi Q_1(s,a) \geq \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} [Q_1(s',a')]$, we get $\mathcal{M}^\pi Q_1(s,a) \geq \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} [Q_2(s',a')]$. Moreover, because $\mathcal{M}^\pi Q_1(s,a) \geq r(s,a)$, we obtain

$$\mathcal{M}^\pi Q_1(s,a) \geq \max \left(r(s,a), \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} [Q_2(s',a')] \right) = \mathcal{M}^\pi Q_2(s,a)$$

which is the desired result.

Contraction: Let us denote the expected action-value function of the next state by $f_i(s,a)$, obtaining the following equation:

$$f_i(s,a) = \gamma \mathbb{E}_{\substack{s' \sim P(\cdot|s,a) \\ a' \sim \pi(\cdot|s')}} [Q_i(s',a')])$$

By using the fact that $\max(x,y) = 0.5(x + y + |x-y|), \forall (x,y) \in \mathbb{R}^2$, we obtain

$$\begin{aligned}
\max(r, f_1) - \max(r, f_2) &= 0.5 \left(r + f_1 + |r - f_1| \right) - 0.5 \left(r + f_2 + |r - f_2| \right) \\
&= 0.5 \left(f_1 - f_2 + |r - f_1| - |r - f_2| \right) \\
&\leq 0.5 \left(f_1 - f_2 + |r - f_1 - (r - f_2)| \right) \\
&= 0.5 \left(f_1 - f_2 + |f_1 - f_2| \right) \\
&\leq |f_1 - f_2|
\end{aligned}$$

Therefore

$$\begin{aligned}
\|\mathcal{M}^\pi Q_1 - \mathcal{M}^\pi Q_2\|_\infty &= \| \max(r, f_1) - \max(r, f_2)\|_\infty \\
&\leq \|f_1 - f_2\|_\infty \\
&\leq \gamma \|\mathbb{E}_{a'} Q_1(\cdot, a') - \mathbb{E}_{a'} Q_2(\cdot, a')\|_\infty \\
&= \gamma \|\mathbb{E}_{a'} (Q_1(\cdot, a') - Q_2(\cdot, a'))\|_\infty \\
&\leq \gamma \| \max_{a'} (Q_1(\cdot, a') - Q_2(\cdot, a'))\|_\infty \\
&\leq \gamma \max_{a'} \|(Q_1(\cdot, a') - Q_2(\cdot, a'))\|_\infty \\
&= \gamma \|Q_1 - Q_2\|_\infty
\end{aligned}$$

□

B MARKOVIAN GOLDMINING TOY EXAMPLE

We also run a comparison between Q-learning and Max-Q on a fully Markovian setting on a similar goldmining environment. The Markovian property is enforced by not removing the rewards once the goldmines have been visited, and by also including the timestep information in the state observation. From the learning perspective, the latter is enforced simply by making the Q-value table 3-dimensional with the first dimension referencing the time step.

The markovian setting is on a similar toy domain with multiple goldmines in a 3×6 grid. The agent starts in the bottom left corner and at each time step, can choose from among the four cardinal actions: up, down, left and right. The environment is deterministic with respect to the action transitions and the agent cannot leave the grid. All states in the grid that are labeled with values other than -1 represent goldmines and each value represents the reward that the agent will collect upon reaching that particular state. Transitions into a non-goldmine state results in a reward of -1. The episode terminates after 5 time steps and the discount factor is $\gamma = 0.99$. The observation is a tuple consisting of an integer denoting the state number of the agent and an integer denoting the current timestep. Thus, this is a Markovian setting.

The environment is shown in Figure 6. If the agent goes up to the top row and continues to move right, it will only get a cumulative return of 6. If it instead traverses the bottom row to the right, it can receive a cumulative return of 9, which is the highest cumulative return possible in this environment.

-1.0	-1.0	-1.0	10.0	-1.0	-1.0
-1.0	-8.0	-8.0	-8.0	-8.0	-8.0
	-1.0	-1.0	-1.0	6.0	6.0

Figure 6: A visualization of the gold-mining toy example. The bottom left state, shown in green, denotes the starting state. States with values other than -1 denote the goldmines and the values denote their respective rewards.

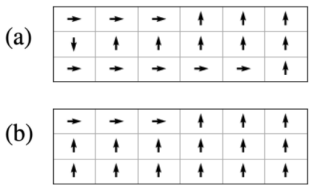
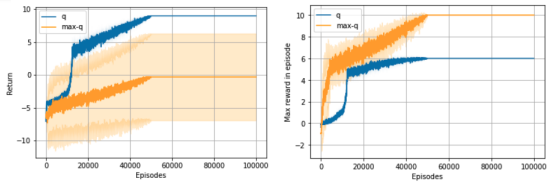


Figure 7: Learned policies for **(a)** Q-learning **(b)** Max-Q



(a) Average return (b) Maximum reward per episode

Figure 8: Comparison between Q-learning and Max-Q

A comparison of cumulative return and maximum reward is shown in Figure 8b. Q-learning learns the optimal policy while Max-Q learning the policy to reach the highest rewarding state. Thus, consistent results are obtained for the Markovian setting as well. The exploration schedule and hyperparameters are the same as those used for the non-Markovian setting.

For the Markovian setting, the grid was made smaller and the number of goldmines were reduced. These changes were made only to make the task easier to solve. Since the rewards do not vanish in this setting, not applying these minor changes meant that it was a much harder exploration problem and it was observed that the agent simply oscillated between states. Without these changes, the agent took much longer to converge to the optimal policy. The increased variance in the average return for Max-Q in Figure 8a is as expected, since there is no incentive for the Max-Q agent to avoid the -8 penalties. This high variance is not observed in the non-Markovian toy example only because the rewards guide the agent around the negative reward filled middle row.

It should be noted that in the Markovian setting, since the Q-value table has 3 dimensions, there is no clear way to visualize the learned policy. Figure 7 has been obtained by plotting the greedy action for each state after applying the mean across the time axis of the Q-value table.

C PSEUDO CODE - PGFS+MB

In this section, we explain the PGFS+MB algorithm in detail and provide the pseudo code.

The actor module Π consists of two networks f and π. The role of the actor module is to compute the action a for a given state s. In this case, the state is the reactant $R^{(1)}$, and the action outputs are reaction template T and a tensor a in the space defined by feature representation of all second reactants. First, the reactant $R^{(1)}$ is passed through the f-network to compute the template tensor T that contains the probability of each of the reaction templates.

$$T = f(R^{(1)})$$

For a given reactant $R^{(1)}$, only few reaction templates are eligible to participate in a reaction involving $R^{(1)}$. Thus, all the invalid reaction templates are masked off by multiplying element-wise with a template mask T_{mask}, which is a binary tensor with value 1 if its a valid template, 0 otherwise.

$$T = T \odot T_{mask}$$

Finally, the reaction template T in one-hot tensor format is obtained by applying Gumbel softmax operation to the masked off template T. It is parameterized by a temperature parameter τ that is slowly decayed from 1.0 to 0.1

$$T = \text{GumbelSoftmax}(T, \tau)$$

The one-hot template along with the reactant $R^{(1)}$ is passed through the π network to obtain the action a in the space defined by feature representation of all second reactants.

$$a = \pi(R^{(1)}, T)$$

The critic module consists of the Q-network and computes the $Q(s, a)$ values. In this case, it takes in the reactant $R^{(1)}$, reaction template T, action a and compute its Q value: $Q(R^{(1)}, T, a)$.

For the fairness in comparison, we used the exact same network sizes as described in the PGFS paper i.e, The f-network has four fully connected layers with 256, 128, 128 neurons in the hidden layers. The π network has four fully connected layers with 256, 256, 167 neurons in the hidden layers. All the hidden layers use ReLU activation whereas the final layer uses tanh activation. The Q-network also has four fully connected layers with 256, 64, 16 neurons in the hidden layers, with ReLU activation for all the hidden layers and linear activation for the final layer.

The environment takes in the current state s and action a and computes the next state and reward. First, it computes the set of second reactants that are eligible to participate in a reaction involving chosen reaction template T

$$\mathcal{R}^{(2)} = \text{GetValidReactants}(T)$$

The k valid reactants closest to the action a are then obtained by passing the action a and set of valid second reactants $\mathcal{R}^{(2)}$ through the k nearest neighbours module.

$$\mathcal{A} = \text{kNN}(a, \mathcal{R}^{(2)})$$

For each of these k second reactants, we compute the corresponding products $R^{(1)}_{t+1}$ obtained involving the reactant $R^{(1)}$ and reaction template T by passing them through a forward reaction predictor module, and then compute the corresponding rewards by passing the obtained products through a scoring function prediction module.

$$\mathcal{R}^{(1)}_{t+1} = \text{ForwardReaction}(R^{(1)}, T, \mathcal{A})$$

$$\mathcal{R}ewards = \text{ScoringFunction}(\mathcal{R}^{(1)}_{t+1})$$

Then, the product and the reward corresponding to the maximum reward are chosen and returned by the environment. In all our experiments, we use $k = 1$.

During the optimization ("backward") phase, we compute the actions for next time step T_{i+1}, a_{i+1} using target actor network on a randomly sampled mini-batch.

$$T_{i+1}, a_{i+1} = \text{Actor-target}(R^{(1)}_{i+1})$$

We then compute one-step TD (temporal difference) target y_i (using the proposed max-Bellman formulation) as the maximum of reward at the current time step and discounted Q value (computed by critic-target) for the next state, next action pair. To incorporate the clipped double Q-learning formulation used in TD3 (Fujimoto et al. (2018a)) to prevent over-estimation bias, we use two critics and only take the minimum of the two critics.

$$y_i = \max[r_i, \gamma \min_{j=1,2} \text{Critic-target}(\{R^{(1)}_{i+1}, T_{i+1}\}, a_{i+1})]$$

Note that this is different from PGFS (Gottipati et al. (2020)) where the authors compute the TD target using the standard Bellman formulation: $y_i = r_i + \gamma \min_{j=1,2} \text{Critic-target}(\{R^{(1)}_{i+1}, T_{i+1}\}, a_{i+1})$. We then compute the critic loss $\mathcal{L}_{\text{critic}}$ as the mean squared error between the one-step TD target y_i and the Q-value (computed by critic) of the current state, action pair.

$$\mathcal{L}_{\text{critic}} = \sum_i |y_i - \textsc{Critic}(R^{(1)}_i, T_i, a_i)|^2$$

The policy loss $\mathcal{L}_{policy}$ is negative of the critic value of the state, action pair where the actions are computed by the current version of the actor network

$$\mathcal{L}_{policy} = -\sum_i \textsc{Critic}(R_i^{(1)}, \textsc{Actor}(R_i^{(1)}))$$

Like in PGFS, to enable faster learning during initial phases of the training, we also minimize an auxiliary loss which is the cross entropy loss between the predicted template and the actual valid template

$$\mathcal{L}_{auxil} = -\sum_i (T_i^{(1)}, log(f(R_i^{(1)})))$$

Thus, the total actor loss is the sum of policy loss and the auxiliary loss

$$\mathcal{L}_{actor} = \mathcal{L}_{policy} + \mathcal{L}_{auxil}$$

The parameters of all the actor and critic networks (f, π, Q) are updated by minimizing the actor and critic losses respectively.

$$\min \mathcal{L}_{\text{actor}}, \mathcal{L}_{\text{critic}}$$

Algorithm 1 PGFS+MB

procedure ACTOR($R^{(1)}$)
  $T \leftarrow f(R^{(1)})$
  $T \leftarrow T \odot T_{mask}$
  $T \leftarrow$ GumbelSoftmax(T, τ)
  $a \leftarrow \pi(R^{(1)}, T)$
  return T, a
procedure CRITIC($R^{(1)}$, T, a)
  return $Q(R^{(1)}, T, a)$
procedure ENV.STEP($R^{(1)}, T, a$)
  $\mathcal{R}^{(2)} \leftarrow$ GetValidReactants(T)
  $\mathcal{A} \leftarrow$ kNN$(a, \mathcal{R}^{(2)})$
  $\mathcal{R}_{t+1}^{(1)} \leftarrow$ ForwardReaction$(R^{(1)}, T, \mathcal{A})$
  $\mathcal{R}ewards \leftarrow$ ScoringFunction$(\mathcal{R}_{t+1}^{(1)})$
  $r_t, R_{t+1}^{(1)}, done \leftarrow \arg\max \mathcal{R}ewards$
  return $R_{t+1}^{(1)}, r_t$, done
procedure BACKWARD(buffer minibatch)
  $T_{i+1}, a_{i+1} \leftarrow$ Actor-target$(R_{i+1}^{(1)})$
  $y_i \leftarrow \max[r_i, \gamma \min_{j=1,2}$ Critic-target$(\{R_{i+1}^{(1)}, T_{i+1}\}, a_{i+1})]$
  $\mathcal{L}_{\text{critic}} \leftarrow \sum_i |y_i - \textsc{Critic}(R_i^{(1)}, T_i, a_i)|^2$
  $\mathcal{L}_{policy} \leftarrow -\sum_i \textsc{Critic}(R_i^{(1)}, \textsc{Actor}(R_i^{(1)}))$
  $\mathcal{L}_{auxil} \leftarrow -\sum_i (T_i^{(1)}, log(f(R_i^{(1)})))$
  $\mathcal{L}_{actor} \leftarrow \mathcal{L}_{policy} + \mathcal{L}_{auxil}$
  $\min \mathcal{L}_{\text{actor}}, \mathcal{L}_{\text{critic}}$
procedure MAIN(f, π, Q)
  for episode = 1, M **do**
    sample $R_0^{(1)}$
    for t = 0, N **do**
      $T_t, a_t \leftarrow$ Actor$(R_t^{(1)})$
      $R_{t+1}^{(1)}, r_t$, done $\leftarrow$ env.step$(R_t^{(1)}, T_t, a_t)$
      store $(R_t^{(1)}, T_t, a_t, R_{t+1}^{(1)}, r_t$, done) in buffer
      sample a random minibatch from buffer
      Backward(minibatch)