# Leveraging Classical Planning for Automatic Reward Densification

**Nishanth Kumar**
njk@mit.edu

**Willie McClinton**
wbm3@mit.edu

**Viraj Parimi**
vparimi@mit.edu
*

## 1 Introduction

A core challenge with scaling up Deep Reinforcement Learning (Deep RL) for use in robotic tasks of practical interest is the task specification problem Agrawal [2021], which typically manifests as the difficulty of reward design. In particular, most robotic tasks are naturally specified by a goal state (e.g. the desired state of all objects for a "clean" room), which is captured by a 'one-hot' reward signal (i.e the agent is given positive reward when it is in the goal state and a small negative reward in every other state). Unfortunately, such a reward signal is too sparse to optimize, even for state-of-the-art algorithms. Thus, many RL researchers manually 'densify' the reward (e.g. in the popular benchmark MetaWorld problems Yu et al. [2019]). However, this can lead to undesirable behavior via "reward hacking" Clark and Amodei [2016]. As a result, reward densification is often a laborious process of tweaking the reward function to be more dense, training a Deep RL agent with the tweaked function, then tweaking the reward function again to prevent any undesirable behavior, and so on.

Given both the tediousness and difficulty of manually tuning reward functions, there have been several works that have attempted to either partially or fully automate this process. Several works Zou et al. [2019], Memarian et al. [2021] attempt to do so with Machine Learning (ML) techniques like meta-learning or self-supervision, but often require either small problem instances to train on, or do not generalize well to settings where the probability of randomly encountering positive reward is vanishingly small. Other works have provided the agent with access to additional information, such as task plans Grzes and Kudenko [2008] or Linear Temporal Logic (LTL) formulae Jiang et al. [2020]. However, these approaches have only been tried in simple, discrete domains that are poor approximations of robotic tasks.

In order to reduce the difficulty of reward function design in continuous robotics environments, we propose to develop a method that automatically densifies sparse, goal-based reward in robotic tasks such that the optimal policy is preserved. In particular, we propose to do this by leveraging task plans similar to Grzes and Kudenko [2008]. We hypothesize that for many robotic tasks, (1) while it is difficult for humans to specify a dense reward that cannot be hacked, it is easy to specify an abstract plannable model in PDDL McDermott et al. [1998] that conveys information about the dynamics of the domain, and (2) that valid abstract plans within this model can be leveraged to automatically densify sparse reward via potential-based reward shaping [Devlin and Kudenko, 2012] sufficiently enough for state-of-the-art RL approaches to solve these tasks. We perform an extensive empirical evaluation of our system across different PDDL models with varying granularity, choices of potential functions, choices of learning algorithms (PPO Schulman et al. [2017] and SAC Haarnoja et al. [2018]) and tasks. Overall, our findings suggest that intuitive PDDL models can help densify sparse reward tasks and improve the performance of learning algorithms, but enabling these algorithms to fully solve even simple robotic tasks still requires tuning of the PDDL models and potential shaping functions.

*For code and videos, see project webpage here

6.484 Computational Sensorimotor Learning Final Project (Spring 2022).

## 2 Related Work

Recent work has attempted to leverage ML techniques to automatically densify sparse reward. Zou et al. [2019] leverage meta-learning by training the agent on a number of smaller problem instances and then learning how best to initialize dense reward in a new problem. Importantly, these smaller instances must share the same reward function structure. By contrast, our approach does not require smaller problem instances and does not require the exact same reward function structure to hold across problems (we can generalize to any goal that is expressable within the PDDL model). Memarian et al. [2021] use the problem's sparse reward to generate a ranking over trajectories that can then be used to automatically densify the reward. However, this requires that the agent is able to generate trajectories via exploration that achieve some reward, which is often difficult in most robotic tasks.

Another widely studied approach to reward densification is to exploit structure in the reward function. More specifically Toro Icarte et al. [2022] proposed a reward machine where the idea is to expose the internal reward structure to an RL agent so that it can exploit it to learn better policies in a sample efficient manner. Similar to this work, there have been other approaches like Jiang et al. [2020] which provide the domain knowledge using Linear Temporal Logic (LTL) formulae. In contrast to these works, our approach requires neither a particularly structured reward function that can be exposed to an agent, nor input LTL formulae. Moreover, we focus on attempting to solve continuous robotic domains whereas most of these works apply their proposed techniques to discrete environments.

Early work by Grzes and Kudenko [2008] leveraged STRIPS style operator knowledge for discrete domains for guiding reward shaping. Building on this, Gehring et al. [2021] proposes the idea of leveraging domain-independent heuristic functions that are widely used in the classical planning literature to act as dense reward generators. These works have only explored discrete domains and defined a fixed potential function over the states. We take significant inspiration from these ideas and intend to work towards reward shaping applied to continuous domains.

## 3 Problem Formulation

We consider an environment represented as a Markov Decision Processes (MDP), in which an agent must learn to reach a given goal. A MDP is defined by the tuple $\langle S, A, T, R, \gamma \rangle$, where $S$ is the state space; $A$ is a continuous action space; $T : S \times A \times S \to [0, 1]$ is the transition function, which gives the probability of the next state conditioned on the previous state and action; $R : S \to \mathbb{R}$ is the reward function that will generate a sparse reward only if the goal is reached; and $\gamma : S \to [0, 1]$ is the discount rate. The objective in our MDP is to learn a policy $\pi : S \to [0, 1]^{|A|}$ probability distribution over the action space conditioned on the current state that maximizes the expected, cumulative, and discounted reward.

Given an MDP, we propose to also specify a PDDL model that can be seen as a tuple $\langle \Psi, \mathcal{S}_\Psi, \Omega, \mathcal{F}, \mathcal{O} \rangle$ that defines deterministic high-level transition model over the MDP. Here, $\Psi$ is a set of predicates, $\mathcal{S}_\Psi$ represents the space of high-level states, $\underline{\Omega}$ is the space of high-level operators grounded by objects $\mathcal{O}$, and partial function $\mathcal{F} : \mathcal{S}_\Psi \times \underline{\Omega} \nrightarrow \mathcal{S}_\Psi$ defines the transition dynamics.

A predicate $\psi$ is characterized by an ordered list of variables $(\lambda_1, ..., \lambda_m)$. For instance, the predicate *Holding* may, given a state and two objects, robot and block, describe whether the block is held by the robot in this state. A lifted atom is a predicate with variables (e.g., Holding(?robot, ?block)). A ground atom $\underline{\psi}$ consists of a predicate $\psi$ and objects $(o_1, ..., o_m)$.

**Definition 1 (High-level state)** *A* high-level state $\hat{s}$ *is the set of ground atoms under* $\Psi$ *that hold true in some MDP state* $s$*:*

$$\hat{s} = \textsc{Abstract}(s, \Psi)$$

**Definition 2 (Operator)** *An* operator *is a tuple* $\omega = \langle \textsc{Par}, \textsc{Pre}, \textsc{Eff}^+, \textsc{Eff}^- \rangle$ *where:* PAR *is an ordered list of* variables*, and* PRE, $\textsc{Eff}^+$, $\textsc{Eff}^-$ *are* preconditions*,* add effects*, and* delete effects *respectively, each a set of lifted atoms over* $\Psi$ *and* PAR*.*

**Definition 3 (Ground operator)** *A* ground operator $\underline{\omega} = \langle \omega, \delta \rangle$ *is an operator* $\omega$ *and a substitution* $\delta : \textsc{Par} \to \mathcal{O}$ *mapping parameters to objects. We use* $\underline{\textsc{Pre}}$, $\underline{\textsc{Eff}^+}$*, and* $\underline{\textsc{Eff}^-}$ *to denote the ground preconditions, ground add effects, and ground delete effects, where variables in* PAR *are substituted with objects under* $\delta$*.*

**Definition 4 (High-level transition model)** *The* high-level transition model *induced by predicates* $\Psi$ *and operators* $\Omega$ *is a partial function* $\mathcal{F} : \mathcal{S}_\Psi \times \underline{\Omega} \nrightarrow \mathcal{S}_\Psi$. $\mathcal{F}(\hat{s}, \underline{\omega})$ *is only defined if* $\underline{\omega}$ *is* applicable *in* $s$*:* $\underline{\text{PRE}} \subseteq \hat{s}$*. If defined,* $\mathcal{F}(\hat{s}, \underline{\omega}) \triangleq (\hat{s} - \underline{\text{EFF}^-}) \cup \underline{\text{EFF}^+}$.

With this high-level transition model we can come up with a plan $P$: $[\hat{s}_0, \underline{\omega}_1, ..., \underline{\omega}_f, \hat{s}_f]$ using an off-the-shelf PDDL planner like Fast Downward Helmert [2006]. This plan specifies a sequence of high-level states (subgoals), which can then be used to derive our potential-based reward function.

The goal of reward shaping is to provide guidance to our policy learning algorithm via a denser reward that encodes subgoal knowledge, in order to reduce the difficulty of exploration for an RL algorithm. For example the new shaped reward function $\hat{R}(s) = R(s) + F(s, s')$, where $F(s, s')$ is the state-based bonus shaping the reward, can be used instead of $R(s)$ to incentivize beneficial actions during training. Naturally, if reward shaping is used improperly it can change the optimal policy and lead to unintended behavior. To deal with this problem, potential-based reward shaping Ng et al. [1999] was proposed as the difference of some potential function $\phi$ defined over a source $s$ and a destination state $s'$:

$$F(s, s') = \gamma\phi(s') - \phi(s)$$

In the context of an abstract plan, we want the potential function to be applied to abstract states coming from the plan. We can obtain such states ($\hat{s}$) from the low-level state via Definition 1 Thus:

$$F(s, s') = \gamma\phi(\hat{s}') - \phi(\hat{s})$$

Potential-based reward shaping has the property that an optimal policy learned with the shaped reward $\hat{R}$ is guaranteed to be equivalent to the one learnt with the original reward $R$ Ng et al. [1999]. Given some plan $P$ induced by a high-level transition model $\mathcal{F}$, we can obtain a series of subgoal states $\hat{S}$ for the agent to reach sequentially by taking each grounded operator in $P$ and applying it from the initial state onward. Given a sequence of subgoals $\hat{S}$, we will follow Grzes and Kudenko [2008] and define $\phi(s)$ such that:

$$\phi(\hat{s}) = step(\hat{s})$$

where the function $step(\hat{s})$ returns the index of step in the plan at which the given high-level state $\hat{s}$ appears. We refer to this choice of potential function as a *plan-based* potential function.

Unfortunately, this cannot be used directly since it does not assign a potential for non-plan states, which are likely to be encountered during the course of exploration. Grzes and Kudenko [2008] address this issue by keeping track of the most-recent plan state that the agent achieves. However, it is not immediately obvious that this is correct, since doing this will make the potential function no longer solely dependent on the current state $s$[2]. Intuitively however, we can see that such a $\phi$ will still preserve optimal policies because we can just change the MDP's state-space to include the most-recent subgoal accomplished as part of the state (doing so will not change the optimal policy because it will not change the reward obtained by any particular trajectory). A formal proof of this is presented in the Appendix (Section 8.1).

Devlin and Kudenko [2012] showed that potential functions can be parameterized by both the state and timestep:

$$F(s, t, s', t') = \gamma\phi(\hat{s}', t') - \phi(\hat{s}, t)$$

where $t$ is the time the agent arrived at previous state $s$ and $t'$ is the current time when arriving at the current state $s'$ (i.e. $t < t'$). We refer to this choice of potential function as a *dynamic plan-based* potential function.

## 4 Method

In order to use either a plan-based or dynamic plan-based potential function as described in the previous section, it is necessary to be able to compute a high-level state given a particular low-level state. In general, specifying a function to accomplish this computation might be rather difficult.

[2]To the best of our knowledge, none of the related works that use this technique presented a proof that doing so preserves the optimal policy.

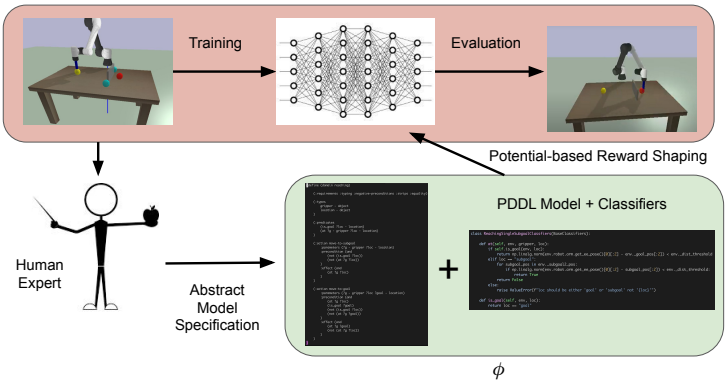


Figure 1: **An overview of our system for automatic reward densification.** The process illustrated in the top red box represents the typical RL pipeline. In our system, a human expert provides an abstract model of the environment consisting of a hand-specified PDDL model as well as classifier functions that take in a state and output the value of a particular ground predicate. Given this information, we can automatically increase the density of the original sparse reward and thereby improve the RL training process.

However, since a high-level state is nothing but a collection of predicates that are ground over all the objects in a problem, we can compute such a state by simply evaluating the values of all groundings of all predicates in the domain over all objects in the problem. To do this, we only require the specification of a binary predicate classifier $c_\psi$ for every predicate $\psi$ defined as follows:

**Definition 5 (Predicate Classifier)** *A predicate classifier function $c_\psi(o_1, ..., o_m, s)$ is a function associated with an $m$-arity predicate $\psi$ that takes in $m$ objects of the correct type and a low-level state and outputs a binary value indicating whether or not the predicate ground with the specified objects holds in the current state.*

$$c_\psi(o_1, ..., o_m, s) : s \mapsto \{True, False\}, \forall s \in S$$

Since PDDL domains do not generally contain a large number of predicates, requiring one classifier per predicate is a relatively reasonable demand from a human designer.

Our overall approach is to have human designers specify abstract models and associated predicate classifier functions for robotic tasks with sparse, goal-based reward, and then leverage these to automatically increase the density of the reward when an RL agent is trained on such a task. A high-level overview of this process is depicted in Fig. 1. The main unexplained part of this process is the computation of the potential-based reward given a state (illustrated as the green box in the Figure). The pseudocode for this process is shown in Algorithm 1 below. Intuitively, we first run a planner on the input PDDL to produce a plan. We then apply each action of this plan to produce a sequence of high-level subgoals. We initialize the '`max_idx`' variable to 0 and then simply call the pseudocode from Algorithm 1 for every state and timestep encountered during the rest of the trajectory. Note that we do not incentivize or penalize the agent to follow the given plan to account for scenarios where the agent might be able to learn better policies.

### 4.1 Potential Functions

We evaluated different choices of potential functions: (1) *plan-based* potential, (2) *dynamic plan-based* potential where we tried two variants, (i) *dynamic time-varying* potential, and (ii) *dynamic distance-varying* potential. Our *plan-based* potential is an implementation of the reward shaping seen in Grzes and Kudenko [2008] where we use the maximum plan step described above as $\phi$. Our *dynamic time-varying* potential is similar to our *plan-based* potential function, but it scales the maximum plan step reward by a factor of $1/t$ where $t$ is the current time step, this incentivizes the agent to achieve the plan as quickly as possible due to the diminishing returns of achieving a subgoal later. In this case:

$$\phi(\hat{s}, t) = (1/t) \times step(\hat{s}) \qquad \text{(dynamic time-varying)}$$

**Algorithm 1:** Automatic Reward Densification

**Input :** (1) set of predicate classifiers $C_\psi$,
(2) $\hat{S}$: a list of high-level subgoals that are encountered during execution of a valid plan,
(3) $R(s)$: the original MDP's reward function,
(4) $s$: the current low-level state,
(5) $t$: the current time-step in the trajectory,
(6) max_idx: the max. index of a state in $\hat{S}$ that's been reached in the current trajectory,
(7) $\phi(\hat{s}, t)$: a choice of potential function

$\hat{s} \leftarrow$ ABSTRACT($s$, $C_\psi$)
$\hat{R}(s) \leftarrow R(s) + \phi(\hat{s}, t) - \phi(\hat{S}[\text{max\_idx}], t-1)$
curr_plan_step $\leftarrow \hat{S}$.index($\hat{s}$) // finds index of $\hat{s}$ in $\hat{S}$
**if** *curr_plan_step* > *max_idx* **then**
  max_idx $\leftarrow$ curr_plan_step
**end**
return $\hat{R}(s)$

We tried other functions of $t$ to scale the reward like $e^{-t}$ or $k/t$ where $k$ is some scalar, but neither performed better than $1/t$. Lastly, our dynamic distance-varying potential tries to use the intuition that much of the movement between subgoals provided by plans for many arm-based robotic tasks is through free space (i.e, task plans will provide waypoints such that the straight-line path between consequent subgoals is not obstructed). We define

$$\phi(\hat{s}) = -dist(\hat{s}, loc(\hat{S}[\text{max\_idx} + 1])), \qquad \text{(dynamic distance-varying)}$$

where $s$ is the current low-level state, $\hat{s}$ is the corresponding abstract state and $loc(\hat{S}[\text{max\_idx + 1}])$ is the location of the end-effector ($ee$) needed to achieve the next subgoal in the plan. During tasks like push where multiple objects are under consideration (each with individual location subgoals) we use the distance between each object in the subgoal separately, trying to achieve each individually with the end-effector given priority. For our pushing environment this looks like this:

$$\phi(\hat{s}) = \begin{cases} -dist(\hat{s}_{ee}, loc(\hat{S}[\text{max\_idx} + 1]_{ee})), & \text{if } ee \text{ is not at subgoal} \\ -dist(\hat{s}_{obj}, loc(\hat{S}[\text{max\_idx} + 1]_{obj})), & \text{otherwise} \end{cases}$$

# 5 Experimental Evaluation

## 5.1 Environment Details

For the purpose of experiments we use the AIRobot Chen et al. [2019] testbed (provided in HW4). We started off with the reaching and pushing environments that were already provided and then extended the reaching environment to a more complex scenario which we call the maze-reach environment as shown in the Figure 2. In the reaching and maze-reach environments, the observation space include $(x_e, y_e)$ position of the end-effector while the push environment additionally contains $(x_o, y_o)$ position of the target object. All environments have 2D continuous action space, $(\Delta x, \Delta y)$. The end-effector can move in the $x$ direction by $\Delta x$, $y$ direction by $\Delta y$. The goal of the reaching and maze-reach environments is to reach a pre-specified goal location and stay there till the end of the episode. The goal of the push environment is to push a target object to a pre-specified goal location.

As part of our experiments, we first ran the baselines using handcrafted sparse and dense rewards similar to the solutions for HW4. Later we ran the *plan-based* potential using PDDL files that created plans going through (i) a single subgoal, (ii) multiple (3) subgoals and (iii) a number of grid locations (after discretizing the state-space into a dense grid). To enable us to run these experiments, we wrote several PDDL models for each of the above cases and used the widely benchmarked Fast Downward planner to generate the high-level symbolic plan. We used these same PDDL models to test both variants of *dynamic plan-based* potentials as (described in Section 4.1). Importantly, note that we actually spent some time experimenting with using a constant multiplier by which to multiply this potential function in order for our agents to achieve good performance.

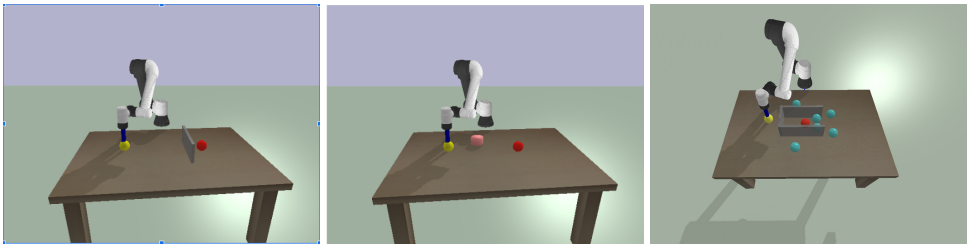

Figure 2: (Left): The reaching environment, (Middle): The pushing environment, (Right): The complex maze-reach environment with a maze to navigate around.

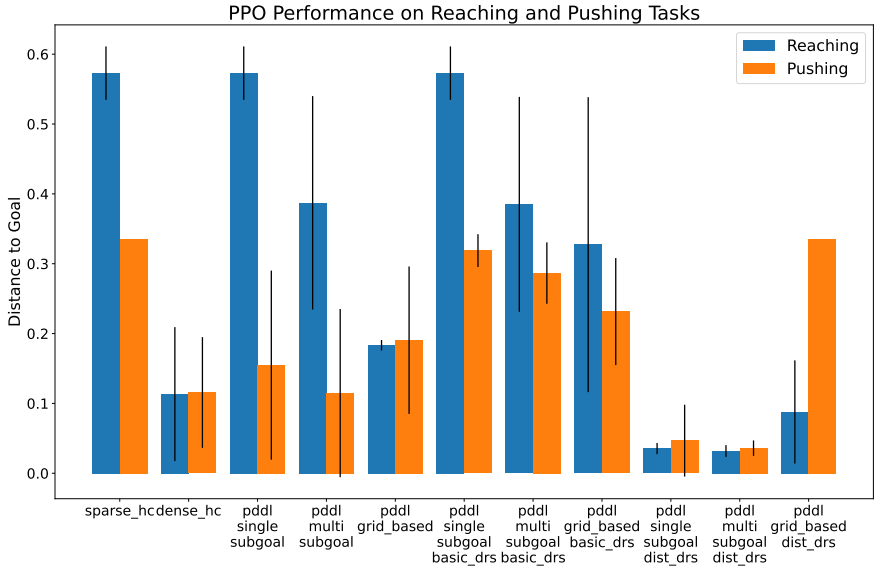


Figure 3: **Our reward shaping methods versus a hand-crafted dense reward function on the reaching and pushing environments using PPO.** Distance to goal of a trained policy after a 25 step episode, for all 3 reward shaping methods with all 3 PDDL models. All results are averaged over 3 seeds where the horizontal black bars denote standard deviations. Here, our multi-goal and grid-based reward shaping performs better than sparse reward and dynamic distance-varying reward shaping performs even better than our dense hand-crafted reward.

We ran each of our experiments with 3 different seeds to see if the performance of our developed approaches can be replicated for different initializations of the algorithms. Additionally, since our proposed approach is agnostic to the choice of learning algorithm, we also decided to use PPO and SAC as implemented in the EasyRL repository Chen [2020]. In total, across the reaching and pushing environments, we ran **132 experiments**. Based on the results of these, discussed in Section 5.2, we ran a limited subset (**12**) of experiments for the final maze-reach environment.

### 5.2 Results

Our main results are shown in Figures 3 and 4. From these, a few trends of note are:

- SAC seems unable to learn good policies for all our densification approaches, even the manually-specified dense reward. We believe this is most-likely due us choosing the default hyperparameters available in EasyRL and these were not good enough for our tasks.
- With PPO, all of our densification approaches find a better policy than the original sparse reward. Moreover, the more granular the PDDL (grid-based > multiple subgoal > single subgoal), the better the performance quantified in terms of distance to goal. This is expected since the more granular the PDDL, the more dense the reward function is and the more likely the agent is going to encounter some positive reward. The glaring exception to this is

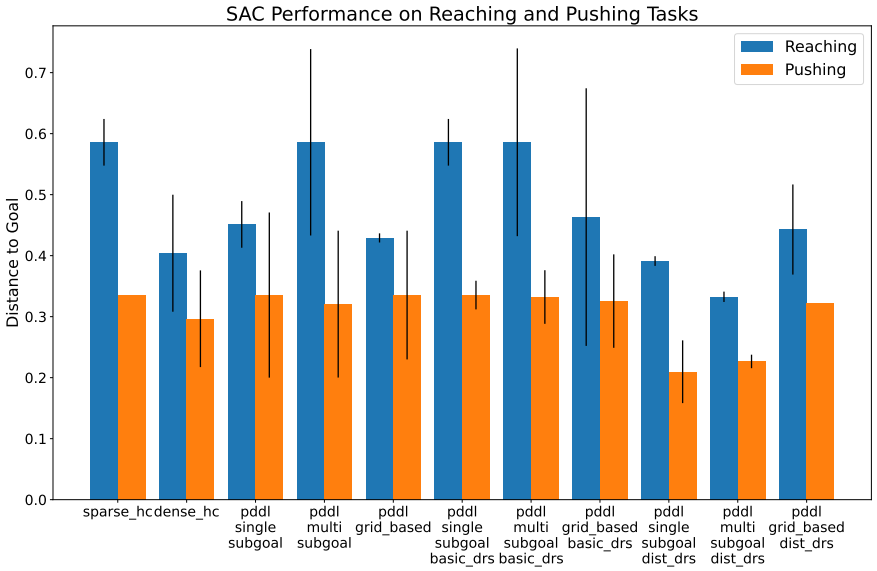


Figure 4: **Our reward shaping methods versus a hand-crafted dense reward function on the reaching and pushing environments using SAC.** Distance to goal for a trained SAC policy after a 25 step episode, for all 3 reward shaping methods with all 3 PDDL models. As you can see, despite results not turning out as well as PPO some of our reward shaping methods perform better than sparse reward and all of the dynamic distance-based reward shaping do as well as our dense hand-crafted reward function.

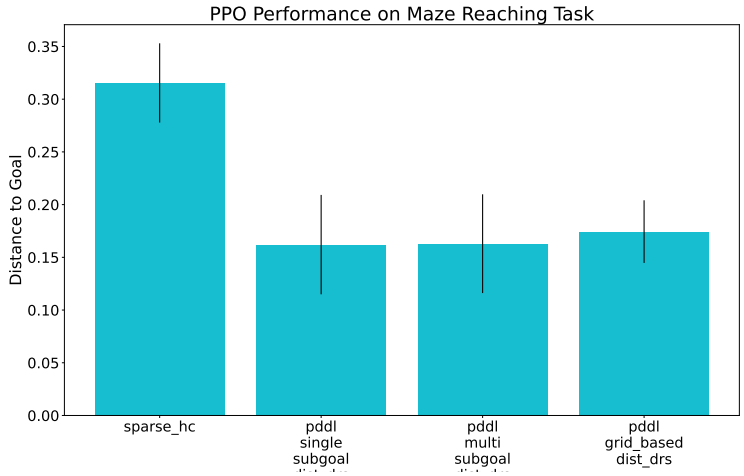


Figure 5: **Our reward shaping methods versus a hand-crafted dense reward function on the maze reaching environment using PPO.** Distance to goal of the block for a trained SAC policy after a 25 step episode, for all 3 reward shaping methods with all 3 PDDL models. Our reward shaping methods perform better than the original sparse reward task, demonstrating our approaches usability on hard to specify reward functions.

the grid_based PDDL for the pushing task. We believe that the issue lies in the fact that the plan involves the agent approaching the block from below, and this often causes it to push the block away from the goal during training.

- The *dynamic time-varying* potential shaping did worse than the *plan-based* potential. This shows that incorporating the $1/t$ factor into the potential function was not useful or even helpful for these tasks.

- Out of all of our approaches, *dynamic distance-varying* potential worked best (with the single subgoal and multiple subgoal PDDLs) and was able to consistently produce policies that solved the task. This makes sense because this approach provides the densest reward out of all our approaches.

Given these trends, we chose to try only our best approach (*dynamic distance-varying* potential with PPO) in the challenging maze-reach task. Importantly, for this task, we did not tune the constant factor used to weight the function, but rather used the same from the reaching environment. Moreover, for the single subgoal and multiple subgoal PDDL cases, this environment happens to violate our free-space motion assumption (i.e, the straight line path between consequent subgoals happens to be obstructed). Our results are displayed in Figure 5.

Considering Figure 5, it seems that the single subgoal and multiple subgoal methods seemed to out perform the higher granularity grid-based method. However, upon closer qualitative inspection, we found that the single subgoal and multiple subgoal polices entered a local minimum on the other-side of an obstructing wall preventing it from reaching the goal, while the grid-based method moved around the obstructing wall and looked qualitatively closer to the optimal policy. While these approaches did enable the agent to learn a better policy than with the sparse handcrafted reward, none of them enabled the agent to learn a policy that actually solves this task.

## 6 Conclusion and Discussion

In summary, we implemented a system that is able to leverage classical planning over human-specified PDDL models to automatically increase the density of robotic tasks with sparse, goal-based reward. We proved that an existing approach (*plan-based* potential) used in discrete environments is theoretically sound, implemented versions of previous approaches, and evaluated their utility to help improve performance on two robotic tasks with continuous state-action spaces and PDDL models with different granularities. Inspired by these approaches, we also developed our own potential-based shaping approach (*dynamic distance-varying* potential) and evaluated its utility across three different robotic tasks. Our results indicate that the granularity of the PDDL model plays an important role in how useful it is, and that while using our shaped reward generally outperforms using only the original sparse reward, outperforming a handcrafted dense reward usually requires a lot of tuning of the PDDL model and weighting assigned to various terms in the potential function.

While our system has shown some promise, there are a variety of important limitations. The biggest of these is that there is still a considerable amount of tuning required to get agents to reliably complete tasks. We might be able to overcome this by combining information from different PDDL models together, utilizing more sophisticated exploration algorithms, or providing other ways for designers to specify useful guiding information to the agent (e.g. natural language corrections to behavior). Additionally, while our methods of shaping do preserve the optimal policy, they may cause the agent to fall into sub-optimal local return maxima because of a sub-optimal plan.

## 7 Contributions

**Nishanth**: I contributed the idea of trying out a time-varying potential function. I also came up with the idea to re-use the environments from HW4 of the class. I contributed heavily to the implementation of our ideas, particularly the architecture of our repository that enabled us to easily run particular collections of experiments. Finally, I helped prove the theorem in Section 8.1 and also helped write several sections of this paper.

**Willie**: I helped decide the direction of the project and possible environments we could use to evaluate our ideas. I also contributed heavily to the implementation of our project, particularly the implementation of our three potential based reward functions and the creation of scripts to run our experiments. Finally, I proved the theorem in Section 8.1 and wrote several sections of this paper.

**Viraj**: I contributed heavily to the codebase especially towards designing the PDDL models and automating the grid-based state generation process. I also worked on developing the complex maze-reach environment. After looking at the performance of the simple and dynamic plan-based potential functions, I proposed the idea of densifying the potential space using distance functions. I also made the website to showcase all the videos and finally I helped write several sections of this paper as well.

# 8 Appendix

## 8.1 Proof that optimal policy is preserved by plan-based potential functions

Suppose our chosen environment is modeled by an MDP $M = \langle S, A, R, T, \gamma \rangle$. Let us define a new MDP $M^+ = \langle S^+, A, R^+, T^+, \gamma \rangle$. Here, a state $s_i^+ \in S^+$ is composed of a corresponding state $s$ from $M$ and a variable $n_i$ that represents the most recent plan step the agent has achieved: $s_i^+ = \langle s_i, n_i \rangle$. The reward function $R^+(s_i^+) = R^+(\langle s_i, n_i \rangle) = R(s_i) : \forall s_i^+ \in S^+$, and transition function $T^+(s_i^+, a_i) = T^+(\langle s_i, n_i \rangle, a_i) = \langle T(s_i, a_i), \chi(s_i, n_i) \rangle = s_{i+1}^+ : \forall s_i^+ \in S^+ \wedge a_i \in A$ where $\chi$ is a function describing how $n_i$ changes between transitions (i.e. whether we successfully accomplished the subgoal in our plan or not). Given these, we can define a return $G$ for a policy $\pi$:

$$G_\pi = \mathbb{E}_{s_0 \in S}[R(s_0) + R(T(s_0, a_0)) + R(T(s_1, a_1)) + \ldots] \text{ and } \pi(s_i, a_i) = a_{i+1} \tag{1}$$

and similarly $G$ given $\pi^+$:

$$G_{\pi^+} = \mathbb{E}_{s_0^+ \in S^+}[R^+(s_0^+) + R^+(T^+(s_0^+, a_0)) + \ldots] \text{ where } \pi^+(s_i^+, a_i) = a_{i+1} \tag{2}$$

We want to show $\forall G_{\pi^+}$, $G_{\pi^+} = G_\pi$ where $\pi(s_i) = \pi^+(\langle s_i, n_i \rangle) = a_{i+1}$ and $\pi^+(\langle s_i, n_i \rangle) = \pi^+(\langle s_i, n_j \rangle) = \pi^+(s_i) \; \forall \; n_i, n_j \in N$

We know by definition:

$$\begin{aligned} G_{\pi^+} &= \mathbb{E}_{s_0^+ \in S^+}[R^+(s_0^+) + R^+(T^+(s_0^+, a_0)) + R^+(T^+(s_1^+, a_1)) + \ldots] \\ &\implies G_{\pi^+} = \mathbb{E}_{s_0^+ \in S^+}[R^+(\langle s_0, n_0 \rangle) + R^+(T^+(\langle s_0, n_0 \rangle, a_0)) + \ldots] \end{aligned}$$

Now, if we substitute our definition of the transition function $T^+$ into this above equation, we obtain:

$$G_{\pi^+} = \mathbb{E}_{s_0^+ \in S^+}[R^+(\langle s_0, n_0 \rangle) + R^+(\langle s_1, \Psi(s_0, n_0) \rangle) + \ldots]$$

given our definition of transition function $T^+$ and thus, we can substitute in our definition for $R^+$ to obtain:

$$G_{\pi^+} = \mathbb{E}_{s_0^+ \in S^+}[R(s_0) + R(T(s_0, a_0)) + R(T(s_1, a_1)) + \ldots] = G_\pi \tag{3}$$

Therefore we know that our optimal policy is preserved in our new MDP because for all policies return is preserved. [3]

# References


Pulkit Agrawal. The task specification problem. In *5th Annual Conference on Robot Learning, Blue Sky Submission Track*, 2021. URL `https://openreview.net/forum?id=cBdnThrYkV7`.

Tao Chen. EasyRL. `https://github.com/taochenshh/easyrl`, 2020.

Tao Chen, Anthony Simeonov, and Pulkit Agrawal. AIRobot. `https://github.com/Improbable-AI/airobot`, 2019.

Jack Clark and Dario Amodei. Faulty reward functions in the wild. *Internet: https://blog. openai. com/faulty-reward-functions*, 2016.

Sam Michael Devlin and Daniel Kudenko. Dynamic potential-based reward shaping. In *Proceedings of the 11th international conference on autonomous agents and multiagent systems*, pages 433–440. IFAAMAS, 2012.

Clement Gehring, Masataro Asai, Rohan Chitnis, Tom Silver, Leslie Pack Kaelbling, Shirin Sohrabi, and Michael Katz. Reinforcement learning for classical planning: Viewing heuristics as dense reward generators. *CoRR*, abs/2109.14830, 2021. URL `https://arxiv.org/abs/2109.14830`.


[3] Note that in the last step we take an expectation over ${s_0}^+ \in S^+$ not $s_0 \in S$, really it should be an expectation on the set of initial states which would be the same size where ${s_0}^+ = \langle s_0, 0 \rangle, \forall {s_0}^+ \in S^+$ and $\forall s_0 \in S^+$.

Marek Grzes and Daniel Kudenko. Plan-based reward shaping for reinforcement learning. In *2008 4th International IEEE Conference Intelligent Systems*, volume 2, pages 10–22–10–29, 2008. doi: 10.1109/IS.2008.4670492.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL https://arxiv.org/abs/1801.01290.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246, 2006.

Yuqian Jiang, Sudarshanan Bharadwaj, Bo Wu, Rishi Shah, Ufuk Topcu, and Peter Stone. Temporal-logic-based reward shaping for continuing learning tasks. *CoRR*, abs/2007.01498, 2020. URL https://arxiv.org/abs/2007.01498.

Drew McDermott, Malik Ghallab, Adele E. Howe, Craig A. Knoblock, Ashwin Ram, Manuela M. Veloso, Daniel S. Weld, and David E. Wilkins. Pddl-the planning domain definition language. 1998.

Farzan Memarian, Wonjoon Goo, Rudolf Lioutikov, Ufuk Topcu, and Scott Niekum. Self-supervised online reward shaping in sparse-reward environments. *CoRR*, abs/2103.04529, 2021. URL https://arxiv.org/abs/2103.04529.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208, Jan 2022. ISSN 1076-9757. doi: 10.1613/jair.1.12440. URL http://dx.doi.org/10.1613/jair.1.12440.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2019. URL https://arxiv.org/abs/1910.10897.

Haosheng Zou, Tongzheng Ren, Dong Yan, Hang Su, and Jun Zhu. Reward shaping via meta-learning, 2019. URL https://arxiv.org/abs/1901.09330.