Goal Modelling for Deep Reinforcement Learning Agents

Jonathan Leung ✉, Zhiqi Shen, Zhiwei Zeng, and Chunyan Miao

School of Computer Science and Engineering, Nanyang Technological University,
50 Nanyang Avenue, 639798 Singapore
jonathan008@e.ntu.edu.sg, {zqshen, zhiwei.zeng, ascymiao}@ntu.edu.sg

Abstract. Goals provide a high-level abstraction of an agent's objectives and guide its behavior in complex environments. As agents become more intelligent, it is necessary to ensure that the agent's goals are aligned with the goals of the agent designers to avoid unexpected or unwanted agent behavior. In this work, we propose using Goal Net, a goal-oriented agent modelling methodology, as a way for agent designers to incorporate their prior knowledge regarding the subgoals an agent needs to achieve in order to accomplish an overall goal. This knowledge is used to guide the agent's learning process to train it to achieve goals in dynamic environments where its goal may change between episodes. We propose a model that integrates a Goal Net model and hierarchical reinforcement learning. A high-level goal selection policy selects goals according to a given Goal Net model and a low-level action selection policy selects actions based on the selected goal, both of which use deep neural networks to enable learning in complex, high-dimensional environments. The experiments demonstrate that our method is more sample efficient and can obtain higher average rewards than other related methods that incorporate prior human knowledge in similar ways.

Keywords: Deep Reinforcement Learning · Hierarchical Reinforcement Learning · Goal Modelling

1 Introduction

Deep reinforcement learning (DRL) has enabled agents to achieve human-level, and in some cases superhuman-level, results in complex, high-dimensional environments. In many applications, agents are required to achieve multiple goals in complex environments. However, many DRL methods are limited in that they can only complete one task or goal. Kaelbling [13] proposed a method to train reinforcement learning agents to learn to achieve a wide variety of goals. This work forms the basis of recent goal-conditioned and multi-goal reinforcement learning methods that make use of deep neural networks [23].

As agents become more intelligent, it is necessary to ensure that the agent's goals are aligned with the goals of the agent designers, which has been referred to as the agent alignment problem [16]. Although many recent deep learning

methods reduce the amount of prior knowledge given to models to improve performance, the inclusion of such knowledge may improve agent alignment by providing more context to the agent. One way to leverage prior human knowledge in RL would be to model goals so that they can be understood and specified by agent developers and designers, regardless of their technical knowledge. Goal models, which originate from Goal-Oriented Requirements Engineering (GORE), can provide a way for agent designers to express the high-level behavior that they desire from their agents. GORE focuses on goals as a way to define system objectives and to communicate the rationale behind system requirements to stakeholders of varying technical knowledge [31]. In GORE, goal models have been used in agent design to support formal representation and reasoning with goals [4, 33]. Goal models define goals and capture the relationships between them, such as AND/OR relationships between subgoals that conjunctively/disjunctively achieve a high-level goal.

In this work, we propose a hierarchical reinforcement learning (HRL) model that incorporates an agent designer's prior knowledge about an agent's overall goal within a Goal Net model. Goal Net is an agent modelling methodology that uses goal modelling to define agent behavior [27]. Unlike other goal models that only specify the decomposition of goals into subgoals, Goal Net allows agent designers to specify the sequential relationships between goals to allow agents to reason about goals at run time, which makes it a suitable choice for our work. Our model consists of a high-level goal selection policy that provides goals to a low-level action selection policy, as shown in Figure 1. Given a high-level goal, an agent may select subgoals to achieve the goal which in turn affects the agent's actions. A Goal Net model is used to provide valid goal selection options to the high-level policy, and the low-level policy is a goal-conditioned policy that operates in a goal-augmented state space which incorporates a symbolic goal space. We propose an algorithm that trains a hierarchical Deep Q-Network (h-DQN) [15] combined with a Goal Net model. Then, we evaluate our model against other related methods in which Goal Net could be incorporated, namely deep abstract Q-networks (DAQN) [25] and reward machines (RM) [12]. The results suggest that our method is more sample efficient and can achieve higher average rewards in environments with randomized goal locations.

2 Background

Reinforcement Learning (RL) aims to train an agent to act optimally within an environment [28]. This problem is typically formulated as a Markov decision process (MDP), which is defined as a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}(s'|s, a)$ is a transition probability function, $r(s, a, s')$ is a reward function, and γ is a discount factor. At a time step t, an agent observes a state $s_t \in \mathcal{S}$ and takes an action $a_t \in \mathcal{A}$. After the action is executed, the agent observes a new state s_{t+1} and receives rewards r_{t+1} according to the reward function. The goal of the agent is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the rewards the agent obtains while interacting with the environment.

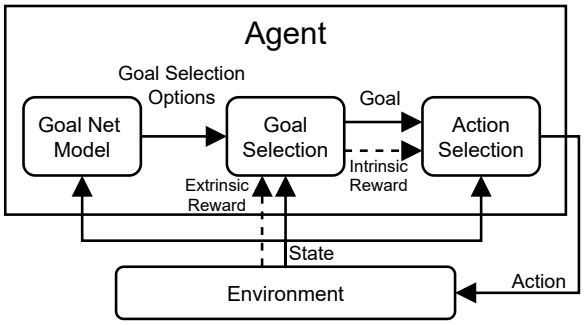


Fig. 1: An overview of the proposed model. The goal selection and action selection, in this work, are trained through reinforcement learning.

Many RL algorithms make use of value functions in order to learn the optimal policy. The Q-value function measures the expected future discounted rewards an agent can obtain by taking an action in a given state, and is defined as:

$$Q(s,a) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_{t+1} | s_0 = s, a_0 = a]. \tag{1}$$

Deep Q-Networks (DQN) use deep neural networks to estimate the Q-value function [20]. DQN uses an experience replay buffer [18] that stores tuples containing information such as the states and actions the agent experiences while interacting with the environment. Experience tuples are sampled from the replay buffer to train the network, which enables data reuse and stabilizes the learning process.

Hierarchical Reinforcement Learning (HRL) involves training an agent to use multiple levels of policies where higher level policies may invoke or direct lower level policies to achieve subgoals. The options framework is a commonly used HRL formalism in which a high-level policy may use a temporally extended option, or macro-action, instead of a primitive action [29]. The framework makes use of the semi-Markov decision process (SMDP) that generalizes MDPs to the settings where actions may take a varying number of timesteps [24]. An option is a tuple $\langle \mathcal{I}_o, \pi_o, \beta_o \rangle$ where $\mathcal{I}_o \subseteq \mathcal{S}$ is an initiation set describing in which states the option can be invoked, $\pi_o : \mathcal{S} \to \mathcal{A}$ is an intra-option policy, and $\beta_o : \mathcal{S} \to [0, 1]$ is a termination function indicating when the option ends.

Goal-Conditioned Reinforcement Learning trains agents to learn a value function parametrized by the agent's goal g, which generalizes learning experience in achieving one goal to other goals [13]. Universal Value Function Approximators (UVFAs) make use of function approximators such as deep neural networks to enable generalization to new goals unseen at training time [26]. Hindsight Experience Replay (HER) [2] improves sample efficiency by adding relabelled experience tuples to the replay buffer. The relabelling process replaces the agent's original goal with the goal the agent actually reaches. Such works have

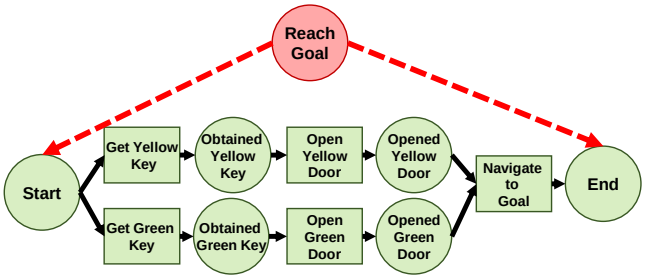


Fig. 2: An example Goal Net for an agent trying to reach an end goal.

given rise to multi-goal RL [23], which trains agents to achieve a wide variety of goals.

Goal Net is a graphical model that defines agents' goals, subgoals, and the relationships between those goals [27]. Agent development using Goal Net involves the co-operation of agent designers who may be domain experts that can define high-level behavior and logic of an agent, and agent developers who have the development skills to implement the functions required by the agent. A Goal Net consists of goals and actions, which are represented graphically by circles and rectangles, respectively. Actions represent the transitions between goals and define any tasks that need to be completed in order to reach a goal. Goals can be composite, meaning that they can be decomposed into more goals, or atomic. Composite and atomic goals are represented as red and green circles, respectively. An example Goal Net is shown in Figure 2, which shows a Goal Net for an agent attempting to reach an end goal by either obtaining a green key and opening a green door, or by using a yellow key to open a yellow door. We denote the set of goals within a Goal Net as G_{net}. Each Goal Net contains a root composite goal that indicates the overall goal to be achieved, a start goal, and an end goal. Arcs connect goals and actions together, indicating valid paths the agent may take to achieve the overall goal. At run time, the agent begins in the start goal, and uses goal selection algorithms to determine which goal to pursue and action selection algorithms to decide how goals should be achieved. The agent transitions to the next goal if it successfully achieves it.

Goal Net can also represent and define concurrent goal pursuit. A concurrency relation between goals represents a partially ordered goal achievement requirement where all goals in the concurrency relationship must be achieved. Concurrent goal paths will synchronize at a goal or action, which represents the point at which all paths must reach before transitioning to the next goal. Graphically, concurrent goals are represented using diamond-shaped arcs. Figure 3 shows an example Goal Net that contains a concurrent goal relation where the agent must reach both a yellow and blue subgoal before navigating to the final goal state, but the order in which the subgoals are reached does not matter.

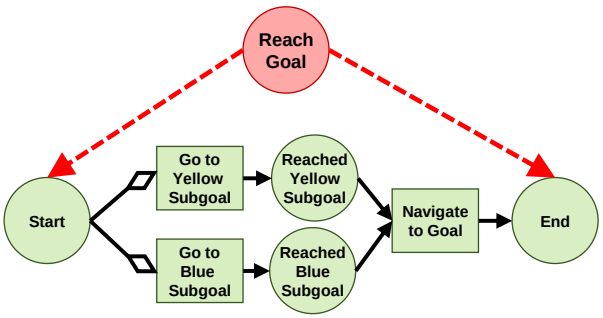


Fig. 3: Concurrent goal pursuit represented by diamond-shaped arcs, indicating that the agent must complete both subgoals before reaching the final goal.

3 Deep Reinforcement Learning with Goal Net

In this work, we utilize Goal Nets as models to define high-level agent behavior, which may be provided by agent designers. We consider the case where the goal and action selection within a Goal Net model are learned using reinforcement learning. This may be desirable when the environment is complex, or to reduce the workload of developers so that the goal and action selection algorithms do not need to be hand-engineered. We treat this setting as a hierarchical reinforcement learning problem with two policy levels: a high-level goal selection policy and a low-level action selection policy. Our hierarchical structure is based on the options framework, as well as h-DQN which trains two DQNs: a low-level controller and a high-level meta-controller [15].

In addition to the Goal Net model, we require agent designers and developers to create a goal space $\mathcal{G}$ that consists of symbolic attributes related to the goals and subgoals of the agent. For example, in a goal reaching task where an agent needs to reach a given position in a coordinate space, an agent designer may define the goal space as the agent's current coordinates. We will refer to goals within the Goal Net model as $g_{net} \in G_{net}$ to differentiate between points in the goal space $g \in \mathcal{G}$ and the Goal Net goals. Referring back to Figure 1, a Goal Net goal g_{net} is passed to the goal selection policy and is used to select a target goal g'_{net}, and then this is converted to goal space $\mathcal{G}$. Similar to other related methods such as DAQN and QRM, this conversion is performed by a labelling function $\mathcal{F} : \mathcal{S} \to \mathcal{G}$, which we assume to be given by agent developers. The goal space allows us to take advantage of goal-conditioning and HER by augmenting the state space, inducing a state space $\mathcal{S}_{lo} = \mathcal{S} \times \mathcal{G}$. The low-level policy operates in this goal-augmented state space and is therefore defined as $\pi_{lo} : \mathcal{S}_{lo} \to \mathcal{A}$. The low-level policy selects actions using a goal-conditioned DQN that is trained to estimate the optimal Q-value function:

$$Q^*_{lo}(s_{lo}, a, g) = r_i + \gamma \sum_{s'_{lo} \in \mathcal{S}_{lo}} \mathcal{P}(s'_{lo}|s_{lo}, a, g) \max_{a'} Q^*_{lo}(s'_{lo}, a', g), \tag{2}$$

where $g \in \mathcal{G}$ is a subgoal to achieve and $r_i \in \{0, 1\}$ is an intrinsic reward of 1 if the low-level policy reaches a given subgoal and 0 otherwise.

We denote the high-level goal selection policy as π_{hi}, which operates in the state space $\mathcal{S}_{hi} = \mathcal{S} \times G_{net}$ and can be defined as $\pi_{hi} : \mathcal{S}_{hi} \rightarrow G_{net}$. The goal selection policy selects the next Goal Net goal to target and does not need to select goals directly in goal space. A goal achievement function $\beta_g : \mathcal{S}_{hi} \rightarrow \mathcal{G}$ is used to generate the terminal conditions within goal space. This helps increase the training speed of the goal selection policy since invalid and unused goals in the goal space are pruned. The goal selection policy operates within a SMDP, and its associated DQN is trained to estimate the optimal Q-value function:

$$Q^*_{hi}(s_{hi}, g_{net}) = \sum_{s'_{hi}, \tau} \mathcal{P}(s'_{hi}, \tau | s_{hi}, g_{net})[R_\tau + \gamma^\tau \max_{g'_{net}} Q^*_{hi}(s'_{hi}, g'_{net})], \quad (3)$$

where $s_{hi} \in \mathcal{S}_{hi}$, τ is the number of timesteps taken by the low-level policy to complete the subgoal, and $R_\tau = \sum_{t=0}^{\tau} \gamma^t r_{t+1}$. The extrinsic rewards from the environment obtained while running the low-level policy are passed to the high-level policy.

Algorithm 1 shows the overall training procedure for our model. Line 6 is the start of the goal selection loop, and in lines 7-8 the next Goal Net goal for the agent to achieve is selected and converted to goal space. In lines 11-12, the action selection policy is used to select actions based on the target goal. Both the goal and action selection use ϵ-greedy style exploration strategies based on the exploration strategy used in h-DQN. Such strategies select a random action with probability ϵ, and select the action with the maximum Q-value otherwise. A common strategy to enable sufficient exploration is to initialize ϵ with a high value and to decay it, typically linearly, over the course of the training process. However, in complex, temporally extended problems, it is difficult to pick a decay rate that ensures that the agent adequately explores the environment. Therefore, we take advantage of the Goal Net model to determine the exploration rates for the action and goal selection policies. The exploration rate of the action selection policy scales according to the success rate of achieving the selected Goal Net goal:

$$\epsilon_{lo} = -(\epsilon_{max} - \epsilon_{min})(\text{success}_N(g_{net}, g'_{net})) + \epsilon_{max}, \quad (4)$$

where ϵ_{max} and ϵ_{min} are hyperparameters defining the maximum and minimum values of ϵ_{lo}, respectively, and $\text{success}_N(g_{net}, g'_{net})$ is the average success rate of achieving goal g'_{net} starting from g_{net} over the past N attempts at achieving the goal. The exploration rate for goal selection follows a standard ϵ-greedy strategy, but instead of randomly selecting a goal with probability ϵ_{hi}, we weigh the probability of selecting particular goals based on the success rate of achieving them. The formula for determining the probability of selecting a goal is:

$$p(g_{net}, g'_{net}) = \frac{1 - \text{success}_N(g_{net}, g'_{net}) + \rho}{\sum_{g^*_{net} \in G^*_{net}} 1 - \text{success}_N(g_{net}, g^*_{net}) + \rho}, \quad (5)$$

where $\rho < 1$ is a small number that ensures that all goals have a chance to be selected and to prevent any division by 0, and G^*_{net} is the set of goals that

Algorithm 1: h-DQN Training with Goal Net

Input : Goal Net Model

1 Initialize DQNs Q_{lo}, Q_{hi}
2 Initialize experience replay buffers $\mathcal{R}_{lo}, \mathcal{R}_{hi}$
3 **for** $i = 0$ **to** `num_episodes` **do**
4 $\quad g_{net} \leftarrow$ initial Goal Net goal
5 $\quad s, s_{hi}, s_{lo}, g \leftarrow$ environment reset
6 $\quad$ **for** $j = 0$ **to** `max_steps` **do**
7 $\quad\quad g'_{net} \leftarrow \text{SelectGoal}(s_{hi})$
8 $\quad\quad g_{target} \leftarrow \beta_g(s, g'_{net})$
9 $\quad\quad r_{total} \leftarrow 0, \texttt{steps} \leftarrow 0$
10 $\quad\quad$ **for** $k = j$ **to** max_steps **do**
11 $\quad\quad\quad a \leftarrow \text{SelectAction}(s_{lo}, g_{target})$
12 $\quad\quad\quad s', g', r, \texttt{done} \leftarrow \text{ExecuteAction}(a)$
13 $\quad\quad\quad r_{total} \leftarrow r_{total} + r$
14 $\quad\quad\quad g^*_{net} \leftarrow \text{GNetReached}(s', g', g_{net})$
15 $\quad\quad\quad \texttt{done}_{lo} \leftarrow (g^*_{net} \mathrel{!}= g_{net})$ or `done`
16 $\quad\quad\quad$ **if** $g^*_{net} == g'_{net}$ **then**
17 $\quad\quad\quad\quad r_i \leftarrow 1$
18 $\quad\quad\quad$ **else**
19 $\quad\quad\quad\quad r_i \leftarrow 0$
20 $\quad\quad\quad \text{Add } \langle s_{lo}, a, g_{target}, (s', g'), r_i, \texttt{done}_{lo} \rangle \text{ to } \mathcal{R}_{lo}$
21 $\quad\quad\quad \text{Update } Q_{lo} \text{ using } \mathcal{R}_{lo}$
22 $\quad\quad\quad s_{lo} \leftarrow (s', g')$
23 $\quad\quad\quad \texttt{steps} \leftarrow \texttt{steps} + 1$
24 $\quad\quad\quad$ **if** $\texttt{done}_{lo}$ **then**
25 $\quad\quad\quad\quad$ **break**
26 $\quad\quad$ **end**
27 $\quad\quad \text{Add relabelled experience tuples to } \mathcal{R}_{lo}, \text{ replacing } g_{target} \text{ with } g'$
28 $\quad\quad \text{Add } \langle s_{hi}, g^*_{net}, (s', g^*_{net}), r_{total}, \texttt{done}, \texttt{steps} \rangle \text{ to } \mathcal{R}_{hi}$
29 $\quad\quad \text{Update } Q_{hi} \text{ using } \mathcal{R}_{hi}$
30 $\quad\quad s_{hi} \leftarrow (s', g^*_{net})$
31 $\quad\quad$ **if** `done` **then**
32 $\quad\quad\quad$ **break**
33 $\quad$ **end**
34 **end**

can be selected from g_{net} as defined by the Goal Net model. By using this goal exploration strategy, we attempt to ensure that the agent learns to achieve all goals by focusing on goals that the agent cannot reach consistently.

During training, it is likely that the action selection policy inadvertently achieves a different goal than the goal proposed by the goal selection policy. For example, an agent using the Goal Net in Figure 2 may obtain the yellow key even though it tried to obtain the green key. Line 14 checks which Goal Net goal the agent has reached by comparing g' with $\beta_g(s, g'_{net})$ across all possible Goal Net goals reachable from g_{net}. An intrinsic reward is provided to the low-level

policy if it reaches the proposed goal, and the loop breaks if the low-level policy transitions to a new Goal Net goal. In line 28, we add experience tuples to the high-level replay buffer using the Goal Net goal actually reached by the agent rather than the one proposed by the high-level policy.

To handle concurrent goals, Algorithm 1 uses a list of goal paths in the Goal Net model containing the goals the agent has currently reached. Then the available goals that can be selected consists of all possible goal selection options across all goal paths.

4 Experiments

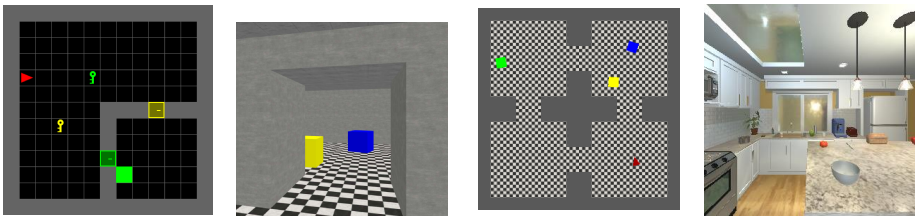
In the experiments[1], we make use of the Minigrid environment [6], Miniworld environment [5], and AI2-THOR [14]. Some example images of the environments used are shown in Figure 4. More details about each environment are provided in the following subsections. The extrinsic reward function used in our experiments is based on the default reward function provided by Minigrid, defined as:

$$R = 1 - 0.9\Big(\frac{n_{steps}}{n_{max}}\Big), \tag{6}$$

where n_{steps} is the number of steps taken by the agent to reach the goal, and n_{max} is the maximum episode length. We use this reward function as it integrates both the agent's success rate and steps taken to reach the goal.

The goal of the experiments is to compare our method with other related methods in which a Goal Net model could be incorporated and to highlight problems that they have. We compare 4 different models: the proposed model, a variant of our model where the low-level policy operates on the state space without goal-augmentation, a model based on DAQN, and a model based on Q-learning for Reward Machines (QRM). We will refer to these models as GNet, GNet without GA, DAQN, and QRM respectively. GNet without GA will be used to compare whether using a goal-augmented state space for the low-level policy provides any benefits. We use DAQN and QRM as comparisons as both methods provide ways for agent designers to provide knowledge to an agent so that they can achieve temporally extended goals in a similar manner as our proposed method. DAQN is a HRL method where the high-level policy operates in an abstract state space defined by an agent designer and may invoke a low-level policy that is trained to reach a specific abstract state. The high-level policy in the DAQN model uses a tabular Q-learning algorithm that selects goals according to the provided Goal Net model. To make comparisons fairer, we also provide the valid goal selection choices based on the Goal Net model to the high-level policy. Additionally, we use goal-conditioning on the low-level DAQN policy by providing the Cartesian coordinates of the target goal and use HER to relabel experience tuples using the coordinate reached by the agent. QRM provides a comparison to a flat, non-goal-conditioned model capable of training an agent

[1] Code available at: `https://github.com/jleung1/goal_modelling_rl`



(a) Two keys environment. (b) 3D four rooms environment. (c) Top view of the 3D four rooms environment. (d) An example kitchen from AI2-THOR.

Fig. 4: Sample images of the experimental environments. Positions of objects, subgoals, and the agent are randomized each episode.

to complete temporally extended tasks by having agent designers define a finite state machine that represents the reward function. The reward machines used in our experiments are based on the Goal Net models used in each experiment, where in most cases each goal acts as a reward machine state that provides the agent with a reward of 1. The transitions between reward machine states are determined by the goal spaces and β_g used in each experiment.

We attempt to use similar neural network architectures for all models. The low-level DQNs of the hierarchical models and the QRM model use convolutional layers that take the environment state as input. In GNet, the goal state and target goal are concatenated with the output of the convolutional layers and then passed through a set of linear layers. Only the target goal is used in GNet without GA. In DAQN and QRM, we pass one-hot encodings to the network to differentiate between multiple policies as opposed to the multi-headed or separate networks used in the respective works. We use Double DQN [11] for all models, as was done in both original DAQN and QRM works.

4.1 Two Keys

The first experimental environment, which we will refer to as the "two keys" environment, was created using Minigrid. In this environment, shown in Figure 4a, the agent must reach the green goal coordinate. To do this, the agent needs to learn to acquire either the yellow or green key, then open the corresponding door to reach the goal room located in the bottom right quadrant. This environment tests the agent's ability to choose the key and door that lead it to the goal the fastest. Every episode, the agent and the keys are positioned randomly outside of the goal room, and the goal's position within the goal room is also randomized. In addition, the positions of the two doors may be randomly swapped. The actions available to the agent include moving forward, turning left or right, picking up a key, and opening a door. The Goal Net model used in this experiment is shown in Figure 2. The goal space consists of the x and y coordinates of the agent's target goal, followed by a set of propositional symbolic features indicating whether

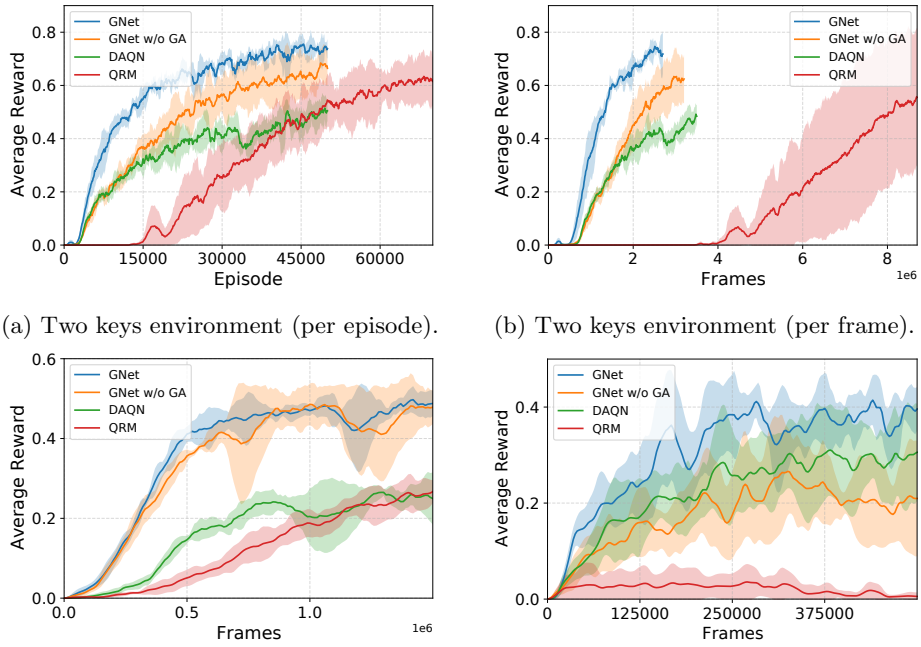
the agent has the yellow or green key, and whether the yellow and green doors are opened or closed. We use a symbolic state space provided by the Minigrid environment, which is a $3\times13\times13$ grid that contains information about the object type, color, and status at each grid coordinate.

We train each agent for 50000 episodes, except for QRM which needed more training to converge, with a maximum episode length of 300 steps. We perform 100 evaluation episodes every 100 training episodes where the agent takes the greedy action at each time step. This process is repeated 5 times using the same set of random seeds across all models, and the means and standard deviations of the average rewards obtained are reported. The results are shown in Figure 5a, and the rewards per frame are shown in Figure 5b to illustrate the difference in sample efficiency between QRM and the other methods.

GNet is able to learn from the transitions within the goal-augmented state space and relate it to the target goals used in training a goal-conditioned DQN for the low-level policy. Since the state space used in GNet without GA does not directly include the goal space, the agent needs to learn the associations between the target goal and the state space itself, and thus is slightly less sample efficient. It should be noted that the goal space in this experiment contains information that is readily available within the state space given to the agent. This showcases the potential of providing an agent with a simplified representation of the state space alongside the full state space to improve learning efficiency.

DAQN converges to a lower average reward value because the abstract goal space does not provide enough information to the high-level policy about which key it should obtain. This problem was discussed by Gopalan et al. [10] and is demonstrated by this experiment. The agent cannot differentiate between obtaining the yellow of green key because both options lead the agent to the goal in the same number of steps with respect to the high-level policy. In order for DAQN to find a better policy, a goal space containing more information about the state space would be required. In contrast, the GNet models use the full state space at both policy levels and thus are more robust to the goal space design, imposing fewer restrictions on agent designers.

QRM can learn a policy that converges to higher rewards than DAQN, however it takes much longer to learn since it does not use HER. Whereas QRM only uses the goal space to determine reward machine state transitions, GNet allows the agent to actively select its next goal, which allows the use of HER. As QRM does not use a hierarchy, it is not clear how a goal selection mechanism would be incorporated into the method to allow for the use of goal-conditioning and HER. As described by Icarte et al. [12], QRM can share learning experience between RM state Q-value functions by using the RM to determine whether any RM state transitions have occurred when the agent interacts with the environment. However, this does not aid the agent in this environment because each RM state corresponds to separate sets of environment states. For example, if the agent opens the yellow door and then reaches the goal, this experience cannot be transferred to the case where the agent opens the green door and reaches the goal because the environment state is different in both cases.



(a) Two keys environment (per episode).

(b) Two keys environment (per frame).

(c) 3D four rooms with subgoals.

(d) AI2-THOR kitchen environment.

Fig. 5: Average rewards obtained across all runs of the experiments.

4.2 3D Four Rooms with Subgoals

In this experiment, we modify an implementation of a 3D version of the four rooms environment provided by Miniworld [5]. The goal of the agent is to visit a blue and yellow subgoal before reaching the green goal, whose positions are all randomized at each episode. The order in which the agent visits the subgoals does not matter, and so this environment tests the agent's ability to handle partially ordered subgoals in a high-dimensional 3D environment with much randomness. If the agent reaches the final goal before reaching both subgoals, the agent receives a reward of 0. The agent views the environment in a first person perspective and receives RGB images as state observations, as shown in Figure 4b. We provide the agent with the previous 4 frames to help the agent handle partial observability, making the size of the agent's observation $4\times3\times60\times80$. An overhead view of the environment is shown in Figure 4c. The Goal Net model used for this experiment is shown in Figure 3, however the equivalent RM contains an extra state that represents the agent having reached both subgoals. The agent can turn left or right by a random amount between 10 and 30 degrees and move forward.

The goal space used in this experiment consists of the x and z positions of the goal, subgoals, and the agent, as well as whether the agent has reached the blue and yellow subgoals. To make comparisons fairer, we provide the goal,

subgoal, and agent coordinates as additional state information to GNet without GA, DAQN, QRM. We use a deeper neural network for this experiment based on the one used by Espeholt et al., which uses residual connections [9]. We perform our evaluation similarly to the previous experiment where we run 100 evaluation episodes after every 100 training episodes and repeat the training process 5 times. We use a maximum episode length of 300 steps and use an ϵ of 0.05 during evaluation episodes. The results are shown in Figure 5c.

Both GNet and GNet without GA perform similarly, with a bit more instability in GNet without GA. Since we provided the agent coordinates, subgoal, and goal locations to all methods, the only difference between GNet and GNet without GA is the inclusion of the subgoal completion statuses within the goal-augmented state space. However, as will be shown in the next experiment, this small difference can have a larger impact on the agent's performance in some environments. As in the two keys environment, DAQN performs worse because the high-level policy cannot determine whether visiting the yellow or blue subgoal first is better. Exploration in this experiment is easier than the two keys environment, as there are only three movement actions, making QRM learn quicker than in the previous experiment.

4.3 Kitchen Navigation and Interaction

In this experiment we use AI2-THOR, a 3D home environment created in the Unity game engine [14]. AI2-THOR provides various rooms where agents can interact with various objects. For this experiment, we use the 30 different kitchen environments provided by AI2-THOR and train the agent to first close the fridge door, and then turn off the light switch. An example of one of the kitchens is shown in Figure 4d. The actions available to the agent are turning left and right, moving forward, closing an object, and toggling off an object. The episode ends when the agent turns off the light, with a reward of 0 being given if it turns off the light before closing the fridge door. This experiment tests the agent's use of the goal space to learn to navigate and complete tasks in high-dimensional environments that vary greatly between episodes. Since the sequence of subgoals is always the same in this experiment, the high-level policies of the hierarchical models do not need to be trained, which allows the low-level policy to be isolated and analyzed. The neural network architecture is similar to the previous experiment, using a deeper model with residual connections. The state observations given to the agent consists of the last four 100×100 RGB-depth images. The goal space consists of the agent's position, as well as the position of the fridge and light switch. Similar to the previous experiment, we give the x and z positions of the agent, fridge, and light switch to all models as extra state information to make comparisons fairer. Each episode, the agent is positioned randomly in one of the 30 kitchens and runs for a maximum of 200 steps. We perform evaluation every 50 episodes where we run the agent once through each kitchen using greedy actions. This process is repeated 5 times with different random seeds, and the results are shown in Figure 5d.

As in the previous experiment, the only difference between GNet and GNet without GA is the subgoal completion statuses within the augmented state space. In the 3D four rooms environment, each action the agent took affected the x and z positions of the agent. However, this environment contains actions to close objects and toggle objects off, which do not have any effect if the agent is not near any object where these actions are applicable. This problem was demonstrated in the first experiment, where GNet without GA had to learn to associate the target goal with the state space. By using a goal-augmented state space, we provide a generalized way for agent designers to guide agents. DAQN performs similarly to the GNet models because when only considering the low-level policy, the methods are similar. Thus, a key benefit of our proposed method is the use of the full state space in the high-level policy.

5 Related Work

There have been proposed methods to incorporate prior knowledge in a RL agent. A closely related method is the hierarchy of abstract machines (HAM) [22], where partial policies can be defined using a hierarchy of finite state machines. Reward Machines also use finite state machines, but instead of directly defining an agent's behavior, they are used to define reward functions that may represent complex, temporally extended tasks [12]. Andreas et al. proposed a method to include agent designers' prior knowledge using policy sketches that are used to train an agent to complete tasks via subtask sequences [1]. However, a dataset of policy sketches is assumed to be available whereas our proposed method assumes a labelling function is defined. Additionally, policy sketches impose a specific ordering of subgoals whereas a Goal Net model also enables the definition of partially ordered subgoals. Roderick et al. proposed DAQN [25], which extends abstract MDPs [10] by combining tabular RL and DRL. Lyu et al. also propose a HRL method that combines tabular RL and DRL and uses symbolic planning to incorporate prior human knowledge [19]. Our method, however, uses HER to improve sample efficiency and can handle environments where goals may change between episodes. Icarte et al. use Linear Temporal Logic (LTL) formulae to describe tasks and decompose them into subtasks [30]. Our method uses Goal Net to provide a representation of an agent's objectives that is understandable to stakeholders who may have little technical knowledge, however a combination of LTL and Goal Net could be explored in the future. Zhang et al. propose a method where agents learn to plan in a human-defined attribute space, which is similar to the goal space of our method, and use count-based exploration to train agents in a task agnostic manner [34]. Unlike their method, our proposed method augments the state space using the goal space, making our method less reliant on how an agent designer defines the goal space.

Hierarchical reinforcement learning has roots in works such as the options framework [29], MAXQ value function decomposition [8], and Feudal Networks [7]. Many recent works in HRL incorporate deep neural networks. Bacon et al. extended the options framework by training agents to learn the intra-option

and option termination functions in an end-to-end manner [3]. Vezhnevets et al. extend Feudal Networks to use deep neural networks and propose a model where a manager and worker are learned in parallel, and the manager learns to produce goals that represent a desired direction in a learned latent goal space [32]. Levy et al. proposed a method for training a hierarchical agent with potentially many levels of goal-conditioned policies [17]. Nachum et al. improved the sample efficiency of HRL methods using off-policy RL for the high-level policy by correcting transitions in the replay buffer with new goals according to the current low-level policy [21]. We note that our method is not necessarily orthogonal to other HRL methods and could potentially be integrated such that an agent designer proposes high-level subgoals via a Goal Net model and the agent learns to further decompose the subgoals through its own hierarchy.

6 Discussion and Conclusion

We proposed a goal-oriented model and algorithm which use agent designers' prior knowledge to train a hierarchical RL agent. We used Goal Net to accomplish this as goals provide an abstraction of agent behavior that is understandable by agent designers with diverse levels of technical knowledge. We compared our method to two related methods, DAQN and QRM, which make use of similar levels of prior knowledge. We demonstrated that the proposed method can make better use of the information provided to it by the agent designers and learn more quickly in various environments. We also showed that the agent is more robust to the goal space design because we augment the state space of the original MDP rather than reduce it. If the goal space is missing information that may help the agent achieve its goal more efficiently, the agent can still learn because it is not necessarily dependent on the goal space. If an agent designer provides redundant information to the agent, it can still leverage the goal space to learn more efficiently. However, information needed by the agent that is not contained in the state space should be included in the goal space.

A future direction may investigate methods of learning goal spaces to allow agents to have a better understanding of its goals. This may help apply our method to domains outside navigation and goal-reaching tasks, such as dialogue systems. Another direction could involve improving goal selection to handle partially observable environments where the subgoal locations may not be known to the agent. In such environments, the agent may need to change its goal based on new information. An investigation on the use of the proposed method to promote safe AI could be a future direction, as Goal Net can help create agents whose policies are controllable and interpretable. Incorporating other goal types, such as maintenance or avoidance goals, may help in this regard.

Acknowledgments

This research is supported, in part, by the National Research Foundation, Prime Minister's Office, Singapore under its NRF Investigatorship Programme (NRFI

Award No. NRF-NRFI05-2019-0002). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

1. Andreas, J., Klein, D., Levine, S.: Modular multitask reinforcement learning with policy sketches. In: International Conference on Machine Learning. pp. 166–175 (2017)
2. Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O.P., Zaremba, W.: Hindsight experience replay. In: Advances in Neural Information Processing Systems. pp. 5048–5058 (2017)
3. Bacon, P.L., Harb, J., Precup, D.: The option-critic architecture. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)
4. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8**(3), 203–236 (2004)
5. Chevalier-Boisvert, M.: gym-miniworld environment for openai gym. `https://github.com/maximecb/gym-miniworld` (2018)
6. Chevalier-Boisvert, M., Willems, L., Pal, S.: Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid` (2018)
7. Dayan, P., Hinton, G.E.: Feudal reinforcement learning. In: Advances in neural information processing systems. pp. 271–278 (1993)
8. Dietterich, T.G.: Hierarchical reinforcement learning with the maxq value function decomposition. Journal of artificial intelligence research **13**, 227–303 (2000)
9. Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al.: Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In: International Conference on Machine Learning. pp. 1407–1416. PMLR (2018)
10. Gopalan, N., Littman, M.L., MacGlashan, J., Squire, S., Tellex, S., Winder, J., Wong, L.L., et al.: Planning with abstract markov decision processes. In: Twenty-Seventh International Conference on Automated Planning and Scheduling (2017)
11. Hasselt, H.v., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. pp. 2094–2100 (2016)
12. Icarte, R.T., Klassen, T., Valenzano, R., McIlraith, S.: Using reward machines for high-level task specification and decomposition in reinforcement learning. In: International Conference on Machine Learning. pp. 2107–2116 (2018)
13. Kaelbling, L.P.: Learning to achieve goals. In: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence. p. 1094 – 1099 (1993)
14. Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weihs, L., Herrasti, A., Gordon, D., Zhu, Y., Gupta, A., Farhadi, A.: AI2-THOR: An Interactive 3D Environment for Visual AI. arXiv (2017)
15. Kulkarni, T.D., Narasimhan, K., Saeedi, A., Tenenbaum, J.: Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In: Advances in neural information processing systems. pp. 3675–3683 (2016)
16. Leike, J., Krueger, D., Everitt, T., Martic, M., Maini, V., Legg, S.: Scalable agent alignment via reward modeling: a research direction. arXiv preprint arXiv:1811.07871 (2018)

17. Levy, A., Konidaris, G., Platt, R., Saenko, K.: Learning multi-level hierarchies with hindsight. arXiv preprint arXiv:1712.00948 (2017)
18. Lin, L.J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning **8**(3-4), 293–321 (1992)
19. Lyu, D., Yang, F., Liu, B., Gustafson, S.: Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 2970–2977 (2019)
20. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
21. Nachum, O., Gu, S.S., Lee, H., Levine, S.: Data-efficient hierarchical reinforcement learning. In: Advances in Neural Information Processing Systems. pp. 3303–3313 (2018)
22. Parr, R., Russell, S.J.: Reinforcement learning with hierarchies of machines. In: Advances in neural information processing systems. pp. 1043–1049 (1998)
23. Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al.: Multi-goal reinforcement learning: Challenging robotics environments and request for research. arXiv preprint arXiv:1802.09464 (2018)
24. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)
25. Roderick, M., Grimm, C., Tellex, S.: Deep abstract q-networks. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. pp. 131–138 (2018)
26. Schaul, T., Horgan, D., Gregor, K., Silver, D.: Universal value function approximators. In: International Conference on Machine Learning. pp. 1312–1320 (2015)
27. Shen, Z., Miao, C., Gay, R., Li, D.: Goal-oriented methodology for agent system development. IEICE TRANSACTIONS on Information and Systems **89**(4), 1413–1420 (2006)
28. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
29. Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial intelligence **112**(1-2), 181–211 (1999)
30. Toro Icarte, R., Klassen, T.Q., Valenzano, R., McIlraith, S.A.: Teaching multiple tasks to an rl agent using ltl. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. pp. 452–461 (2018)
31. van Lamsweerde, A.: Goal-oriented requirements engineering: a guided tour. In: Proceedings Fifth IEEE International Symposium on Requirements Engineering. pp. 249–262 (2001)
32. Vezhnevets, A.S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., Kavukcuoglu, K.: Feudal networks for hierarchical reinforcement learning. arXiv preprint arXiv:1703.01161 (2017)
33. Yu, E.S.: Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of ISRE'97: 3rd IEEE International Symposium on Requirements Engineering. pp. 226–235. IEEE (1997)
34. Zhang, A., Sukhbaatar, S., Lerer, A., Szlam, A., Fergus, R.: Composable planning with attributes. In: International Conference on Machine Learning. pp. 5842–5851. PMLR (2018)