Interpretable Reinforcement Learning with Multilevel Subgoal Discovery

Alexander Demin , Denis Ponomaryov
Ershov Institute of Informatics Systems, Novosibirsk, Russia
alexandredemin@yandex.ru, ponom@iis.nsk.su

Abstract

We propose a novel Reinforcement Learning model for discrete environments, which is inherently interpretable and supports the discovery of arbitrary deep subgoal hierarchies. In the model, an agent learns information about environment in the form of probabilistic rules, while policies for (sub)goals are learned as combinations thereof. No reward function is required for learning; an agent only needs to be given a primary goal to achieve. Subgoals of a goal G from the hierarchy are computed as descriptions of states, which if previously achieved increase the total efficiency of the available policies for G. These state descriptions are introduced as new sensor predicates into the rule language of the agent, which allows for sensing important intermediate states and for updating environment rules and policies accordingly.

1 Introduction

Hierarchical Reinforcement Learning (HRL) provides a solution to the problem of sample efficiency by the employment of hierarchical policy learning, which gives a clear advantage in comparison to "flat" RL models. The ability to discover subgoal hierarchies allows the agent for interacting with the environment in a more targeted way, which gives an increased learning speed and performance.

Most of the modern end-to-end HRL approaches support the discovery of subgoal hierarchies of a fixed depth. Only some of them do not have this limitation (see Table 2 in (Pateria et al. 2021)), but they provide models, which are not inherently interpretable.

The development of the Deep Learning has greatly broadened the variety of environments and tasks approached by RL, but the problem of interpretabilty of the novel RL models became evident. Typically a subsymbolic RL model is augmented with semantic entities, which make the model behavior more transparent, or it is provided with a glass-box symbolic model, which approximates its behavior. In the recent years, quite a few novel RL approaches have been proposed, which are based on inherently interpretable models (Puiutta and Veith 2020).

Less attention in the modern RL is paid to tasks for discrete environments. However, many non-trivial tasks for RL originate in business and industry, where interpretable models supporting subgoal discovery are required. For example, a whole bunch of tasks is concerned with customer interaction and sales funnel operation, in which it is required to identify the key steps (subgoals) of interaction that lead customers to a purchase in order to guide them on relevant trajectories. Although data preparation and simulation for such environments are separate important problems, we believe that RL technologies for discrete state and action spaces should be further advanced and benchmarked in model environments.

In this paper, we propose a novel RL model for discrete environments, which is inherently interpretable and supports the discovery of arbitrary deep subgoal hierarchies. In our model, an agent learns information about environment in the form of probabilistic rules, while policies for (sub)goals are learned as combinations thereof. No reward function is required for learning; initially an agent only needs to be given a primary goal to achieve. Subgoals of a goal G from the hierarchy are computed as descriptions of states, which, if previously achieved, increase the total efficiency of the available policies for G. These state descriptions are introduced as *invented sensor predicates* into the rule language of the agent, which allows for sensing important intermediate states and for updating environment rules and policies accordingly. The model is implemented as a combination of environment rule, policy, and subgoal learning procedures, which are executed online in an interleaving fashion.

We evaluate our model on a Item Picking Task for grid world environments, in which items of types $1, ..., k$, for $k \geqslant 1$, are randomly distributed, with k being a parameter of the environment. An agent navigating in the environment can pick up an item of each type only once. An item of type i, for $1 \leqslant i \leqslant k$, can be picked up if either $i = 1$, or the agent has previously picked up some item of type $i - 1$. The primary goal set to the agent is to pick up an item of type k. This setting is an abstraction of several prominent subgoal discovery tasks for grid world domains (e.g., the Taxi Domain, in which an agent must first pick up a passenger and then reach a certain destination, or the Multiple-Room Domain, in which an agent must pass a sequence of rooms/doorways in order to access a goal room) and it is general enough for testing the discovery of subgoals of various depths.

In order to provide a ground for examples in the paper, we begin the exposition with a description of a particular instance of the Item Picking Task. Then in Section 3 we de-

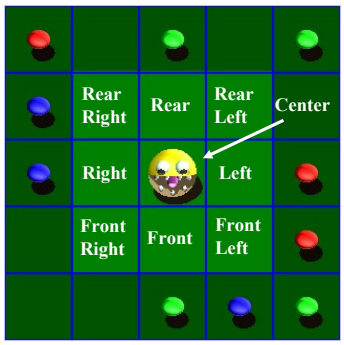

Figure 1: Agent's sensor field

scribe the general architecture of our RL model and consider its components in detail. We provide preliminary experimental results on the performance of the model in Section 4 and we compare our work with the most relevant approaches known in the literature in Section 5. We conclude with a discussion of our approach and topics for future research in Section 6.

2 Example Environment

Reinforcement Learning models for discrete environments are typically benchmarked on grid world domains, in which an agent must learn how to achieve a certain goal state. As an example environment in the paper we consider a 2-dimensional grid of a fixed size, on which items of types $1, ..., k$, for $k \geqslant 1$, are randomly distributed. Every time an agent has a choice of three possible actions: $turn-left, turn{-}right, move$. While moving in the environment the agent picks up items according to the linear order on their types: an item of type i, for $1 \leqslant i \leqslant k$, is picked up iff the agent has reached the position of this item and it has not previously picked up an item of type i, and either $i = 1$, or the agent has previously picked up an item of type $i-1$. The primary goal set to the agent is to pick up an item of type k. Thus, the item types $1, ..., k-1$ correspond to the (sub)goals that the agent must consequently achieve on the way to the primary goal.

The agent has ten sensors in total. Nine of them inform about the types of the items located in the cells near the agent (e.g., $Right(type3)$, see Figure 1), or they indicate $empty$, if the corresponding cells are not occupied by any item (e.g., $Right(empty)$), or $wall$, respectively, if there is a grid boundary. One more sensor named $PickedUp$ indicates whether the agent has picked up some item at current position. Thus, the agent is able to sense the fact of picking up something, but it cannot detect the type of an item it has just picked up, which in general makes the Item Picking Task more complex for the agent.

3 Model Architecture

Our model is based on a combination of four modules, which are depicted on Figure 2. Initially, an agent is given a primary goal as a state description in terms of sensor predicates. For example, $Center(type3), PickedUp$ describes states, in which the agent has reached an picked up an item of type 3. After initialization, the agent performs a number N of random actions in the environment. State transitions made by the agent are recorded in a replay buffer. Every round of N actions is followed by the activation of the Environment Learning module, which learns the effects of agent's actions in the form of (probabilistic) rules. For instance, a typical rule for our example environment could look like

$$Right(type1), turn{-}right \rightarrow Front(type1) \quad (1)$$

which means that turning right in the situation when an item of type 1 is located to the right of the agent results in the situation when the item is in front of it.

Environment Learning is followed by the activation of the Policy Learning module, which learns policies for all (sub)goal states as (probabilistic) rule-like expressions that represent state-action-state trajectories leading to (sub)goals. Policies and environment rules are ranked by the estimation of their probability on the replay buffer. For example, one of the highly ranked policies for a goal state $Center(type3), PickedUp$ in our example environment would look as

$$\begin{array}{c} Right(type3), HasType2\ \{turn{-}right\} \\ Front(type3), HasType2\ \{move\} \\ Center(type3), PickedUp \end{array} \quad (2)$$

where $HasType2$ is a sensor predicate, which is true whenever the agent has previously picked up an item of type 2. These auxiliary predicates are "invented" by the Subgoal Discovery Module launched after each round of M actions (where $M \geqslant N$). For example, $HasType2$ would indicate that the agent has previously achieved the (subgoal) state $Center(type2), PickedUp$. Note that a variant of the policy above without this predicate would look as

$$\begin{array}{c} Right(type3), \{turn{-}right\}\ Front(type3)\ \{move\} \\ Center(type3), PickedUp \end{array} \quad (3)$$

Intuitively, this policy would have a low rank, since the agent is not able to pick up an item of type 3 without having previously picked up one of type 2. Thus, many instances of the trajectory $Right(type3)\{turn-right\}Front(type3)\{move\}$ would not result in the state $Center(type3), PickedUp$. The idea behind the subgoal discovery in our model is that updating policies with predicates for important intermediate states should give policies with an increased rank on average. Subgoal Discovery is followed by the activation of the above mentioned learning modules, which recompute environmental rules and policies in the language with the invented predicates.

Finally, the Action Planning Module is used to decide on the next action depending on the policies applicable in the current situation and on the previously achieved subgoals. The module selects one of the highly ranked policies for the lowest non-achieved (sub)goal in the hierarchy, which is applicable in the current state (Figure 3). If there is no such policy, a random action is performed. Otherwise, the primary action from the policy is executed and the action planning process repeats. Thus, the agent does not have to make a complete sequence of actions from a single policy suitable in the current state; instead it can dynamically combine actions from different policies.

In the next subsections we describe the components of our model in detail.

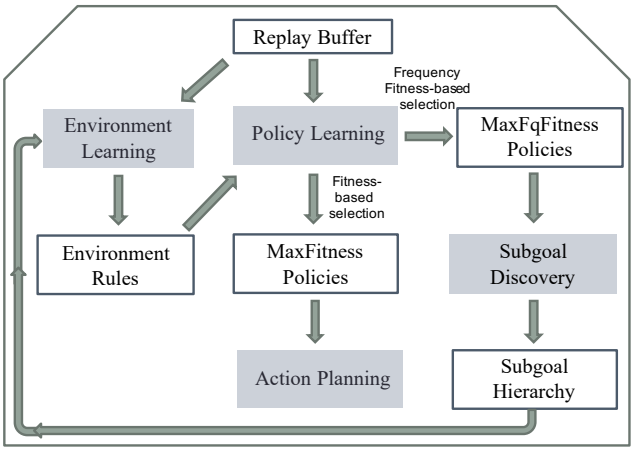

Figure 2: Data flow flow between modules of the model

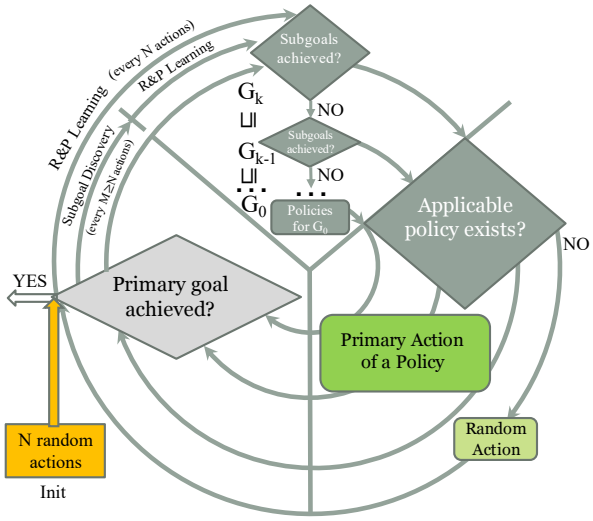

Figure 3: Control flow between modules of the model. Here $G_0 \sqsubseteq \ldots \sqsubseteq G_k$ is an example subgoal hierarchy and "R&P Learning" stands for *Environment Rule and Policy Learning*

3.1 Environment Learning

In our model, an agent learns the effects of actions in the form of probabilistic rules given in the following language. The alphabet of the language is defined as the disjoint union of sets $\texttt{Sens} \cup U_G \cup \texttt{Act}$, where Sens is a set of sensor predicate names, one for each combination of a sensor and a valid indication, U_G is an infinite set of names for (sub)goal predicates, and Act a set of action predicate names, one for each agent's possible action. Although the set U_G is infinite, each time the agent has only a finite subset of (sub)goal predicates $\texttt{SubGoals} \subseteq U_G$ at its disposal. Initially SubGoals is a singleton set, which consists of a predicate name G_{prime}, a notation for the primary goal.

All predicates are nullary and the rule language is inherently propositional. However, in our examples we use shortcuts like $P(c)$ (e.g., $Right(type3)$), where P is a sensor name and c is one of valid indications of the sensor. An agent's *state* is a finite subset of $\texttt{Sens} \cup \texttt{SubGoals}$. We denote the primary goal state as S_{prime}. In the following, we identify a state $\{P_1, ..., P_n\}$, $n \geqslant 0$, with its syntactic representation as a (possibly empty) string $P_1, ..., P_n$ and for a state S, we denote by $|S|$ the number of predicates in S.

Definition 1 (Environment Rule)**.** *An* environment rule *is an expression of the form*

$$S_1, A \to S_2 \tag{4}$$

where S_1, S_2 are non-empty states and $A \in \texttt{Act}$.

For a rule R of the form above we call S_1 and S_2 the *premise* and *conclusion* of R, respectively, and we use notations $\text{pre}(R)$ and $\text{con}(R)$. A *refinement* of a rule R is an environment rule R' such that $\text{con}(R') = \text{con}(R)$, $\text{pre}(R) \subset \text{pre}(R')$, and $|\text{pre}(R')| = |\text{pre}(R)| + 1$.

Let $\langle \Omega, \preceq \rangle$ be a linearly ordered set. We slightly abuse the standard mathematical terminology an call a subset $\{\tau_1, ..., \tau_n\}$ of Ω, where $n \geqslant 1$, a *chain* if for all $i = 1, ..., n-1$ it holds that $\tau_i \preceq \tau_{i+1}$ and for any $\tau \in \Omega$ if $\tau_i \preceq \tau \preceq \tau_{i+1}$ then $\tau = \tau_j$, for some $j \in \{i, i+1\}$.

Definition 2 (Replay Buffer)**.** *A* replay buffer *is a triple $\langle \Omega, \sqsubseteq, \pi \rangle$, where $\sqsubseteq$ is a partial order on* SubGoals*, with G_{prime} being the supremum, $\pi : \texttt{SubGoals} \mapsto 2^{\texttt{Sens}}$ is a mapping such that $\pi(G_{prime}) = S_{prime}$, and Ω is a finite linearly ordered (wrt a relation $\preceq$) set of tuples of the form $\langle S_{pre}, A, S_{post} \rangle$, where $A \in \texttt{Act}$ and S_{pre}, S_{post} are non-empty states, such that for any chain $\{\tau, \tau'\}$ of tuples $\tau = \langle S_{pre}, A, S_{post} \rangle$, $\tau' = \langle S'_{pre}, A', S'_{post} \rangle$ from Ω it holds that $S_{post} = S'_{pre}$.*

Intuitively, a replay buffer serves as a history of agent's transitions provided with the actual subgoal hierarchy $\sqsubseteq$ and an intepretation of subgoal predicates (in terms of states) via the mapping π. In the following, we omit references to the set Ω and say simply that a tuple τ is *from* a replay buffer $\mathcal{B}$. We note that the notion of replay buffer could be reformulated in more strict mathematical terms as a many-sorted algebra.

A predicate $S \subseteq \texttt{Sens} \cup \texttt{Act}$ is *true on a tuple* $\tau = \langle S_{pre}, A, S_{post} \rangle$ from a replay buffer $\mathcal{B}$ if $S \in S_{pre} \cup \{A\}$. A subgoal predicate $G \in \texttt{SubGoals}$ is true on τ if there is a tuple $\tau^{\uparrow} = \langle S^{\uparrow}_{pre}, A^{\uparrow}, S^{\uparrow}_{post} \rangle$ from $\mathcal{B}$ such that

- $\tau^{\uparrow} \preceq \tau$, $\tau^{\uparrow} \neq \tau$
- $\pi(G) \subseteq S^{\uparrow}_{post}$
- for any tuple $\tau^{\downarrow} = \langle S^{\downarrow}_{pre}, A^{\downarrow}, S^{\downarrow}_{post} \rangle$ and subgoal G', if $\tau^{\uparrow} \preceq \tau^{\downarrow} \preceq \tau$ and $G \sqsubset G'$ then $\pi(G') \not\subseteq S^{\downarrow}_{post}$

Informally, the last condition means that the state corresponding to G is achieved in some previous situation $\tau^{\uparrow} \prec \tau$ in the history and the achievement of subgoals is reset by higher goals.

A set $S \subseteq \texttt{Sens} \cup \texttt{SubGoals} \cup \texttt{Act}$ *holds on a tuple* τ from $\mathcal{B}$ (in symbols $\tau \models S$) if every predicate from S is true on τ. A rule $R = S_1, A \to S_2$ *holds on a replay buffer* $\mathcal{B}$ with probability p if $prmconc/prm = p$, where prm is the number of tuples from $\mathcal{B}$, on which $S_1 \cup \{A\}$ holds, and

$prmconc$ is the number of tuples $\tau = \langle S_{pre}, A, S_{post}\rangle$ from $\mathcal{B}$ such that $S_1 \cup \{A\}$ holds on τ, $S_2 \cap \texttt{Sens} \subseteq S_{post}$, and $\pi(G) \subseteq S_{post}$, for all $G \in S_2 \cap \texttt{SubGoals}$. Intuitively, p is the frequency probability of the event $\mathrm{con}(R)$ conditioned on the event $\mathrm{pre}(R)$ in the history given by $\mathcal{B}$ provided that every subgoal predicate G from the conclusion of R is viewed as the event of the achievement of $\pi(G)$. In the paper, we use the *partial* function $\mathtt{p}$, which for a rule R and a replay buffer $\mathcal{B}$ gives the probability of R on $\mathcal{B}$ as above, provided $prm \neq 0$.

Definition 3 (Probabilistic Law)**.** *A rule R is a* probabilistic law *wrt a replay buffer $\mathcal{B}$ if $\mathtt{p}(R, \mathcal{B})$ is defined and $\mathtt{p}(R', \mathcal{B}) < \mathtt{p}(R, \mathcal{B})$, for any rule R' such that $pre(R') \subset pre(R)$.*

For the example environment from Section 2, the rule $R = Right(type3), turn-right \rightarrow Front(type3)$ has probability 1 on any replay buffer $\mathcal{B}$, whenever $\mathtt{p}(R, \mathcal{B})$ is defined. The value of $\mathtt{p}(R, \mathcal{B})$ is undefined in case the agent did not experience the state $Right(type3)$, or did not make a turn to the right from this state. Otherwise $\mathtt{p}(R, \mathcal{B}) = 1$, because the environment is non-stochastic and thus, turning right in the situation, when an item is located to the right of the agent always brings it to the situation, in which the item is located in front of it. If $\mathtt{p}(R, \mathcal{B}) = 1$ then it holds that R is a probabilistic law.

By definition, probabilistic laws with a probability value p are exactly those rules, which by inclusion of premises, are the shortest ones among all the rules having a probability at least p. Thus, the concept of a probabilistic law in our model captures the balance between the size and "informativeness" of environment descriptions, the topic discussed broadly in AI. It is known that searching for the shortest implications true (with probability 1) on a given data is a computationally hard problem. There may exist exponentially many such shortest implications (Theorem 1 in (Kuznetsov 2004)) and the problem to decide whether there is one of a given size is NP-complete (Gunopulos et al. 2003).

The environment learning procedure in our model is implemented as a heuristic-based enumeration of rules by refinement, with the selection of those ones, which satisfy the properties of a probabilistic law. The procedure is given by Algorithm 1.

The heuristic in the algorithm implies that after base enumeration up to depth d (which is one of the hyperparameters of our model) a rule is selected as an additional candidate member for the resulting set of laws only if it refines one of the probabilistic laws R obtained by base enumeration and has a strictly higher probability than R. Monotonicity in this sense does not hold in general. It can be the case that every refinement of a rule R has a lower probability than R, but there is a rule R' obtained by extending the premise of R, e.g., with two predicates, such that R' is a probabilistic law. Our algorithm can find the law R' only if the parameter d is greater than $|\mathrm{pre}(R')|$.

The assumption behind the heuristic is that state descriptions, which are most informative for learning the environment, usually involve a moderate number of sensor predicates. Hence, the majority of probabilistic laws could be found by the algorithm with a small parameter d. By relying further on monotonicity, the algorithm tries to find probabilistic laws with higher probabilities. This is implemented by computing refinements of probabilistic laws obtained by base enumeration incrementally as long as this gives new laws.

Algorithm 1: Environment Rule Learning

Input : Replay buffer $\mathcal{B}$, a non-empty state S
Parameter: Base enumeration depth $d \geqslant 1$
Output : A set of prob. laws wrt $\mathcal{B}$ with conclusion S
begin
1 RULES := $\{R \mid R$ is a rule, with $\mathrm{con}(R) = S$, $|\mathrm{pre}(R)| \leqslant d\}$ /* environment rule enumeration */
2 LWS := $\{R \in$ RULES $\mid R$ is a probabilistic law$\}$
3 $2Refine$:= $\{R \in$ LWS $\mid \neg\exists R' \in$ LWS s.t. $\mathrm{pre}(R) \subset \mathrm{pre}(R')\}$
4 **while** $2Refine \neq \varnothing$ **do**
5 Let $R \in 2Refine$; $2Refine := 2Refine \setminus \{R\}$
6 **foreach** *refinement R' of R* **do**
if *R' is a probabilistic law* **then**
7 $2Refine := 2Refine \cup \{R'\}$
8 LWS := LWS$\cup\{R'\}$
9 **return** LWS

Besides d, an implementation of Algorithm 1 employs the following hyperparameters for fine tuning. To exclude laws from the output, which have a low probability or statistical significance (on the replay buffer), the parameters *Probability-Threshold* and *Confidence-Threshold* are used, respectively. *Probability-Gain-Threshold* is used to exclude those laws from further refinement, for which the last refinement step has given a low probability gain. Finally, the *Max-Sensor-Predicates* parameter restricts the search to the rules with a given maximal number of sensor predicates in the premise. We comment on the hyperparameter settings for experiments in Section 4.

3.2 Policy Learning

Definition 4 (Policy)**.** *A* policy *P for a non-empty state G is an expression of the form*

$$S_1\ \{A_1\} \ldots S_n\ \{A_n\}\ G \tag{5}$$

where $n \geqslant 1$ and for all $i = 1, ..., n$, $A_i \in \texttt{Act}$ is an action predicate and S_i a non-empty state such that $S_i \cap \texttt{SubGoals} = S$, for a (possibly empty) set S.

Intuitively, the policy gives a state-action-state trajectory leading to state G. The last condition in the definition implies that subgoals are never lost along the way to G and they cannot be suddenly achieved at some point of the trajectory. Thus, the policy describes only how the target state G could be achieved, while the ways of how all the required subgoals could be achieved are to be given by separate policies.

For a policy P of the form above the set S_1 is called the *policy premise* (we abuse notation and write $\mathrm{pre}(P)$ to denote the premise of P) and A_1 is called the *primary action*

of P. The number n is called the *length* of the policy and it is denoted as $len(P)$.

Definition 5 (Policy Fitness)**.** *Given a replay buffer $\mathcal{B}$, the* fitness *of a policy $P = S_1\ \{A_1\} \ldots S_n\ \{A_n\}\ S_{n+1}$ wrt $\mathcal{B}$ (in symbols $\texttt{fitness}(P, \mathcal{B})$) is the product of probabilities $\mathrm{p}(S_i, A \to S_{i+1},\ \mathcal{B})$, for all $i = 1, ..., n$, if each of them is defined. Otherwise the fitness of P is undefined.*

Let $P = S_1\ \{A_1\} \ldots S_n\ \{A_n\}\ G$ be a policy and $R = S_0, A_0 \to S_1$ an environment rule such that $S_0 \cap \texttt{SubGoals} = S_1 \cap \texttt{SubGoals}$. A *refinement* of P with R denoted as $\mathrm{REFN}(P, R)$ is the policy $S_0\ \{A_0\}\ S_1\ \{A_1\} \ldots S_n\ \{A_n\}\ G$. If P and P' are policies for the same state then P' is called a *variant* of P if $\mathrm{pre}(P') \subseteq \mathrm{pre}(P)$ and $P \neq P'$. That is, a variant policy provides an alternative trajectory to achieve the same target state from the same or a more general situation.

For a replay buffer $\mathcal{B}$ and a state S, let $Algo1(\mathcal{B}, S)$ denote the set of probabilistic laws obtained by Algorithm 1 on the input[1] $\mathcal{B}, S$. We now describe a procedure for computing a set of policies for a given state G. The procedure builds every policy expression backwards, starting from G, by incremental refinement with probabilistic laws computed by Algorithm 1. A policy obtained this way may thus represent a trajectory that has not been seen as a whole by the agent before. Only particular transitions (given by probabilistic laws) could have been experienced by the agent. This enables the agent to "reason" about unseen trajectories, which contributes to the sample efficiency of our model.

The policy learning algorithm is given below. By the definition of the fitness function, longer policies get lower fitness values, therefore the algorithm computes *strong* policies, i.e. those, which have the greatest fitness value among all variants.

Algorithm 2: Policy Learning

Input : Replay buffer $\mathcal{B}$, non-empty state G
Output : A set of policies for G
begin
1 LWS := $Algo1(\mathcal{B}, G)$
2 Let POL := $\{S_{pre}\ \{A\}\ G \mid S_{pre}, A \to G \in \mathrm{LWS}\}$
/* initialize a set of policies */
3 Let $2Process$:= POL; RPol := $\varnothing$
4 **while** $2Process \neq \varnothing$ **do**
5 **for** $policy \in 2Process$ **do**
6 LWS := filterbyGoals($Algo1(\mathcal{B}, \mathrm{pre}(policy))$)
7 RPol := RPol $\cup$ $\{\mathrm{REFN}(policy, R) \mid R \in \mathrm{LWS}\}$
8 $2Process$:= getStrong(RPol, POL, $\mathcal{B}$)
9 POL := POL $\cup$ $2Process$
10 **return** POL

An implementation of Algorithm 2 employs two hyperparameters for fine-tuning: the *Maximal-Policy-Length* parameter is used to restrict the search only to policies of a fixed maximal length and *Fitness-Gain-Threshold* prevents further refinement of those policies, whose fitness is below a given value.

[1] we assume that all hyperparameters for algorithms are fixed globally.

Function filterbyGoals($rules$)

forall $R \in rules$ **do**
  if $pre(R) \cap \texttt{SubGoals} \neq con(R) \cap \texttt{SubGoals}$ **then**
    rules := rules$\backslash\{R\}$
return $rules$

Function getStrong($policies, wrtpolicies, rBuffer$)

foreach $P \in policies$ **do**
  if $\exists$ *variant* $P' \in wrtpolicies$ *of* P *s.t.* $\texttt{fitness}(P, rBuffer) \leqslant \texttt{fitness}(P', rBuffer)$ *and* $len(P') < len(P)$ **then**
    $policies := policies \setminus \{P\}$
return policies

3.3 Subgoal Discovery

Recall policy (3) for our example environment, which is composed of probabilistic laws $R_1 = Right(type3),\ turn-right \to Front(type3)$ and $R_2 = Front(type3),\ move \to Center(type3), PickedUp$. Let us denote this policy as P. Even if the rule R_1 has probability 1 (on a replay buffer $\mathcal{B}$), the fitness of P may be quite low if so is the probability of R_2 (i.e., if only in few situations in the history the agent was able to pick up an item of type 3 in front of it). However, if we consider only those situations, when it is known that the agent has previously picked up an item of type 2 then the picture becomes different. If an item of type 3 happens to be in front of the agent in one of these situations and the action $move$ is made, then with high probability the agent picks up this item.

If the alphabet $\texttt{SubGoals}$ is extended with a predicate named, e.g., $HasType2$, which is true whenever the agent has previously achieved the state $Center(type2), PickedUp$, then we can consider an update of the rule R_2 as $R_2' = Front(type3), HasType2, move \to Center(type3), PickedUp$, which is now (by an argument as in Section 3.1) a candidate for being a probabilistic law.

Similarly, we can consider an update of R_1 as $R_1' = Right(type3), HasType2,\ turn-right \to Front(type3), HasType2$ obtained by adding the "invented" predicate to the premise and conclusion of R_1.

Finally, consider an update P' of the policy P as (2) in Section 3. If the probability of the both rules R_1' and R_2' is 1, then so is the fitness of P, i.e., P gives the most plausible trajectory to achieve the goal state $G = \{Center(type3), PickedUp\}$ from the state $\{Right(type3), HasType2\}$, where $HasType2$ is a predicate for the *subgoal* state $\{Center(type2), PickedUp\}$ of G.

The idea behind the subgoal discovery in our model is that updating policies for a goal G with the "invented" subgoal

predicates as above should provide policies, which more probably lead to G, than the original ones. Given a replay buffer $\mathcal{B}$, the subgoal states for a state G are searched as combinations of sensor predicates, which are subsets of states visited by the agent. A discovered subgoal state S gets a fresh predicate name P from $U_G \setminus$ SubGoals as a shortcut, the set SubGoals is extended with P, and the value of the interpretation function π for P is defined as S.

Let us note however that the policy P (and hence, P' in our example) may represent a trajectory that the agent has never experienced, i.e., there may exist separate tuples $\tau = \langle\{Right(type3), HasType2\}, turn-right, \{Front(type3), HasType2\}\rangle$ and $\tau' = \langle\{Front(type3), HasType2\}, move, \{Center(type3), PickedUp\}\rangle$ in the replay buffer, but no chain $\{\tau, \tau'\}$. Therefore, to estimate the potential increase of the efficiency of policies after an update with a candidate subgoal, we take into account only those policies, which correspond to trajectories passed by the agent. For this, we introduce a frequency probability measure, called *frequency fitness*, which indicates for a policy $S_1\{A_1\}\ldots S_n, \{A_n\}\ G$, $n \geqslant 1$, how often the agent achieved the goal state G by following the trajectory $S_1\{A_1\}\ldots S_n, \{A_n\}$.

Let $\mathcal{B}$ be a replay buffer and $P = S_1\ \{A_1\}\ldots S_n\ \{A_n\}\ G$ a policy, where $n \geqslant 1$. A tuple τ_1 is called a (possible) *starting point* of P in $\mathcal{B}$ if there is a chain of tuples $\{\tau_1, .., \tau_n\}$ from $\mathcal{B}$ such that $\tau_i \models S_i \cup \{A_i\}$, for all $i = 1, .., n$. A tuple $\tau_n = \langle S_{pre}, A_n, S_{post}\rangle$ from $\mathcal{B}$ is called a (possible) *ending point* of P in $\mathcal{B}$ if there is a chain $\{\tau_1, .., \tau_n\}$ in $\mathcal{B}$, which satisfies the above condition, where $\tau_n = \langle S_{pre}, A_n, S_{post}\rangle$ is a tuple such that $G \cap$ Sens $\subseteq S_{post}$ and $\pi(S) \subseteq S_{post}$, for any $S \in G \cap$ SubGoals.

Definition 6 (Frequency Fitness)**.** *For a policy P and a replay buffer $\mathcal{B}$, the* frequency fitness *of P wrt $\mathcal{B}$ (denoted as* fqfitness$(P, \mathcal{B})$*) is defined as E/S if $S \neq 0$, where E and S is the number of ending and starting points of P in $\mathcal{B}$, respectively, and it is undefined otherwise.*

For a replay buffer $\mathcal{B}$ and a state G, let *Algo2*$(\mathcal{B}, G,$ fqfitness$)$ be the set of rules computed by Algorithm 2, which employs the frequency fitness instead of the original fitness measure. The procedure for the discovery of subgoals for a state G is given by Algorithm 3.

For every tuple τ witnessing the achievement of G, the algorithm selects those policies from *Algo2*$(\mathcal{B}, G,$ fqfitness$)$ with the highest frequency fitness, for which τ is an ending point. Then the family of the best policies computed for all witness tuples is taken as an "aggregate" policy and its frequency fitness (function AvgFqFitness given below) is estimated before and after an update with a candidate subgoal state. Those states, wrt which the fitness gain is greater or equal a certain threshold β (which is a hyperparameter), are taken as subgoals of G and are introduced as fresh subgoal predicate names into the language of the agent.

3.4 Action Planning

For a (subgoal) state G, let *Algo2*$(\mathcal{B}, G)$ denote the set of policies for G computed by Algorithm 2. The action planning algorithm given as a function below essentially makes ranking of the available policies for G depending on the current state of the agent (the truth values of the sensor and subgoal predicates) and the ranking of policies for subgoals of G. The procedure either outputs a random action (in case there is no policy applicable in the current state), or the primary action of the best policy for G if all subgoals of G are achieved, or the primary action of a policy for a minimal (wrt $\sqsubseteq$) non-achieved subgoal of G.

Algorithm 3: Subgoal Discovery

Input : Replay buffer $\mathcal{B}$ and $G \in$ SubGoals
Parameter: Base fitness gain β
Output : An update of $\mathcal{B}$ and SubGoals
begin
- Let POL := *Algo2*$(\mathcal{B}, G,$ fqfitness$)$
- Let $BESTPOL := \varnothing$
- **forall** $\tau = \langle S_{pre}, A, S_{post}\rangle$ *from* $\mathcal{B}$ *s.t.* $\pi(G) \subseteq S_{post}$ **do**
  - Let P_τ := $\{p \in$ POL $\mid \tau$ is an ending point of $p\}$
  - $BESTPOL := BESTPOL \cup \{p \in P_\tau \mid \neg\exists p' \in P_\tau$ s.t. fqfitness$(p,\mathcal{B}) <$ fqfitness$(p',\mathcal{B})\}$
- **if** $BESTPOL = \varnothing$ **then**
  - **return** $\mathcal{B}$, SubGoals /* no subgoal can be found */
- **forall** $S \subseteq$ Sens *s.t. S holds on a tuple from* $\mathcal{B}$ **do**
  - **if** $\neg\exists P_S \in$ SubGoals *s.t.* $\pi(P_S) = S$ *and* $P_S \sqsubseteq G$ **then**
    - Let $P_S \in U_G \setminus$ SubGoals; $\pi(P_S) := S$; SubGoals := SubGoals $\cup \{P_S\}$ /* invented predicate for S */
    - Let $UPDPOL := \varnothing$
    - **forall** $S_1\{A_1\}\ldots S_n\{A_n\}\ G \in BESTPOL$ **do**
      - $UPDPOL := UPDPOL \cup \{S_1, P_S\ \{A_1\}\ldots S_n, P_S\ \{A_n\}\ G\}$
    - **if** *AvgFqFitness(UPDPOL,*$\mathcal{B}$*)-AvgFqFitness(BESTPOL,* $\mathcal{B}$*)* $\geqslant \beta$ **then**
      - $\sqsubseteq := \sqsubseteq \cup \{P_S, G\}$ /* inserted P_S into subgoal hierarchy */
    - **else**
      - SubGoals := SubGoals $\setminus \{P_S\}$ /* removed useless P_S */
- **return** $\mathcal{B}$, SubGoals

Function AvgFqFitness$(pol, rBuf)$

s := no. of τ s.t. $\exists p \in pol(\tau$ is a start. point of p in $rBuf)$
e := no. of τ s.t. $\exists p \in pol(\tau$ is an end. point of p in $rBuf)$
return 0 if $s = 0$ or e/s, otherwise

4 Preliminary Experimental Results

For experiments we used an implementation of the environment for the Item Picking task introduced in Section 2 on a grid of dimension 25x25. The environment was provided

Function Action Planning(*(sub)goal state* G, *replay buffer* $\mathcal{B}$)

Let $isRandom$:= true `/* random action flag */`
Let $curstate$ be the maximal tuple wrt $\preceq$ in $\mathcal{B}$
`/* current situation */`
Let $BESTPOL := getBestPol(G, curstate, \mathcal{B})$
while $BESTPOL \neq \varnothing$ **do**
  Let $P \in BESTPOL$ `/* non-deterministic choice */`
  $BESTPOL := BESTPOL \setminus \{P\}$
  if $Rank(P, curstate, \mathcal{B}) \neq 0$ **then**
    $subG$:= pre(P) $\cap$ `SubGoals` `/* subgoals of G in policy P */`
    if $curstate \models subG$ **then**
      return ⟨ primary action of P, $\neg isRandom$⟩
    Let $S \in subG$ s.t. $curstate \not\models S$ `/* non-deterministic choice */`
    Let $\langle a, israndm \rangle$:= Action Planning(S, $\mathcal{B}$)
    if $\neg israndm$ **then**
      return $\langle a, false \rangle$
return ⟨random action from `Act`, $isRandom$⟩ `/* no policy applicable in the curr.state */`

Function getBestPol(S, $curstate$, $\mathcal{B}$)

Let POL := *Algo2*($\mathcal{B}, S$)
return $\{P \in$ POL $\mid \forall P' \in$ POL $Rank(P, curstate, \mathcal{B}) \geqslant Rank(P', curstate, \mathcal{B})\}$

Function Rank($policy$, $curstate$, $\mathcal{B}$)

if $curstate \not\models pre(policy) \setminus$ `SubGoals` **then**
  return 0 `/* policy not applicable in the current situation */`
Let subG := pre($policy$) $\cap$ `SubGoals`
if $curstate \models subG$ **then**
  return `fitness`($policy, \mathcal{B}$) `/* all subgoals from policy (if any) are achieved */`
forall $S \in subG$ **do**
  Let P[S] $\in getBestPol(S, curstate, \mathcal{B})$
return `fitness`($policy, \mathcal{B}$) $\times \prod_{S \in \text{subG}}$Rank(P[$S$], $curstate, \mathcal{B}$)

with the feature that each time an agent picked up some item an item of the same type appeared at a random non-occupied position on the grid. To support continuous learning, the "semantics" of the subgoal predicates was modified in such a way that as soon as the agent achieved the primary goal, i.e., picked up an item of the maximal type k, all the subgoal predicates became false. The fact of the achievement of an item of the maximal type was recorded in a primary-goal-counter and then the agent was set to achieve the primary goal again. In the experiments, the agent performance was measured in environments with $k = 1, 2, 3$ by the number of the primary goals achieved in a course of 1000 actions. The hyperparameters of the algorithms were set as follows. For Algorithm 1: base enumeration depth d=3, Probability-Threshold=0.1, Confidence-Threshold=0.9, Probability-Gain-Threshold=0.1, Max-Sensor-Predicates=1. For Algorithm 2: Maximal-Policy-Length=4, Fitness-Gain-Threshold=0.5. For Algorithm 3, the base fitness gain parameter β was set to 0.2.

In the experiment for $k = 1$, the number N of initial random actions was equal to 100 and for $k = 2, 3$ it was set to 2000. The upper bound on the number of actions was 10000 in every experiment. Each of the modules of our model (See Figure 2) was initiated every N steps. To smooth the effects of random item distribution in the environment we averaged the performance measurement on ten runs in each experiment. Figure 4 presents experimental results for the environment with $k = 1$, in which the item type hierarchy is degenerate. In this experiment the agent did not discover

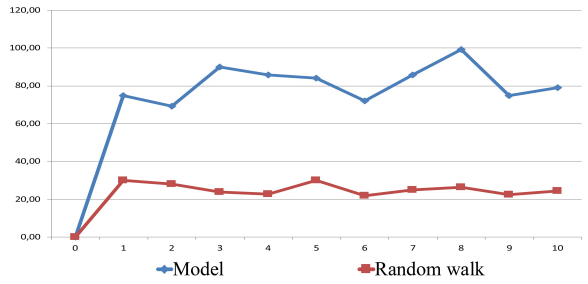

Figure 4: Performance in environment for $k = 1$

any subgoal and it showed a relatively high performance already after the first 100 steps. As subgoals were not used, the achieved performance was only due to the work of the Environment Learning and Policy Learning modules.

The performance rate was non-stable in experiments, which is explained by the following fact. The sensor field of the agent is local and allows for acting efficiently only in the proximity of an item. Our model does not solve the problem of efficient exploration of the environment when the items are out of reach of agent's sensors (this is one of the topics for future research). As a consequence, if an agent cleans up a certain region by picking up items, while new items are randomly generated distantly elsewhere, then it has to make quite a number of (random) steps to reach them, which yields a decreased performance. The performance change is even more radical in the environments with several item types (Figures 5, 6), since items of a required type may happen to be located even more distantly on average from the current agent's position than in the environment with a single item type. In each of the experiments the local performance maxima correspond to the "islands" in the environment, in which there was enough items of required types and thus, the agent was able to use its sensor field efficiently.

For the environment with $k = 3$, Figure 6 shows the relative performance of an agent with unlimited subgoal capacity to that of an agent, which is able to discover a single subgoal. The result shows that the environment is inherently dif-

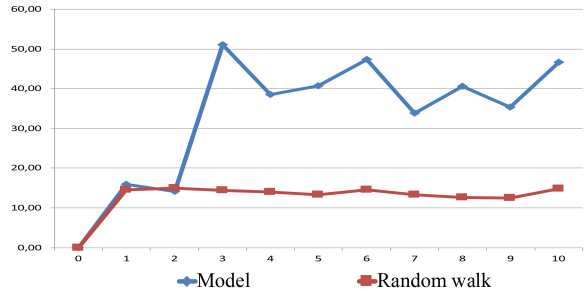

Figure 5: Performance in environment for $k = 2$

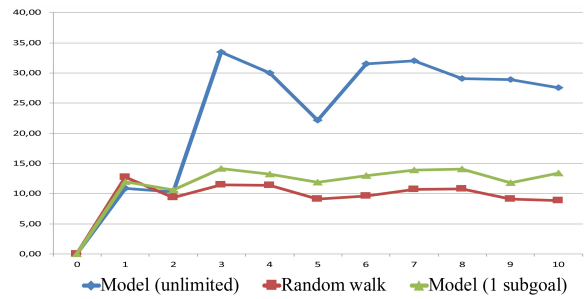

Figure 6: Performance in environment for $k = 3$

ficult for the latter agent. On the other hand, the agent capable of discovering deeper subgoal hierarchies could achieve high performance already after the first round of learning.

A video demonstrating agent's behavior in the above mentioned environments and a listing of policies learned in the experiments are available at shorturl.at/deB24

5 Related Work

In most of the HRL models with subgoal discovery known in the literature the depth of the subgoal hierarchy or the total maximal number of subgoals is restricted by a hyperparameter. There are approaches, in which the problem of hierarchical policy learning is solved with subgoal discovery in a unified fashion, and there are proposals focused on the subgoal discovery as the central problem, without any relation to concrete RL models. To the best of our knowledge, there are only two approaches in the first category, in which arbitrarily deep subgoal hierarchies are supported. First, we comment on each of them.

In (Konidaris and Barto 2009) a method for skill chaining based on the Options Framework (Sutton, Precup, and Singh 1999) is proposed, in which tree-like sequences of options (skills) are built by a backward chaining algorithm, from a goal to subgoals. The initiation states for each option are determined by a state classifier, which discovers those states, from which the achievement of the corresponding (sub)goal state is most likely. The initiation states are then taken as subgoal states and the backward chaining procedure is applied recursively to each of them, thus producing a tree-like option hierarchy. In our model, the policy learning procedure is conceptually similar to the backward skill chaining and each environment rule used in the formulation of a policy could in principle be viewed as a representation of a particular skill (of making a one-step transition between two states). The environment rule learning module could then be seen as an analogue of a state classifier that gives the most probable states, from which a given state is one-step accessible. In our approach however, subgoals differ from skills in that they are learned on top of the previously computed policies and correspond to the descriptions of states, which, if previously achieved, increase the total efficiency of the available policies. Importantly, these state descriptions are introduced as new predicates into the language of the agent, which allows for adapting environment rules and policies accordingly.

In (Vezhnevets et al. 2017) the ideas of Feudal Reinforcement Learning (Dayan and Hinton 1993) are implemented in a two-level model, in which the higher level (manager) sets subgoals to the lower one (worker) as learned abstractions of important intermediate states on the way to the primary goal. The manager communicates the next subgoal to the worker, but it does not specify how to achieve it. The manager and the worker learn independently of each other and operate at different temporal resolutions, which allows for solving tasks involving long-term credit assignment. This way the model is able to support more abstract (automatically learned) subgoals, e.g., "obtain X" in comparison to "obtain X at location Y". We note that these features are present in our approach too. Action Planning in our model can be viewed as a procedure, which communicates tasks between the control levels corresponding to the learned subgoal hierarchy (see Figure 3). Abstract tasks (such as, e.g., "obtain an item of type i") are supported. An important feature of our approach, when formulated in the terms of Feudal RL, is that the model makes a choice of the most appropriate next worker dynamically depending on the current state of the agent and the achieved subgoals.

In most of the approaches to solving the subgoal discovery as a standalone problem subgoals are searched as bottleneck nodes in the State Transition Graph (built for the complete environment or its approximation) by using centrality measures, clustering, or spectral analysis (Mendonça, Ziviani, and Barreto 2019). There are several approaches however, which rely solely on the transition history of an agent instead of the State Transition Graph. For example, (McGovern and Barto 2001) employs the Options Framework (Sutton, Precup, and Singh 1999) and the concept of *diverse density* for discovering bottleneck regions in the agent's observation space. Agent's trajectories are divided into positive (which bring the agent to a goal) and negative ones. Those states are taken as candidate options, which are often present in positive trajectories and never in negative ones. Initiation states for an option (a subgoal) G are defined as those states, which have been visited some n steps before G, where n is a hyperparameter. An additional parameter is used to exclude candidate options, which are too close to the previously discovered subgoals or their initiation states. (Chen et al. 2007) employs the same idea for subgoal discovery, but it uses a different filtering mechanism for candidate subgoal states, which leaves only those ones that correspond to local and global visitation maxima. The idea of interpret-

ing subgoals as bottleneck states in the sense above is similar to how we view subgoals in our paper. For subgoal discovery, we analyze trajectories that correspond to the most efficient policies. The difference is that we consider them as an aggregate policy and we compare its efficiency before and after adding the fact of the achievement of a candidate subgoal, which facilitates finding only the most significant subgoals.

From the point of view of interpretability in RL, in the recent couple of years quite a few models have been proposed in the literature, which are transparent and inherently explainable. Many approaches in interpretable RL are based on a post-hoc analysis and explanation of pretrained black-box models with the help of more simple glass-box models. A slightly different way is followed in (Verma et al. 2018), which employs a pretrained black-box NN model for learning a declarative model, in which policies are given by functional programs. The declarative model is learned to approximate the output of the NN model and it gives human-readable policies, which provide more smooth control in experiments. An important point in this approach is that only those policies are learned that comply with a given syntactic template, the form of which is essentially a hyperparameter dependent on the environment. As in (Verma et al. 2018), the policies in our approach are declarative: they can be viewed as probabilistic rules stating that a sequence of transitions implies the achievement of a (sub)goal with a certain probability. The general form of the rules is not restricted by any domain dependent template, but there are hyperparameters, which allow for fine-tuning the properties of the learned policies.

In (Hein et al. 2017) a rule-based approach is used for the development of fuzzy controllers, which learn via interaction with a NN-simulated environment. Policies are defined as instantiations of fuzzy rule templates, the parameters of which are learned by particle swarm optimization, although evolutionary algorithms or gradient descend are equally applicable. For instance, in the earlier work (Juang, Lin, and Lin 2000) evolutionary algorithms have been used for computing both, rule templates and their parameters. It would be useful to apply these algorithms for environment rule and policy learning in our approach and we leave this for future research. Our work however is conceptually different from (Hein et al. 2017), (Juang, Lin, and Lin 2000), and similar approaches in an important aspect. In the named approaches, in each round of learning the complete set of policies is generated in a top-down fashion and it is then evaluated in the environment. In case performance is unsatisfactory, the whole process repeats. In contrast, our approach is based on goal directed policy generation and supports continuous learning on agent's history consisting of random actions and policy guided transitions. Each policy is built upon probabilistic laws, which are environment rules with balanced size and informativeness. This facilitates the readability of the learned policies and contributes to the sample efficiency of our model.

6 Discussion and Outlook

The proposed model uniquely combines the ability to generate human-readable policies and to discover subgoals, for which an agent does not initially have sensors. Unlike the common RL approaches, the model does not require a reward function to be specified. We believe that the model deserves further development and it can be used as a framework for testing the limits of HRL based on rule learning. Some ideas of hierarchical learning in our model could be employed in the general RL, also for tasks in continuous environments.

The current version of our model supports only incremental achievement of subgoals: an agent can not lose a previously achieved subgoal (e.g., an item it has picked up) and subgoals can not be conflicting (e.g., picking up an item of one type cannot make picking up an item of another type impossible). In general, the subgoal discovery mechanism should be enhanced in order to support different logical constraints between subgoals imposed by environment.

In complex scenarios it can be the case that a particular subgoal is rarely accessible (compared to another subgoal S), but once achieved it allows the agent to reach the primary goal faster. The ability to reason about subgoals this way needs to be integrated into the model, because in the current implementation, an agent will tend to prefer S as a subgoal, as most of the trajectories go through S. In our model, an agent learns how to act efficiently in situations when a (sub)goal is located in the scope of its sensor field, but the model does not help in exploring the environment in other situations; thus, the exploration problem should further be addressed.

Some policies learned by the agent may happen to be faulty if it starts learning in some "atypical" part of environment. For example, if an agent is initialized in a specific area of our model environment, in which the distribution of items is atypically dense, then it could learn probabilistic laws, which would not generalize well. If their probability degrades slowly during exploration then these faulty rules will be for a long time preferred for building policies. In general, it is important to develop mechanisms that could help the agent to recognize significant changes in the environment in order to recalculate rules and policies accordingly.

In this paper, we did not discuss the question of optimality of policies. As policies in our model are human-readable, we could confirm in each of our experiments that the agent obtained optimal policies already after the first round of learning. However, this property obviously needs a theoretical investigation. Finally, it is important to note that several components in our model employ hyperparameters for fine-tuning. Much study could be devoted to the influence of these parameters on agent's performance in various environments. However, it would be more interesting to investigate whether the model architecture could live without (at least some of) the hyperparameters, which is also a part of our further research.

References

Chen, F.; Chen, S.; Gao, Y.; and Ma, Z. 2007. Connect-based subgoal discovery for options in hierarchical reinforcement learning. In *Proceedings - Third International Conference on Natural Computation, ICNC 2007*, volume 4.

Dayan, P., and Hinton, G. E. 1993. Feudal reinforcement learning. In Hanson, S.; Cowan, J.; and Giles, C., eds., *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann.

Gunopulos, D.; Khardon, R.; Mannila, H.; Saluja, S.; Toivonen, H.; and Sharma, R. S. 2003. Discovering All Most Specific Sentences. *ACM Transactions on Database Systems* 28(2).

Hein, D.; Hentschel, A.; Runkler, T.; and Udluft, S. 2017. Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies. *Engineering Applications of Artificial Intelligence* 65.

Juang, C. F.; Lin, J. Y.; and Lin, C. T. 2000. Genetic reinforcement learning through symbiotic evolution for fuzzy controller design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 30(2).

Konidaris, G., and Barto, A. 2009. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems 22 - Proceedings of the 2009 Conference*.

Kuznetsov, S. O. 2004. On the intractability of computing the duquenne-guigues base. In *Journal of Universal Computer Science*, volume 10.

McGovern, A., and Barto, A. G. 2001. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *Proceedings of the 18th International Conference on Machine Learning*, 361–368. Morgan Kaufmann.

Mendonça, M. R.; Ziviani, A.; and Barreto, A. M. 2019. Graph-based skill acquisition for reinforcement learning. *ACM Computing Surveys* 52(1).

Pateria, S.; Subagdja, B.; Tan, A. H.; and Quek, C. 2021. Hierarchical Reinforcement Learning: A Comprehensive Survey.

Puiutta, E., and Veith, E. M. 2020. Explainable Reinforcement Learning: A Survey. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12279 LNCS.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1).

Verma, A.; Murali, V.; Singh, R.; Kohli, P.; and Chaudhuri, S. 2018. Programmatically interpretable reinforcement learning. In *35th International Conference on Machine Learning, ICML 2018*, volume 11.

Vezhnevets, A. S.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; and Kavukcuoglu, K. 2017. FeUdal networks for hierarchical reinforcement learning. In *34th International Conference on Machine Learning, ICML 2017*, volume 7.