Autonomous Generation of Symbolic Knowledge via Option Discovery

Gabriele Sartor[1], Davide Zollo[2], Marta Cialdea Mayer[2], Angelo Oddi[3], Riccardo Rasconi[3] and Vieri Giuliano Santucci[3]

[1]*University of Turin*
[2]*Roma Tre University*
[3]*Institute of Cognitive Sciences and Technologies (ISTC-CNR), Rome, Italy*

Abstract

In this work we present an empirical study where we demonstrate the possibility of developing an artificial agent that is capable to autonomously explore an experimental scenario. During the exploration, the agent is able to discover and learn interesting options allowing to interact with the environment without any assigned task, and then abstract and re-use the acquired knowledge to solve the assigned tasks. We test the system in the so-called Treasure Game domain described in the recent literature and we empirically demonstrate that the discovered options can be abstracted in an probabilistic symbolic planning model (using the PPDDL language), which allowed the agent to generate symbolic plans to achieve extrinsic goals.

Keywords

options, intrinsic motivations, automated planning

1. Introduction

If we want robots to be able to interact with complex and unstructured environments like the real-life scenarios in which humans live, or if we want artificial agents to be able to explore and operate in unknown environments, a crucial feature is to give these robots the ability to autonomously acquire knowledge that can be used to solve human requests and adapt to unpredicted new contexts and situations. At the same time, these robots should be able to represent the acquired knowledge in structures that facilitate and speed up its reuse and eventually facilitate human-robot interactions [1].

The field of Intrinsically Motivated Open-ended Learning (IMOL, [2]) is showing promising results in the development of versatile and adaptive artificial agents. *Intrinsic Motivations*

IPS-RCRA 2021: 9th Italian Workshop on Planning and Scheduling and 28th International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
gabriele.sartor@unito.it (G. Sartor); davidedezollo@gmail.com (D. Zollo); cialdea@ing.uniroma3.it (M. Cialdea Mayer); angelo.oddi@istc.cnr.it (A. Oddi); riccardo.rasconi@istc.cnr.it (R. Rasconi); vieri.santucci@istc.cnr.it (V. G. Santucci)
https://gabrielesartor.github.io/ (G. Sartor); http://cialdea.inf.uniroma3.it (M. Cialdea Mayer); http://www.istc.cnr.it/people/angelo-oddi (A. Oddi); https://www.istc.cnr.it/en/people/riccardo-rasconi (R. Rasconi); https://www.istc.cnr.it/en/people/vieri-giuliano-santucci (V. G. Santucci)
0000-0003-4370-7156 (A. Oddi); 0000-0003-2420-4713 (R. Rasconi)

(IMs, [3, 4]) are a class of self-generated signals that have been used, depending on different implementations, to provide robots with an autonomous guidance for several different processes, from state-and-action space exploration [5], to the autonomous discovery, selection and learning of multiple goals [6, 7, 8]. In general, IMs guide the agent in the acquisition of new knowledge independently (or even in the absence) of any assigned task: this knowledge will then be available to the system to solve user-assigned tasks [9] or as a scaffolding to acquire new knowledge in a cumulative fashion (similarly to what have been called curriculum learning [10]). Notwithstanding the advancements in this field, IMOL systems are still limited in acquiring long sequences of skills that can generate complex action plans. In addition to the specific complexity of the problem, this is also due to the fact that most of these systems store the acquired knowledge (e.g., contexts, actions, goals) in low-level representations that poorly support a higher-level reasoning that would guarantee a more effective reuse of such knowledge. Even if some works have shown interesting results with architectural solutions [11, 12], the use of long-term memory [13], simplified planning [14], or representational redescription [15], the need to use constructs that ensure greater abstraction and thus higher-level reasoning still seems crucial to exploiting the full potential of autonomous learning.

Within reinforcement learning (RL, [16]) the *option framework* [17] implements temporally extended high-level actions (the options), defined as triplets composed of an initiation set (the low-level states from which the option can be executed), the actual policy, and the termination conditions (describing the probability of the option to end in specific low-level states). Options can be handled at a higher level with respect to classical RL policies and, as shown within hierarchical RL (HRL), [18]), they can be chunked together to form longer chains. Moreover, HRL has been combined also with IMs to allow for different autonomous processes including, for example, the formation of skill sequences [19], the learning of sub-goals [20] and, together with deep RL techniques, to improve trajectories explorations [21]. However, despite the theoretical and operational power of this framework, options alone do not provide a complete abstraction of all the necessary elements to allow high-level reasoning and planning. Opposed to the low-level processes typical of IMOL and RL approaches, in classical planning frameworks [22] the states, actions and goals are represented as symbols that can be easily handled and composed to perform complex sequences of behaviours to solve assigned tasks. However, in symbolic planning systems, the knowledge on the domain is commonly fixed and provided by an expert at design time, thus preventing the possibility of exploiting this approach in truly autonomous systems.

Finding a bridge between autonomous approaches gathering low-level knowledge on the basis of IMs and high-level symbolic decision making is thus a crucial research topic towards the development of a new and more complete generation of artificial agents. In a seminal work, Konidaris and colleagues [23] presented an algorithm for the autonomous "translation" of low-level knowledge into symbols for a PDDL domain and then used to solve complex high-level goal-achievement tasks such as the Treasure Game (see Figure 1), by creating sequences of operators (or symbolic plans). However, in [23] the set of options to be abstracted into symbols is given to the system, thus lacking to "close the loop" between the first phase of autonomous exploration and learning, and the second phase of exploitation of the acquired knowledge.

In this work, we will present an empirical study in which we extend the results obtained in our previous research [24]. In particular, we deploy the data abstraction procedure on a

more complex domain characterized by a higher number of variables, hence focusing on the probabilistic version of the PDDL (Probabilistic Planning Domain Definition Language - PPDDL [25]), demonstrating the possibility to develop an artificial agent that is capable to autonomously explore the experimental scenario, discover and learn interesting options allowing to interact with the environment without any pre-assigned task, and then abstract the acquired knowledge for potential re-utilization to reach high-level goals. In particular, we will test the system in the so-called Treasure Game domain described in [23], where the agent can move through corridors, climb up and down stairs, interact with handles, bolts, keys and a treasure. Two different results have been achieved by empirically following the proposed approach: on the one hand, we experimentally verified how the agent is able to find a set of options and generate symbolic plans to achieve high-level goals (e.g., open a door by using a key); on the other hand, we analyzed a number of technicalities inherently connected to the task of making explicit abstracted knowledge while directly exploring the environment (e.g., synthesizing the correct preconditions of a discovered option).

The paper is organized as follows: in Section 2 we introduce the basic notation and the option framework and we describe our algorithm for automatically discovering options. Indeed, in this paper we describe a two-step learning phases: the first, to generate options from scratch and creating a preliminary action abstraction (Section 2), and the second, to produce a higher representation partitioning the options and highlighting their causal effects. The latter is described in Section 3, where we briefly describe the abstraction procedure introduced in [23]. Section 4 describes our empirical results for the Treasure Game domain; finally, in Section 5 we give some conclusions and discuss some possible directions of future work.

2. Finding Options

Options are temporally-extended actions defined as $o(I, \pi, \beta)$ [26], in which π is the policy executed, I the set of states in which the policy can run and β the termination condition of the option. The option's framework revealed to be an effective tool to abstract actions and extend them with a temporal component. The use of this kind of actions demonstrated to improve significantly the performances of model-based Reinforcement Learning compared to older models, such as one-step models in which the actions employed are the primitives of the agent [27]. Intuitively, these low-level single-step actions, or *primitives*, can be repeatedly exploited to create more complex behaviours.

In this section, we describe a possible way to discover and build a set of options from scratch. using the low-level actions available in the *Treasure Game* environment (see Figure 1). In such environment, an agent starts from its initial position (home), moves through corridors and climbs ladders over different floors, while interacting with a series of objects (e.g., keys, bolts, and levers) to the goal of reaching a treasure placed in the bottom-right corner and bringing it back home.

In order to build new behaviours, the agent can execute the following primitives: $1)go\_up$, $2)go\_down$, $3)go\_left$, $4)go\_right$, and $5)interact$, respectively used to move the agent up, down, left or right by 2-4 pixels (the exact value is randomly selected with a uniform distribution) and to interact with the closest object. In particular, the interaction with a lever changes the

state (open/close) the doors associated to that lever (both on the same floor or on different floors) while the interaction with the key and/or the treasure simply collects the key and/or the treasure inside the agent's bag. Once the key is collected, the interaction with the bolt unlocks the last door, thus granting the agent the access to the treasure.

Figure 1: The Treasure Game configuration used for the experimental analysis.

In our experiment, primitives are used as building blocks in the construction of the option, participating to the definition of π, I and β. In more details, we create new options from scratch, considering a slightly different definition of option $o(p, t, I, \pi, \beta)$ made up of the following components:

- p, the primitive used by the execution of π;
- t, the primitive which, when available, stops the execution of π;
- π, the policy applied by the option, consisting in repeatedly executing p until t is available or p can no longer be executed;
- I, the set of states from which p can run;
- β, the termination condition of the action, corresponding to the availability of the primitive t or to the impossibility of further executing p.

Consequently, this definition of option requires p, to describe the policy and where it can run, and t, to define the condition stopping its execution, maintaining its characteristic temporal abstraction. For the sake of simplicity, the option's definition will follow the more compact syntax $o(p, t)$ in the remainder of the paper.

Algorithm 1 Discovery option algorithm

```
1: procedure DISCOVER(env, max_eps, max_steps)
2:     options ← {}
3:     ep ← 0
4:     while ep < max_eps do
5:         T ← 0
6:         env.RESET_GAME()
7:         while T < max_steps do
8:             s ← env.GET_STATE()
9:             p ← env.GET_AVAILABLE_PRIMITIVE()
10:            while env.IS_AVAILABLE(p) and not (env.NEW_AVAILABLE_PRIM()) do
11:                env.EXECUTE(p)
12:                s' ← env.GET_STATE()
13:            if s ≠ s' then
14:                if env.NEW_AVAILABLE_PRIM() then
15:                    t ← env.GET_NEW_AVAILABLE_PRIM()
16:                    op ← CREATE_NEW_OPTION(p, t)
17:                else
18:                    op ← CREATE_NEW_OPTION(p, {})
19:                options ← options ∪ op
20:    return options
```

Algorithm 1 describes the process utilized to discover new options autonomously inside the simulated environment. The procedure runs for a number of episodes max_eps and max_steps steps. Until the maximum of steps of the current episode is not reached, the function keeps track of the starting state s and randomly selects an available primitive p, such that p can be executed in s (line 7-9). Then, as long as p is available and there is no new available primitives (line 10), the option p is executed, and the final state s' of the current potential option is updated. The function *NEW_AVAILABLE_PRIM* returns **True** when a primitive which was not previously executable becomes available while executing p; the function returns **False** in all the other cases. For instance, if the agent finds out that there is a ladder over him while executing the go_right option, the primitive go_up gets available and the function return **True**. In other words, *NEW_AVAILABLE_PRIM* detects the interesting event, thus implementing the surprise element that catches the agent's curiosity. For this reason, the primitive representing the exact reverse with respect to the one currently being executed is not interesting for the agent, i.e., the agent will not get interested in the go_right primitive while executing go_left. The same treatment applied to the (go_left, go_right) primitive pair is also used with the pair (go_up, go_down).

When the stopping condition of the most inner while is verified and $s \neq s'$, a new option can be generated according to the following rationale. In case the while exits because of the availability of a new primitive t in the new state s', a new option $o(p, t)$ is created (line 16); otherwise, if the while exits because the primitive under execution is no longer available, a new option $o(p, \{\})$ is created, meaning *"execute p while it is possible"* (line 18). In either case, the created option op is added to the list $options$ (line 19), which is the output of the function.

In our test scenario, the algorithm generated 11 working options (see Section 4), suitable for solving the environment, and collected experience data to be abstracted in PPDDL [25] format

successively. Consequently, as we introduced above, the agent performs two learning phases: the first, to generate options from scratch and creating a preliminary action abstraction, and the second, to produce a higher representation partitioning the options and highlighting their causal effects. The latter phase, producing a symbolic representation suitable for planning, is analyzed in the next section.

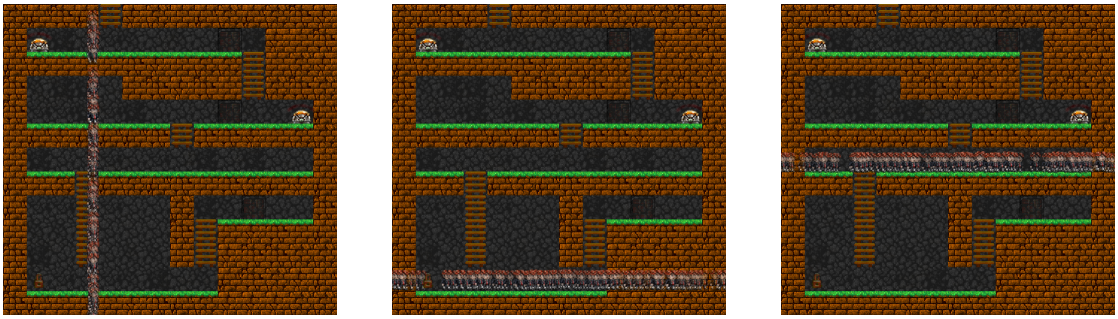
3. Abstracting options in PPDDL

In this section we provide a summary description of the knowledge abstraction procedure, in order to allow the reader to get a grasp of the rationale behind the synthesis of the PPDDL domain. A thorough description of the abstraction algorithm is beyond the scope of this paper; for further details, the reader is referred to [28].

The procedure basically executes the following five steps:

1. **Data collection**: during this step, the options learned according to Section 2 are repeatedly executed in the environment and the information about the initial and final state (respectively before and after the execution of each option) are collected. Such data are successively aggregated, and two data structures are returned from the *Data collection* phase: the *initiation data* and the *transition data*, both to be used in the following steps.
2. **Option partition**: this step is dedicated to partitioning the learned options in terms of *abstract subgoal options*. This operation is necessary as the (P)PDDL operators are characterized by a single precondition set and a single effect set; therefore, options that have multiple termination conditions starting from the same initiation set cannot be correctly captured in terms of (P)PDDL operators. As a consequence, before launching the abstraction procedure it is necessary to generate a set of options each of which is guaranteed to produce a single effect (*partial subgoal option*). This operation utilizes the *transition data* set computed in Step 1, as they capture the information about the domain segment the option modifies. Option partition is ultimately obtained by properly clustering the *transition data* through the DBSCAN algorithm ([29]) present in the scikit-learn toolkit ([30]).
3. **Precondition estimation**: this step is dedicated to learning the symbols that will constitute the *preconditions* of the PPDDL operators associated to all the options. This operation utilizes the *initiation data* set computed in Step 1, and is performed utilizing the support vector machine ([31]) classifier implementation in scikit-learn.
4. **Effect estimation**: analogously, this step is dedicated to learning the symbols that will constitute the *effects* of the PPDDL operators. The effect distribution was modelled through the Kernel density estimation ([32, 33]).
5. **PPDDL Domain synthesis**: finally, this step is dedicated to the synthesis of the PPDDL domain, characterized by the complete definition of all the operators associated to the learned options, in terms of preconditions and effect symbols.

For instance, Figure 3 depicts an example operator (`option-0`) whose action corresponds to modifying the agent's position as it has to climb up a stair to reach a new location. The operator's



(a) $Symbol\_19$. (b) $Symbol\_29$. (c) $Symbol\_30$.

Figure 2: Graphical representation of the symbols used in the *option-0* operator. $Symbol\_19$ represents the agent's x position; $Symbol\_29$ represents the agent's y position before the operator's execution, while $Symbol\_30$ represents the agent's y position after the operator's execution.

```
(:action option-0
:parameters ()
:precondition (and (symbol_19) (symbol_29))
:effect (and (symbol_30) (not (symbol_29)) (decrease (reward) 17.60))
)
```

Figure 3: Example of autonomously produced PPDDL operator whose semantics is "climb the stairs from the 1^{st} to the 2^{nd} floor".

formalization follows the standard Probabilistic PDDL (PPDDL)[1], where the precondition set is composed of the symbols $\{Symbol\_19, Symbol\_29\}$, the effect set is composed of the symbol $\{Symbol\_30\}$, and the negative effect set contains the symbol $\{Symbol\_29\}$ (note that the name of the symbols is automatically generated). The reader should also consider that the PPDDL operators returned by the abstraction procedure are *grounded*; automatically abstracting parametric PPDDL representations is beyond the scope of this work and will be the object of future work. Finally, each PPDDL operator is associated to a reward (17.60 in this case).

In order to provide the reader with some information about the meaning of the symbols that populate the previous operator, the semantics of all symbols is graphically presented in Figure 2. In particular, $Symbol\_19$ proposition in the operator's preconditions has the following semantics: "*the agent's x coordinate is vertically aligned with the stairs*", while the semantics of $Symbol\_29$ proposition is "*the agent's y coordinate positions it at a level equivalent to being at the bottom of the stairs*". From the description above, it is clear that the intersection (i.e., the logical *AND*) of the previous two symbols places the agent exactly at the bottom of the stairs. Relatively to the operator's effects, we see that $Symbol\_29$ gets negated (the agent is no longer at the bottom of the stairs) and it is replaced by $Symbol\_30$ whose meaning is "*the agent's y coordinate positions it at a level equivalent to being at the top of the stairs*". Lastly, the reader should note that $Symbol\_19$ remains valid throughout the whole execution of the operator,

[1]Despite the operator selected in this particular example does not make use of probabilities, it has been chosen due to its simplicity to exemplify the utilization of the automatically generated symbols.

and that the logical intersection of $Symbol\_19$ and $Symbol\_30$ clearly describes the situation where the agent has climbed the stairs.

4. Empirical Analysis

In this section we describe the results obtained from a preliminary empirical study, carried out by testing the Algorithm 1 in the context of the Treasure Game domain [23]. The algorithm was implemented in Python 3.7 under Linux Ubuntu 16.04 as an additional module of the *Skill to Symbols* software [2], using the *Treasure Game* Python package. As previously stated, the Treasure Game domain defines an environment that can be explored by the agent by moving through corridors and doors, climbing stairs, interacting with handles (necessary to open/close the doors), bolts, keys (necessary to unlock the bolts) and a treasure.

In our experimentation, the agent starts endowed with no previous knowledge about the possible actions that can be executed in the environment; the agent is only aware of the basic motion primitives at his disposal, as described in Section 2. The goal of the analysis is to assess the correctness, usability and quality of the abstract knowledge of the environment autonomously obtained by the agent.

The experiment starts by using Algorithm 1, whose application endows the agent with the following set of learned options (11 in total):

$$\begin{aligned} O = &\{(go\_up, \{\}), (go\_down, \{\}), (go\_left, \{\}), (go\_left, go\_up), (go\_left, go\_down), \\ &(go\_left, interact), (go\_right, \{\}), (go\_right, go\_up), (go\_right, go\_down), \\ &(go\_right, interact), (interact, \{\})\} \end{aligned} \tag{1}$$

The test has been run on an Intel I7 3.4GHz machine, and the whole process took 30 minutes. All the options are expressed in the compact syntax (p, t) described in Section 2, where *p* represents the primitive action corresponding to the action's behavior, and *t* represents the option's stop condition (i.e., the new primitive action discovered, or an empty set).

Once the set of learned options has been obtained, the test proceeds by applying the knowledge abstraction procedure described in the previous Section 3. In our specific case, the procedure eventually generated a final PPDDL domain composed by a set of 1528 operators. In order to empirically verify the correctness of the obtained PPDDL domain, we tested the domain with the off-the-shelf mGPT probabilistic planner [34]. The selected planning goal was to find the treasure, located in a hidden position of the environment (i.e., behind a locked door that could be opened only by operating on a bolt with a key) and bring it back to the agent's starting position, in the upper part of the Treasure Game environment. The agent's initial position is by the small stairs located on the environment's 5^{th} floor (up left).

The symbolic plan depicted in Figure 3 was successfully generated and, as readily observable, reaches the previous goal. The plan is composed of 33 actions, which confirmed the correctness of the proposed methodology (note that the PPDDL operators are named after their exact semantics manually, in order to facilitate their interpretation for the reader.). We also discovered

[2] We thank George Konidaris and Steve James for making both the Skills to Symbols and the Treasure Game software available.

```
1. go_down [to 5th floor]; 2. go_left [to handle];
3. interact(handle); 4. go_right [to wall];
5. go_down [to 4th floor]; 6. go_right [to handle];
7. interact(handle); 8. go_left [to key]; 9. interact(key);
10. go_right [to stairs]; 11. go_down [to 3rd floor];
12. go_left [to stairs]; 13. go_down [to 1st floor];
14. go_left [to bolt]; 15. interact(bolt, key);
16. go_right [to wall]; 17. go_up [to 2nd floor];
18. go_right [to treasure]; 19. interact(treasure);
20. go_left [to stairs]; 21. go_down [to 1st floor];
22. go_left [to bolt]; 23. go_right [to stairs];
24. go_up [to 3rd floor]; 25. go_right [to wall];
26. go_left [to stairs]; 27. go_up [to 4th floor];
28. go_right [to handle]; 29. interact(handle)
30. go_left [to stairs]; 31. go_up [to 5th floor];
32. go_left [to stairs]; 33. go_up [home];
```

Figure 4: Plan generated by the mGPT planner with our autonomously synthesized PPDDL domain. The goal of the plan is to (i) find the treasure located behind a closed door that can only be opened by finding a key and then using it to unlock a bolt, and (ii) bringing it to the agent's initial position.

a number of difficulties inherently connected to the task of explicitly abstracting the knowledge by means of a direct exploration of the environment. One consequence of such difficulties is evident from the quality of the plan outlined above. In fact, it is clear that the plan is not optimal, as the agent performs sometime useless actions, such as going left to the bolt and then right to the stairs (22. $go\_left[to\ bolt]$ and 23. $go\_right[to\ stairs]$, respectively) instead of directly executing a $go\_left[to\ stairs]$. Another examples of redundant actions are $25.\ go\_right[to\ wall]$, $26.\ go\_left[to\ stairs]$ instead of directly executing a $go\_right[to\ stairs]$. It can be easily seen that the optimal plan is composed of 31 actions.

The previous analysis is still ongoing work. In this paper, we are presenting the encouraging results obtained so far, though we have observed that a number of improvements are worth being studied. One observation can be made about the quality of the obtained PPDDL domain; despite we have demonstrated that such domain can be successfully used for automated planning, we have also observed that it contains a number of infeasible operators (i.e., characterized by mutually conflicting preconditions) as well as operators characterized by a high failure probability. Of course, the presence of such operators does not hinder the feasibility of the produced plan (i.e., the former operators will always be discarded by the planner, while the latter will at most make the planning process more demanding, thus decreasing the probability of obtaining an optimal solution) yet, further work must be done to arrive to "crisper" domain representations.

In this respect, there are at least two research lines to investigate. The first line entails the study of different fine-tuning strategies of all the parameters utilized in the previously mentioned Machine Learning tools (such as DBSCAN, SVM, Kernel Density Estimator) involved in the knowledge-abstraction process. The second line is about analyzing the most efficient environment exploration strategy used to collect all the transition data that will be used for

the classification tasks that are part of the abstraction procedure, as both the quantity and the quality of the collected data may be essential at this stage.

5. Conclusions and Future Work

In this paper we tested an option discovery algorithm driven by *intrinsic motivations* for an agent operating in the Treasure Game domain [23]. We experimentally demonstrated that the discovered options can be abstracted in an probabilistic symbolic planning model (using the PPDDL language), which allowed the agent to generate symbolic plans to achieve extrinsic goals. One of the possible direction of future work will be the exploration of innovative iterative procedures to incrementally refine [35] the generated PPDDL model.

References

[1] M. Ebrahimi, A. Eberhart, F. Bianchi, P. Hitzler, Towards bridging the neuro-symbolic gap: Deep deductive reasoners, Applied Intelligence (2021) 1–23.

[2] V. G. Santucci, P.-Y. Oudeyer, A. Barto, G. Baldassarre, Intrinsically motivated open-ended learning in autonomous robots, Frontiers in neurorobotics 13 (2020) 115.

[3] P.-Y. Oudeyer, F. Kaplan, V. Hafner, Intrinsic motivation systems for autonomous mental development, IEEE transactions on evolutionary computation 11 (2007) 265–286.

[4] G. Baldassarre, M. Mirolli, Intrinsically Motivated Learning in Natural and Artificial Systems, Springer Science & Business Media, 2013.

[5] M. Frank, J. Leitner, M. Stollenga, A. Förster, J. Schmidhuber, Curiosity driven reinforcement learning for motion planning on humanoids, Frontiers in neurorobotics 7 (2014) 25.

[6] A. Baranes, P.-Y. Oudeyer, Active learning of inverse models with intrinsically motivated goal exploration in robots, Robotics and Autonomous Systems 61 (2013) 49–73.

[7] V. G. Santucci, G. Baldassarre, M. Mirolli, Grail: A goal-discovering robotic architecture for intrinsically-motivated learning, IEEE Transactions on Cognitive and Developmental Systems 8 (2016) 214–231.

[8] C. Colas, P. Fournier, M. Chetouani, O. Sigaud, P.-Y. Oudeyer, Curious: intrinsically motivated modular multi-goal reinforcement learning, in: International conference on machine learning, PMLR, 2019, pp. 1331–1340.

[9] K. Seepanomwan, V. G. Santucci, G. Baldassarre, Intrinsically motivated discovered outcomes boost user's goals achievement in a humanoid robot, in: 2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob), 2017, pp. 178–183.

[10] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th annual international conference on machine learning, 2009, pp. 41–48.

[11] S. Forestier, R. Portelas, Y. Mollard, P.-Y. Oudeyer, Intrinsically motivated goal exploration processes with automatic curriculum learning, arXiv preprint arXiv:1708.02190 (2017).

[12] V. G. Santucci, G. Baldassarre, E. Cartoni, Autonomous reinforcement learning of multiple interrelated tasks, in: 2019 Joint IEEE 9th international conference on development and learning and epigenetic robotics (ICDL-EpiRob), IEEE, 2019, pp. 221–227.

[13] J. A. Becerra, A. Romero, F. Bellas, R. J. Duro, Motivational engine and long-term memory coupling within a cognitive architecture for lifelong open-ended learning, Neurocomputing 452 (2021) 341–354.
[14] G. Baldassarre, W. Lord, G. Granato, V. G. Santucci, An embodied agent learning affordances with intrinsic motivations and solving extrinsic tasks with attention and one-step planning, Frontiers in neurorobotics 13 (2019) 45.
[15] S. Doncieux, D. Filliat, N. Díaz-Rodríguez, T. Hospedales, R. Duro, A. Coninx, D. M. Roijers, B. Girard, N. Perrin, O. Sigaud, Open-ended learning: a conceptual framework based on representational redescription, Frontiers in neurorobotics 12 (2018) 59.
[16] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.
[17] R. S. Sutton, D. Precup, S. Singh, Intra-option learning about temporally abstract actions, in: Proc. 15th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA, 1998, pp. 556–564.
[18] A. G. Barto, S. Mahadevan, Recent advances in hierarchical reinforcement learning, Discrete event dynamic systems 13 (2003) 41–77.
[19] C. M. Vigorito, A. G. Barto, Intrinsically motivated hierarchical skill learning in structured environments, IEEE Transactions on Autonomous Mental Development 2 (2010) 132–143. doi:10.1109/TAMD.2010.2050205.
[20] J. Rafati, D. C. Noelle, Learning representations in model-free hierarchical reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, 2019, pp. 10009–10010.
[21] T. D. Kulkarni, K. Narasimhan, A. Saeedi, J. Tenenbaum, Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation, Advances in neural information processing systems 29 (2016) 3675–3683.
[22] D. Nau, M. Ghallab, P. Traverso, Automated Planning: Theory & Practice, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
[23] G. Konidaris, L. P. Kaelbling, T. Lozano-Perez, From skills to symbols: Learning symbolic representations for abstract high-level planning, Journal of Artificial Intelligence Research 61 (2018) 215–289. URL: http://lis.csail.mit.edu/pubs/konidaris-jair18.pdf.
[24] A. Oddi, R. Rasconi, V. G. Santucci, G. Sartor, E. Cartoni, F. Mannella, G. Baldassarre, Integrating open-ended learning in the sense-plan-act robot control paradigm, in: ECAI 2020, the 24th European Conference on Artificial Intelligence, 2020.
[25] H. Younes, M. Littman, PPDDL1.0: An Extension to PDDL for Expressiong Planning Domains with Probabilistic Effects, Technical Report, Carnegie Mellon University, 2004. CMU-CS-04-167.
[26] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 1998.
[27] R. S. Sutton, D. Precup, S. Singh, Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning, Artif. Intell. 112 (1999) 181–211. URL: http://dx.doi.org/10.1016/S0004-3702(99)00052-1. doi:10.1016/S0004-3702(99)00052-1.
[28] G. Konidaris, L. P. Kaelbling, T. Lozano-Perez, From skills to symbols: Learning symbolic representations for abstract high-level planning, Journal of Artificial Intelligence Research 61 (2018) 215–289. doi:https://doi.org/10.1613/jair.5575.
[29] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proceedings of the Second International

Conference on Knowledge Discovery and Data Mining, KDD'96, AAAI Press, 1996, p. 226–231.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[31] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (1995) 273–297. URL: https://doi.org/10.1023/A:1022627411411. doi:10.1023/A:1022627411411.

[32] M. Rosenblatt, Remarks on some nonparametric estimates of a density function, The Annals of Mathematical Statistics 27 (1956) 832–837. URL: http://www.jstor.org/stable/2237390.

[33] E. Parzen, On estimation of a probability density function and mode, The Annals of Mathematical Statistics 33 (1962) 1065–1076. URL: http://www.jstor.org/stable/2237880.

[34] B. Bonet, H. Geffner, Mgpt: A probabilistic planner based on heuristic search, J. Artif. Int. Res. 24 (2005) 933–944.

[35] Y. Hayamizu, S. Amiri, K. Chandan, K. Takadama, S. Zhang, Guiding robot exploration in reinforcement learning via automated planning, Proceedings of the International Conference on Automated Planning and Scheduling 31 (2021) 625–633. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/16011.