Ethology-Inspired Advanced Problem Solving Mechanism

Jaroslav Vítků[1]

[1]Dept. of Cybernetics, Czech Technical University, Technická 2, 166 27 Praha, Czech Republic

vitkujar@fel.cvut.cz

Abstract. *Presented topic is from the research field called Artificial Life (ALife), but contributes also to the field of Artificial Intelligence (AI). We have built a new architecture of autonomous agent. Our agent is capable of fully autonomous learning. The course of learning is similar to newly born animal. The agent learns everything itself, from basic tasks towards the more complex knowledge. This newly gained knowledge is stored in the hierarchy of abstract actions. The main innovation in our approach can be seen in our new way how to implement hybrid hierarchical planner. This planner combines Reinforcement Learning (RL) and classical planning. The resulting hierarchical planner is domain independent, because it is capable to adapt itself to the given domain. We have tested this on the artificial agent within the virtual simulation environment.*

Keywords

Agent, Behavior, Planner, Hybrid, Reinforcement, Learning

1. Introduction

Originally, the scientists working on the robotic research worked with real robots. But the scientists that are interested in the higher (more complex) behavior do not want to deal with problems like a noisy sensory data or uncertain results of actions. This is the main reason for extensive use of simulations. The term ”robot” is replaced with the term ”agent”, this agent has often only virtual embodiment and operates in some virtual environment.

The agent (robot) has to fulfill several main conditions in order to be successful in his task. These are mainly:

- ability to learn
- ability to adapt to the environment changes
- ability to deal with the complex problems
- ability to react quickly enough
- ability to behave rationally - to maximize its future benefit

Some of these conditions are contradictory. In order to behave rationally, the agent can be equipped e.g. with the planning system and therefore can be able to create complex plans. But if the agent will use too complex action selection mechanism, the reaction time can be slower than the environment dynamics. In this situation the agent's plans will be based on old information about the world.

We are interested in the design of artificial agent architectures. The main goal of this research is to build such a system, which will be capable of higher decision making, but will be able also to deal with the environment dynamics. The ideal system has to be able to learn efficiently and to solve complex problems using this knowledge with minimum need of computation power.

In order to make the decision systems more efficient, some approaches can be used, e.g. the decomposition of the original problem into smaller pieces. For overview of hierarchical planners see [1]. Several state-of-the-art architectures use the hierarchical decomposition, for example a hierarchical RL or a hierarchical planning. The main common drawback of these approaches is a need of a priori knowledge. These systems often need predefinition of the action hierarchy. This means that nowadays hierarchical planners are either domain dependent or domain configurable.

In the *Hierarchical Task Network* (HTN) planning [2] the plan is composed of set of tasks. Task is in our terminology an *abstract action* and describes set of more primitive actions that should be done. Each task has predefined set of methods that may, or may not, be applicable at the current state. HTM planning uses problem reduction: it recursively decomposes tasks into subtasks.

Nowadays hierarchical planners, as is e.g. *Simple Hierarchical Ordered Planner* (SHOP) [3], are domain independent. But these planners need a domain description on its input. This main disadvantage has been also studied, there are some systems that are able to partially learn the knowledge about given domain needed by these hierarchical planners [4].

Another possible approach to decomposition of the problem into the smaller, manageable, pieces is in a hybrid system. In more complex environments (such is e.g. robot in a real environment) the planning system is not convenient to solve everything. The planning system can create complex

and abstract plans, while low-level actions (as an obstacle avoidance, control of current speed etc) can be efficiently solved using some another subsystem [5]. We have selected the RL for dealing with these lower-level actions. In such a resulting system the tasks in RL represent some actions that can be handled by the planning system. Similar architectures, which use this principle, can be found in [6], [7]. However, the first architecture needs domain description in form of plan library and the second one uses different representation of states and plans.

We present our domain independent planner with autonomous learning system which gains all its knowledge about the problem structure itself by interaction with the environment. It discovers dependencies by method trial and error. If new, potentially useful, action is discovered, the system tries to learn how to perform this action as efficiently as possible.

2. Theory

Here we would like to introduce the two main principles used in this work: a planning and a reinforcement learning systems. For planning in our agent we decided to use well known *Stanford Research Institute Problem Solver* (STRIPS) language [8]. As a core for RL we used a Q-learning algorithm. The agent is able to use multiple RL engines in an autonomously created hierarchy of abstract actions, in the section 2.2 we will briefly describe the main principles of creation and use of this hierarchy.

2.1. Reinforcement Learning

The key and basic principle used in our architecture is the RL, learning method inspired in behaviorist psychology, where an agent learns which actions should take in the given state in order to maximize its future reward from his environment. The basic idea is the same with a dynamic programming. RL is very general approach and the only main disadvantage is fact that an environment formulated as a *Markov Decision Process* (MDP) is required [13]. Interaction with the MDP environment means that each discrete time step t an agent perceives the finite set of states $|S|$ and is able to execute finite set of actions $|A|$. After executing the selected action u_t in the state x_t, the environment responds with a *reward* or *punishment* $r(x_t, u_t)$ and a *new state* $x_{t+1} = T(x_t, u_t)$ is generated. The next-state function T and the reinforcement r function are not known to the agent, the important property of MDP is that the transition function T is based only on the actual state and executed action.

The goal of RL is to choose actions in response to states so that the reinforcement is maximized, this means that an agent is learning policy: a mapping from states to actions. There are several possible ways to implement a learning process; we used a Q-learning algorithm. In this type of RL an agent learns to assign values to state-action pairs a Q-value function, the value of this function is sum of all future events. While immediate rewards are more important, we will use discounted cumulative reinforcement, where future reinforcements are weighted by value $\gamma \in \langle 0, 1\rangle$. The equation (1) represents the optimal Q-value function.

This Q-value function can be represented for example as $(n+1)$-dimensional matrix containing numbers. In this case n is the number of variables and size of the last dimension corresponds to the number of primitive actions available. Basically: the agent learns how "useful" (in context of the future reward) the execution of particular action in the given state is. This is done by filling the Q-value matrix with numbers, the higher the number is the more useful is the action in the state.

$$Q^*(x_t, u_t) = r(x_t, u_t) + \gamma \max_{u_{t+1}} Q^*(x_{t+1}, u_{t+1}) \qquad (1)$$

At each step, the agent executes one action (selected based on the discounted Q-value function) receives reinforcement and updates Q-value of a given state-action pair in the table according to the off policy *Temporal Difference* (TD) control - equation (2), where $\alpha \in \langle 0, 1\rangle$ is the learning rate.

$$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha\big[r(x_t, u_t) + \gamma Q(x_{t+1}, u_{t+1}) - Q(x_t, u_t)\big] \qquad (2)$$

In order to get a good trade-off between exploration and exploitation, an action selection mechanism uses some kind of randomization, instead of pure greedy method. A system that implements this entire mechanism will be called *return predictor*.

2.2. Hierarchical Reinforcement Learning

The classical RL approach has one disadvantage: size of look-up table (matrix) for storing Q-values grows very fast with an environment complexity. This means that in slightly more complex environment the Q-value matrix can have too many dimensions and the learning convergence can be very slow. In order to beat the course of dimensionality, *Hierarchical RL* (HRL) was introduced. We define *Decision space* (D) as some defined subset of all possible actions and environment states. The return predictor operates over this decision space. This means that the return predictor has only some bounded knowledge about the whole problem, and is able to select only from some actions. The decision space can be then seen as an *abstract action*, or a *behavior*.

Each such decision space represents some behavior, e.g. opening the door. In case of the hierarchical RL these behaviors can be seen as other actions. This means that some decision spaces can contain actions which are represented by another decision space (composed behavior). If such composed action is selected, the corresponding behavior (learned in the Q-value matrix) is executed. This decomposition of original decision space to hierarchy of smaller ones can be seen in the Fig.1.

For completness: we use the MAXQ value function decomposition [14], where a received rewards are distributed into the hierarchy of decision spaces D_i according to a factorization function. Roughly speaking: decision spaces, which contributed more to receiving of the particular reinforcement, are rewarded more.

2.3. Ethology-Inspired Autonomous Discovery of Action Hierarchy

We built on an architecture called *"Hierarchy, Abstraction, Reinforcements, Motivations Agent Architecture"* (HARM) [10]. The main advantage of this approach is the fact that this agent is able to build the hierarchy of actions without need of any predefined knowledge. The hierarchy is build based on the various types of reinforcements received during the agent's life.

We use *motivation* $m(D_i)$ which defines "how much" the agent wants to execute particular action (corresponding to a decision space D_i). The resulting *utility* of action (for a decision space on the top of the hierarchy) is defined in the following equation:

$$\varphi_{D_i}(s,a) = m(D_i)Q_i(s,a). \tag{3}$$

Utility of actions with some "parent" action are then composed of utility to execute the action and motivations provided by all its parents. Motivations originate in the agents physiology. The physiology is represented by predefined *physiological state space*. This state space contains variables with some defined dynamics, where each variable corresponds to some of agents needs. For example: the variable *battery_charge_state* produces need (and therefore a motivation) for recharging the battery. This physiology is connected to the RL in the following fashion: if some physiological variable is actively moved towards the optimal condition, the agent receives reinforcement.

During his life, the agent continuously learns how to satisfy these motivations, that is e.g.: how to recharge the battery. If the corresponding reinforcement is received, a new decision space (behavior) is created. This space contains only some subset of known environment variables and actions. The algorithm presented in Kadleček's thesis selects which variables and actions which should be in the particular decision spaces online during agent's life. In the Fig.2 there is depicted the example of autonomously created hierarchy of RL actions during the agent's life. These behaviors were identified based on the reinforcements received from the agents physiology. The physiological state space contained here three main variables: *water, food* and *obligation need*.

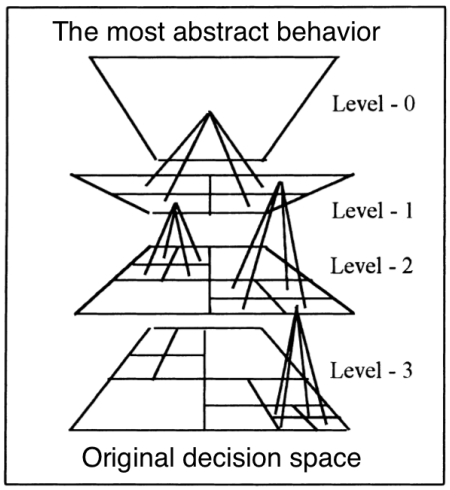

Fig. 1. Example of decision-space decomposition. Original decision space (on the bottom-level 3 here) is consecutively decomposed into smaller spaces. On the top there is a most abstract decision space, which represents the most complex behavior learned so far. Picture partially taken from [12].

Among several other improvements of the original architecture, we added the ability *to autonomously create intentions* during agent's life. If the agent finds out that he actively changed some environment variable (that is: discovered potentially useful behavior), the new intention variable is created. Intention is connected to the newly created decision space. Intentional state space has similar dynamics as the physiological one. It means that from this moment, the intention variable motivates learning this new behavior. Because of this system, the agent behaves similarly to newly born animal. The agent explores surrounding environment, if some potentially useful thing is discovered, the agent plays with it and tries to learn new behavior. This new behavior can be later used for something useful.

2.4. Planning

In this section we will briefly describe the STRIPS language. It is formally represented as a quadruple $\langle P, O, I, G\rangle$. The P is the set of conditions expressed by propositional variables describing the world state, I is the description of initial state and G is description of properties which are fulfilled in a goal state(s). O is the set of operators - actions, each operator consists of the quadruple $\langle \alpha_s, \beta_s, \gamma_s, \delta_s\rangle$. The elements α_s and β_s describe the constraints when the action can be applied, that is: describe which conditions must be

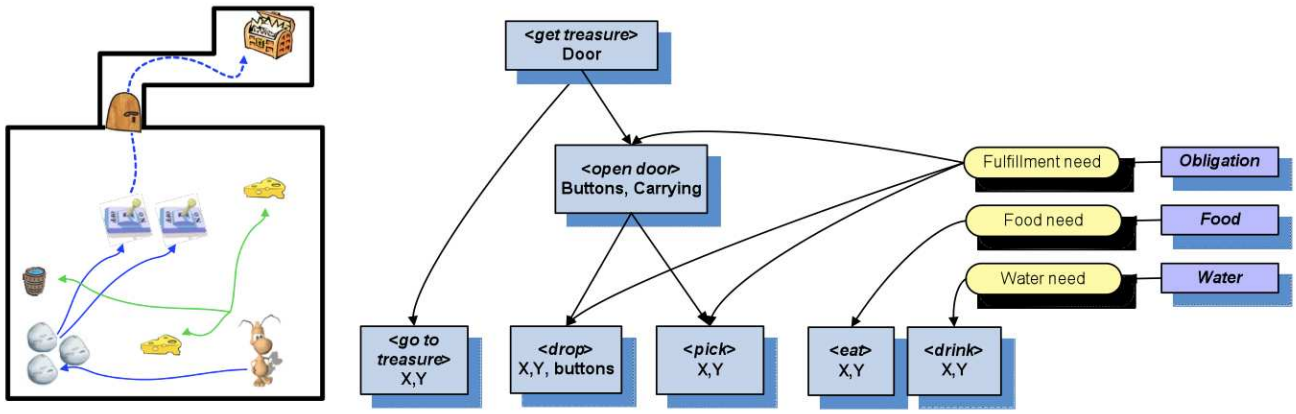

Fig.2. The treasure problem: agent has to reach the treasure. In order to open the door he must put the stones onto the switches in the specified order. Agent has to follow two physiological needs: hunger and thirst. On the right side there is depicted the result of creating the action hierarchy, where in each decision space is name of the behavior and a list of variables that are considered by its return predictor. Picture taken from [10].

true and which false in the given situation. The elements γ_s and δ_s describe action effects after its application, that is: which propositional variables will become true and which false. The current state of the world is described by a binary vector, where operators change values of bits on a specified position in a specified manner. The plan is a sequence of applicable operators that consecutively transform the description of initial state towards the state which fulfills the goal conditions.

2.5. Fusion of Planning and Reinforcement Learning

As a typical planner, our STRIPS-based planner requires on its input three main things: description of the current state, description of a goal state and a set of possible actions. Our latest architecture, presented in [9] is able to automatically infer the description of current state and set of possible actions from the hierarchy of RL abstract actions (described in the section 2.2). We will describe this process of generating this world description in more details here. This concept will be explained on the example behavior. In the selected experiment, our agent was put into an unknown map. THe lights can be controlled from some position on this map. The agent is able to move in four directions and to execute the action *press_button*. After successful changing the *lights_state* (either switching *on* or *off*), new intention to turn *on*/*off* the lights is created and connected to newly created decision space. This decision space is depicted in the Fig.3. It can be seen that the agent successfully learnt how to approach to the position of button and to execute primitive action *press* on the right position.

In this particular example, the agent learnt how to change the *lights_state*. It means that the variable *lights_state* is not contained in the decision space, but it is so-called a **main variable**. Change of this main variable always causes receiving the reinforcement. In the current situation we suppose that only binary variables can become main variables. After assuming this, each RL decision space can be seen as a primitive action in the STRIPS language. This primitive action can change the main variable in two directions (set to 1 or 0). The state description for the STRIPS language contains only one binary variable, for this action. In this case, the two primitive actions O_1 (turn on the lights) and O_2 (turn off the lights) can be autonomously created in the form of STRIPS operators $\langle \alpha_s, \beta_s, \gamma_s, \delta_s \rangle$ as follows:

$$O_1 = \langle -, 1, 1, - \rangle \quad (4)$$

$$O_2 = \langle 1, -, -, 1 \rangle \quad (5)$$

where "1" denotes the first binary variable and "-" denotes no effect or no condition. This principle is described graphically in the Fig.3. By combining these "main variables" from multiple RL decision spaces, the STRIPS-based planner consecutively builds a description of the world. This description is represented as a vector of these main variables. This description of the world contains **only useful variables**, which even more **dramatically reduces the decision space for the planner**. Here we showed how the set of actions and description of world state can be obtained autonomously. In this situation, the user is allowed to describe the goal state. The planner tries to solve the given task by means of available autonomously discovered actions. In the following paragraph we will describe the course of plan execution.

If the planner decides to execute the action, it sets the intention to execute the corresponding behavior to the maximum. The RL engine then executes the behavior based on the knowledge stored in the Q-value matrix. After successful

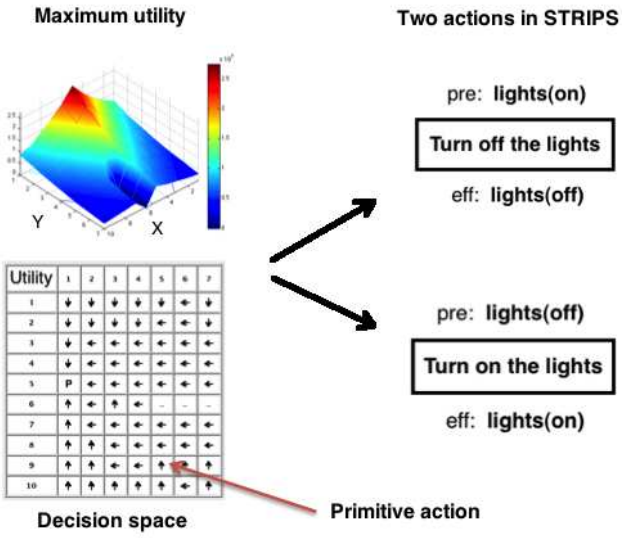

Fig. 3. Example of automatic conversion of behavior represented in form of RL into two primitive actions in the STRIPS language. This is example of *switching the lights* by pressing the switch in some defined part of the map. On the left there is the graph representing the maximum utility based on the agents position in the map. On the right there are two automatically generated primitive actions in the STRIPS language.

execution of behavior the motivation falls towards zero, this means that the planner can order execution of a next action in the plan. In this manner the planning system is able to compose deliberatively composed sequence of actions in order to reach the desired goal state. These actions can be primitive, or it can be complex behaviors which are composed of several sub-actions in the RL action hierarchy.

3. Conclusion

We simulate the the life of our agent in a virtual environment, which is developed originally for Multi-Agent Contest. Agent reads the information about the surrounding world by receiving data in XML format over the TCP/IP. After sensing the environment, the agent responds with one action selected from the predefined set of primitive actions. We showed on various experiments that this new agent architecture is able to deal with unknown and dynamic environment, to learn how to survive and even how to be useful to its owner. There are several disadvantages that should be dealt with, there are mainly: a need of MDP environment or the fact that only binary main variables can be considered now.

The main advantages of our approach are mentioned here: the hierarchy of RL actions decomposes the original, potentially huge, decision space into set of small decision spaces. This approach dramatically reduces the size of decision space and enables our agent to learn and to successfully operate in unknown environments. The other main advantage of our approach is in the fact that the planner builds the description of the world only from the variables that are somehow potentially useful. This means that the information filtered through the hierarchy of RL actions are filtered further before passing to the planner. This enables the agent to ”think” (to consider them while building the plan) only those important actions and variables, while not worrying about the rest. The planner operates on a high-level representation of the problem. The lower-level parts of the problem are solved online using the RL approach.

Compared to other similar architectures (which use some hierarchical or hybrid principles), our agent needs practically *no a priory knowledge about the problem* domain. Things that have to be redefined are only: set of primitive actions (actuators), agents needs (physiology), and a sensory system. The rest is solved and learnt online during the interaction with the environment.

Acknowledgements

This research has been funded by the Dept. of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague and Centre for Applied Cybernetics under Project 1M0567.

Author wants to thank to his supervisor Prof. Assoc. Pavel Nahodil for his time, significant help and guidance throughout the entire process of development and research.

References

[1] WILKINS, D. E., *Hierarchical Planning: Definition and Implementation*, Technical Note 370. AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, Dec 1985.

[2] NAU, D., SMITH S.J.J., and EROL, K., Control Strategies in HTN Planning: Theory versus Practice. *In AAAI-98/IAAI-98 Proceedings*, pp. 1127-1133, 1998.

[3] NAU, D., CAO, Y., LOTEM, A., and MUÑOZ-AVILA H., SHOP: Simple Hierarchical Ordered Planner. *In IJCAI-99*, pp. 968-973, 1999.

[4] ILGHAMI, O., NAU, D., MUÑOZ-AVILA H., and AHA D. W., *CaMeL: Learning methods for HTN planning*. AIPS-02, 2002.

[5] NILSSON, N.J., Teleo-reactive programs for agent control, *Journal of Artificial Intelligence Research 1*, pp.139-158, 1994.

[6] RYAN M. and PENDRITH, M., RL-TOPs: An Architecture for Modularity and Re-Use in Reinforcement Learning, *In Proceedings of the Fifteenth International Conference on Machine Learning*, San Francisco, CA, USA, pages 481-487, 1998.

[7] ROSS, M., *Hierarchical Reinforcement Learning: A Hybrid Approach*, PhD Thesis, The University of New South Wales, School of Computer Science and Engineering, 2002.

[8] FIKES, R.E., NILSSON, N.J., Strips: A new approach to the application of theorem proving to problem solving *Artificial Intelligence*, Vol. 2, No. 3-4., pp. 189-208, 1971.

[9] Vítků, J.: *An Artificial Creature Capable of Learning from Experience in Order to Fulfill More Complex Tasks*, CTU in Prague, FEE, Diploma Thesis supervised by Nahodil, P., pp.142, 2011.

[10] Kadleček, D.: *Motivation Driven Reinforcement Learning and Automatic Creation of Behavior Hierarchies*, CTU in Prague, FEE, PhD Thesis supervised by Nahodil,P., pp.143, 2008.

[11] Nahodil, P., Kadlecek, D.: Adopting Animal Concepts in Hierarchical Reinforcement Learning and Control of Intelligent Agents, *In Proc. of 2nd IEEE/RAS-EMBS Intern. Conf. on Biomedical Robotics and Biomechatronics*, BioRob, pp.122-131, Scottsdale, 2008.

[12] Kbir, M. Ait,Maalmi, K., Benslimane, R. and Benkirane, H.: Hierarchical fuzzy partition for pattern classification with fuzzy if-then rules, *Pattern Recogn. Lett.*, vol 21, pp.503-509, Elsevier Science Inc., New York, NY, USA2000.

[13] Bellman, R., *A Markovian Decision Process*, Indiana Univ. Math. J. 6: pages 679–684, 1957.

[14] Dietterich, T.G., Hierarchical reinforcement learning with the maxq value function decomposition, *Journal of Artificial Intelligence Research 13*, pp.227–303, 1998.

About Authors...

Jaroslav VÍTKŮ was born in Prague, Czech Republic, graduated at CTU in Prague at 2011. His Diploma Thesis was awarded by Price of Dean. Currently he is a PhD student in the CTU, FEE, Department of Cybernetics. His research interest includes Behavioral Robotics, Cognitive Science, Biologically Inspired Algorithms and Artificial Life in common. His e-mail address is: vitkujar@fel.cvut.cz