Technical Report

IRI-DT 0508

# "On-Line Learning of Macro Planning Operators using Probabilistic Estimations of Cause-Effects"

Alejandro Agostini
Florentin Wörgötter
Enric Celaya
Carme Torras

October 2008

Institut de Robòtica i Informàtica Industrial

# On-line Learning of Macro Planning Operators using Probabilistic Estimations of Cause-Effects

Agostini A., Wörgötter F., Celaya E., Torras C.

**Abstract.** In this work we propose an on-line learning method for learning action rules for planning. The system uses a probabilistic approach of a constructive induction method that combines a beam search with an example-based search over candidate rules to find those that more concisely describe the world dynamics. The approach permits a rapid integration of the knowledge acquired from experience. Exploration of the world dynamics is guided by the planner, and – if the planner fails because of incomplete knowledge – by a teacher through action instructions.

## Introduction

In the last years service robot applications are widening with the improvements in computer science techniques and the development of new technologies. These applications range from simple chores, like vacuum cleaners, to complex tasks requiring complex cognitive capabilities, similar to those forming the human capacity of performing complex tasks in real environments.

In this work we face the problem of decision making for a multitasking service robot embedded in a human environment that should rapidly learn to perform tasks in an on-line way, and without any previous knowledge of the world dynamics or the tasks to be performed.

The selection of which paradigm to apply relies on the characteristic of the problem faced. In general, there are two alternative approaches used to build an intelligent agent: the deliberative and the reactive approaches. The deliberative approach is based on the principle of rationality [1], and involves planning techniques in which actions are executed in accordance to plans built after reasoning about possible sequences to reach the goal. In reactive approaches [2] actions are not longer driven by the rationality principle, but just executed from already coded behaviours that lead to the goal without deliberation.

Both paradigms have drawbacks and advantages. In planning approaches the amount of deliberation could be very large even for the simplest kind of problem, mainly when the environment has complex dynamics. Additionally, deliberative techniques require a model of the dynamics of world, which in many cases is extremely difficult to code. This is not usually the case for reactive techniques which learn dynamics of the world automatically. Nevertheless, in reactive approaches drawbacks are caused by the limitation of their applicability to single goal tasks and by the requirement of large experience to reach an acceptable convergence. We propose a decision making technique that combines both approaches with the aim of diminishing the drawbacks of either one of them using the advantages of the other.

In general, reactive approaches have proved to be valid for many low-level repetitive tasks, while deliberative approaches are more suitable for high-level tasks where specified goals are rarely repeated. Usually, it is observed that high-level tasks can be performed by a succession of simpler low-level tasks.

We develop a method to learn on-line action rules that permit to reactively perform low-level tasks when they are executed as planning operators of high-level plans. These rules could significantly relieve the amount of deliberation as they might merge repetitive sequences of actions, or plans found with large computational cost. One remarkable aspect of the method is that there is no need for codification of the world dynamics as they are learned automatically while acting.

The learning method generates action rules using a constructive induction approach that combines a beam search with an example-based search [3] over candidate action rules to find those that more concisely describe the world dynamics. The approach permits a rapid integration of the knowledge acquired from experience. Exploration of the world dynamics is guided by the planner, and – if the planner fails because of incomplete knowledge – by the teacher through action instructions.

It is very simple for humans to know which action to perform in a situation given a plain task. But it could be much more complicated to explain a priori all the sequences of actions that should take place in all the possible situations. We take benefit of the human capabilities of knowing which action to perform in currently observed situations to efficiently generate knowledge for decision making in a multitask robot.

The idea of learning cause-effects is based on Piaget's theory of cognitive development [4] which claims that children gradually acquire knowledge of cause-effect relations by repeatedly executing processes and sequencing actions to reach goals. As we will see, action rules are created using learned cause-effect relations observed from experienced situations after actions executions. Cause-effects are not only expressed as a unique set of conditions that afford changes, but also as multiple set of conditions that have different chances to afford a change.

As suggested in previous works [5], [6] the explicit coding of the world conditions and actions through rules and cause-effects presented above is one possible instantiation of the concept of object-action complexes (OACs) [7], which is considered as the main block for building cognitive systems for a complex service robot [8]. In a few words, the OAC concept claims that the world contains undistinguished “things” meaningless for the agent that only become meaningful “objects” through actions and tasks, where the objects are described by the properties relevant for the fulfilment of the final desired outcome through the action. We believe that the contribution of this work is another step toward a formalization of the OAC concept as the probabilistic approach of cause-effect could be seen as how likely a thing is an object, where an object description now is not restricted to a unique set of conditions but multiple set of conditions that have different chances of affording the object functionality.

The learning module is embedded in a more general decision making system containing a planning module that uses the learned action rules as planning operators. The global system is based on a previous study [5] where the teacher guides the exploration of actions and also explains the world dynamics at the level of currently experienced cause-effects. The current contribution is an extension of [5] where dynamics of the world are automatically learned while performing the required tasks.

Other approaches propose a decision-making system that permits fast agent responses to new situations using reactive layers while the deliberative layers generate behaviors used later by the reactive modules [9]. Some let the low-level action control to be driven by reactive behaviors, which are selected or modulated by a higher deliberative layer [10], [11]. Finally, others focus mainly on the generation of behaviors such as macro-actions [12], primitive behaviors [13], or activation rules [14], which store sequences of actions frequently used or difficult to calculate, to use them later as macro planning operators in a deliberative system.

In any of the previous cases a large amount of computation is usually required due to the need of exploring different acting behaviors to select the one suitable for the task. The problem turns to be more complicated if the robot has no previous knowledge of the world dynamics and should perform learning while predicting what would occur with different behaviors. Incomplete knowledge has been tackled using techniques like incomplete planning [15], learning planning operators [16], [17], [18] or policy learning [19], but the drawback of computational complexity derived of the application of AI techniques is still not surmounted.

## System structure

As mentioned before, the decision making system has two main modules: a learning module that provides action rules in the form of planning operators, and a planning module that uses the learned operators. The action rules learned have STRIPS like structure [20] suitable to be used by any planner that can deal with them.

A general overview of the method is the following. Given a goal, the agent tries to generate a plan using the existing rules. If the planner fails to return a behavior, as a consequence of an incomplete knowledge, the agent asks the teacher about which action or actions to perform. The agent executes every instructed action and generates what we denote as a *probabilistic cause-effect* used to estimate the probability of the occurrence of changes under different sets of conditions. A probabilistic cause-effect is the main structure for learning and generation of STRIPS like cause-effects for planning purposes.

When the agent is able to find a plan then it executes and evaluates it at the level of each cause-effect. During cause-effects execution relevant experiences are stored as example situations which are used to learn the conditions that afford desired changes. For instance, if any of the outcomes obtained after a cause-effect execution is different from the one expected, a fact referred to as *surprise*, the experienced situation is memorized as a negative example for obtaining the expected change.

Whenever a sequence of cause-effects is successfully executed it could be memorized into a *rule* to relieve the reasoning load of the planner. Rules are essentially similar to cause-effects but instead of an action they contain sequences of actions, each one in turn expressed as cause-effect.

Figure 1 illustrates a general schema of the system.

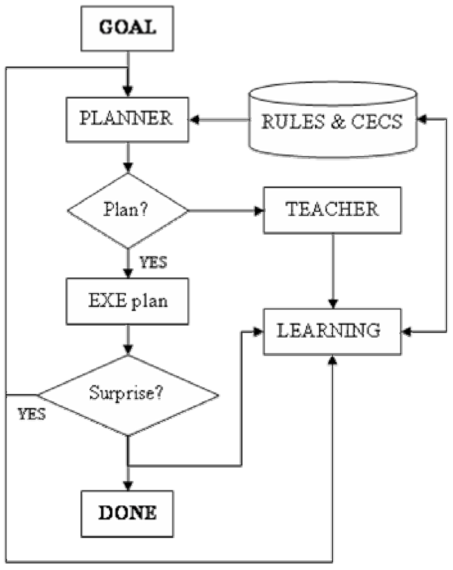

Figure 1. General architecture of the decision making system.

## Notation

We assume that the agent has a set of $N$ detectors $d_i$, $i$=1...$N$, that could take different discrete values $d_{ij}$, $j$=1..|$d_i$|, called conditions. A state $s$ is constituted by a set of conditions $d_{ij}$, $s=\{d_{1j}, d_{2k}, ..., d_{Nl}\}$. Any set of state conditions is denoted as a subspace $ss$.

At every moment the agent is able to perform any of the $k$ actions from the set $A=\{a_1, a_2, \ldots, a_k\}$.

We define a *probabilistic cec* (*pcec*) by a tuple containing a precondition part that involves two set of subspaces, the working ($SS_w$) and the candidate ($SS_c$) subspaces, and a list of example states $L_s$, an action part $a_{pcec}$, and the expected outcome $O_{pcec}$ of the *pcec*, coded also as a subspace.

$$pcec = < \{SS_w, SS_c, L_s\}, a_{pcec}, O_{pcec} >$$

Each subspace $ss$ in $SS_w$ and $SS_c$ has associated three numbers,

- $n_+^{ss}$ , counter for positives examples covered by $ss$.
- $n_-^{ss}$ , counter for negatives examples covered by $ss$.
- $n^{ss}$ , total number of possible states in the region covered by $ss$.

Where positive and negative examples are those states experienced, and stored under which the expected change $O_{pcec}$ occurs after the execution of $a_{pcec}$ or fails to be obtained, respectively. In a similar way, we refer to the probability of a positive example as the probability of obtaining the expected change, and the probability of a negative one for the converse.

Additionally, we formally represent a cause-effect $cec_i$ using a tuple that consists in a subspace $P_i$ called the preconditions, an action $a_i$, and a subspace $O_i$ denoted as the expected outcome of the $cec_i$. The preconditions indicate under which conditions the cause-effect can be applied, and the expected outcome reflects the effects that will be obtained after its execution.

$$cec_i = <P_i=\{d_{gj},...,d_{ml}\}, a_i , O_i=\{d_{kl},...,d_{pq}\}>$$

Thus, a *cec* can be seen as a formal instantiation of the more abstract OAC as discussed in the Introduction.

In the same way, a rule $R_j$ is described using a tuple that consists in a subspace $P_j$ called the preconditions of the rule $R_j$, a sequence of cec's $CECS=(cec_k, cec_i,..., cec_m)$, and a subspace $O_j$ denoted as the expected outcome of the rule,

$$R_j = <P_j=\{d_{ih},...,d_{ml}\}, CECS , O_j=\{d_{kl},...,d_{pq}\}>$$

Rules can therefore be considered as chains of OACs. In our approach, the expected outcome serves two purposes: it will be used by a goal-achieving deliberative system for planning and to evaluate the outcomes.

## Generating cecs from pcecs

It is not possible to use the probabilities estimations directly for planning because the system is not designed for a probabilistic planner but for a deterministic one. Thus, for plan generation only *cecs* and rules can be used. In this section it is explained how *cecs* are obtained from *pcecs*.

To generate *cecs* from *pcecs* we only use subspaces of the set of working subspaces $SS_w$. First, an estimation of the probabilities for a positive and a negative example for each subspace in $SS_w$ need to be obtained. There are many ways of estimating these probabilities. Due to the problem that small numbers for the example can bias statistical estimators much we propose using the following estimator, which is robust against this effect.

$$P_+^{ss} = \frac{1}{2}\left(1 + \frac{n_+^{ss}}{n^{ss}} - \frac{n_-^{ss}}{n^{ss}}\right)$$

$$P_-^{ss} = \frac{1}{2}\left(1 + \frac{n_-^{ss}}{n^{ss}} - \frac{n_+^{ss}}{n^{ss}}\right) = 1 - P_+^{ss}$$

where 2 accounts for the number of classes.

In the general case the probability of a class $i$ is,

$$P_i^{ss} = \frac{1}{K}\left(1 + (K-1)\frac{n_i^{ss}}{n^{ss}} - \sum_{j,\forall j \neq i} \frac{n_j^{ss}}{n^{ss}}\right)$$

where $K$ is the total number of classes and $j$ accounts for all the classes except $i$.

This model consists in a function which outcome ranges in [0, 1], hence with probabilistic interpretation, that not only takes into account the class examples to determine the probability of that class (positive or negative in our case), but also other classes examples. Additionally, it also takes into account the densities of examples for different classes.

Calculating the probability in this way is similar to assuming that, in lack of any evidence, each state in the subspace has the same "proportion" of a positive and a negative example (uniform distribution). This proportion will change as new evidence is gathered as a function of the density of classes. For instance, the probability for a positive example could range from 0, when all the covered states are instantiated with negative examples, to 1 when the whole subspace is occupied with positive instances.

After these definitions we can continue with the *cec* generation procedure: After calculating the probabilities for a working subspace, if the probability of a positive example is nonzero, a *cec* is created and added to the list of *cecs* that will be used for planning. Every new *cec* is composed of,

$$cec_{new}=<P_{new}=ss_w,\ a_{pcec},\ O_{new}=O_{pcec}>$$

where $ss_w$ is the working subspace.

The process of *cec* generation is performed whenever a *pcec* is created and for every recently promoted working subspace that have nonzero probability for a positive example.

### About Exploration

In the decision making system exploration of actions is dictated by the teacher and the planner. Nevertheless, as the planner generates plans using *cecs*, which not yet completely evaluated we use forced exploration that permits learning a more complete model of the world dynamics.

## Learning Module

The learning module of the decision making system learns rules and *pcecs* evaluating under which conditions a change would occur after an action execution. Incomplete knowledge of the conditions necessary to afford changes leads to uncertainties about the occurrence of the change under a situation. Nevertheless, it is possible to estimate the probability for that change to occur given a subspace. In the next section we present how these estimations are improved, and how the minimal set of conditions that affords the expected changes is found.

## Learning *pcecs*

The core of the method for learning *pcecs* is the estimation of probabilities for a positive and a negative example for different subspaces. The estimations and subspaces generation procedures are guided by experienced states stored in $L_s$. The aim is to find the smallest sets of conditions for which the expected change has a high probability to occur.

The learning process consists in selectively storing positive and negatives examples related to each *pcecs*, and refining the $SS_w$ representation by promoting candidate subspaces from $SS_c$ to $SS_w$.

### Initialization

The generation of a *pcec* occurs after executing an instructed action. The initial structure of a *pcec* consists in the expected outcome $O_{pcec}$, which involves the changed conditions after the action execution, the action itself $a_{pcec}$=instructed action, and the initial set of working and candidate subspaces.

The initial set of working subspaces $SS_w$ is composed by one subspace formed with the conditions changed following the action $a_{pcec}$. In the case of $SS_c$, the initial set of subspaces is composed by subspaces formed with the conditions changed with $a_{pcec}$, and one additional condition of a detector not involved in the changes. It is considered one candidate subspace for each condition of those detectors.

### Memorizing Examples

Every experienced state, either positive or negative, is stored in $L_s$ of the *pcec* used to create the *cec* whenever the probability of error $P_e$ (probability of misclassification) is greater than a critic threshold $P_c$.

The probability of the error is calculated as [21],

$$P_e = \min\left[P_-^{ss_w}, P_+^{ss_w}\right]$$

The threshold $P_c$ is determined using a linear function of the density of the subspace,

$$P_c = P_e^{\max}\delta = \frac{\delta}{2}$$

where $\delta$ is the density calculated as,

$$\delta = \frac{n_-^{ss_w} + n_+^{ss_w}}{n^{ss_w}}$$

and $P_e^{max}$ is the maximum possible probability of error.

In this way, for low density subspaces the probability error is less significant and examples are then stored to reduce the uncertainty in the estimations, while, as the density becomes larger, the probability of the error becomes more meaningful.

Note that the storing criterion implies that every *cec* should have a pointer to the *pcec,* from which it originated and to the working subspace of that *pcec,* which was used to set all its conditions.

Finally, after an example is stored, all the counters of the subspaces in $SS_w$ and $SS_c$, the conditions of which are included in the stored example, are updated.

### Promoting Candidate Subspaces

The necessity of improving the estimation by promoting candidate subspaces is also measured using the probability of the error and the density of the subspaces. The promotion of candidate subspaces occurs when enough examples were stored but the estimation capability of the system is still bad. To consider this requirement we propose using a threshold for the probability of the error that is a linear decreasing function of the density. In this manner, when the evaluated working subspace has a few examples the uncertainty is high and more evidence should be accumulated before promoting any candidate subspace. On the other hand, when the working subspace is densely occupied with examples, higher probabilities of the error are more trustable as indicators of the necessity for refining the representation. Then, the threshold for the probability of the error is calculated as,

$$P_c^{prom} = \left(P_e^{\min} - P_e^{\max}\right)\delta + P_e^{\max}$$

where $P_e^{min}$ is the highest error allowed, which plays the role of an upper bound for the precision in the estimation, preventing over-fitting, and allowing a better treatment of noisy data.

After the calculation, if the probability of error of the evaluated working subspace is above the threshold, then, from all the candidate subspaces from $SS_c$ involving the evaluated working subspace, the one with lowest probability of error is promoted. To save computational resources, only working subspaces related to *cecs* that produce surprises are evaluated.

### Generating Candidate Subspaces

Every time a candidate subspace is promoted new candidate subspaces are generated.

The generation takes place for each consistent combination of the recently promoted subspace with the conditions of another working subspace of the *pcec*. All the possible combinations will produce new candidate subspaces.

Finally, once the candidate subspaces are generated, the counters of examples are initialized in accordance to the positive and negative examples covered by them.

## Learning Rules

So far, we have explained how *pcecs* are learned from experience and how *cecs* are generated from *pcecs*. Now, we will explain how to relieve the job of the planner by memorizing sequences of successfully executed *cecs* into a macro planning operator called *rule*.

To guarantee successful execution of a sequence, the precondition of the to-be-executed rule should ensure the occurrence of the *cec*-preconditions in the proper order. This is achieved firstly by accumulating (in the precondition part of the rule) the preconditions of the *cecs* needed to afford the changes. In case a detector takes more than one condition value during the sequence, the condition closer to the start of the sequence is considered as it should occur first.

On the other hand, the outcome of the rule should enumerate the results obtained after the execution of the whole sequence. This is done by accumulating the outcomes of each *cec* into the outcome-part of the rule. As latest outcomes in the sequence cancel early ones when the same detectors are involved, the rule outcome should consider for each detector the condition of those *cec* later in the sequence whenever the same detector is involved.

It is important to remark that, in this first approach, we let the teacher control the rule generation by the instruction given. The teacher will instruct a single action when no sequence is convenient to be merged into a rule, and he/she will instruct a sequence of actions whenever he/she knows that this sequence will be need many times during the task or could be difficult to find for the planner.

### Cecs and Rules Correction

When a surprise arises, the *pcec* from which the *cec* was obtained is evaluated. For this we require that there is a working subspace in the *pcec* that has higher probability for being a positive example than the subspace from which the *cec* originated. Furthermore this subspace has to be consistent with the whole sequence of *cecs* stored in the rule. Only then an update is performed. This is done by replacing the conditions of the *cecs* with the conditions of the working subspace with the highest probability.

Consistency in the sequence requires that the changes produced by previous *cecs* in the sequence as well as all the preconditions of posteriors *cecs* are contemplated in the situations where a *cec* should be applied. Additionally, changed affordance may require conditions that do not change by themselves but are nonetheless necessary for the execution of the *cec*. Those conditions should also be guaranteed. Thus, to update a *cec* in a rule, all the previous restrictions need to be verified.

If the *cec* results are modified, the rule correction is simply performed by updating the rule preconditions and outcomes with the new added conditions. This is done following the procedure of rule generation but applied only to the modified parts.

## Outline of the Algorithm

In this section we present the algorithm of the decision making system using a pseudo-code.

### Pseudo-code

```
INIT system
Define GOAL
WHILE goal is not reached
  {
  IF PLAN found
    {
    Execute PLAN
    }
  ELSE (plan not found)
    {
```

```
    Teacher instructs actions
    FOR each action instructed,
      {
      Execute action
      GENERATE new pcec
      GENERATE new cec from new pcec
      APPEND new cec to LCECS
      }
    GENERATE RULES using LCECS
    } (else if plan found)
}(end while GOAL is not reached)
```

### Execute PLAN

```
FOR each cec in PLAN
  {
  Execute action
  IF surprise
    {
    IF high uncertainty in the estimations
      {
      STORE negative example
      }
    If necessity of promoting candidate subspace
      {
      PROMOTE best candidate subspace
      GENERATE new candidate subspaces
      CORRECT cec with promoted subspace
      CORRECT rules containing cec
      }
    EXIT FOR (stop plan execution and replan)
    }
  ELSE (no surprise)
    {
    IF high uncertainty in the estimations
      {
      STORE positive example
      }
    }
  } (end for)
```

The learning module procedures are detailed in figure 2. To contextualize see that these procedures are those that take place inside the learning box in the schema of figure 1.

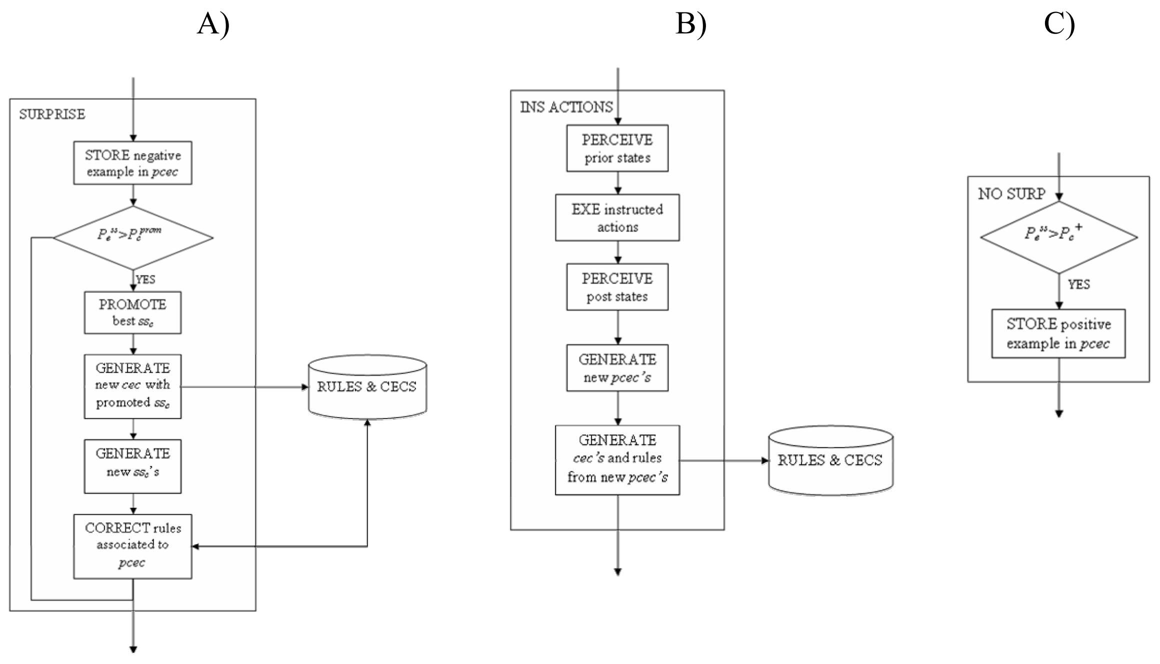

Figure 2. A) Learning when there is a surprise; B) Learning after action instructions; C) Learning after a successful execution of a *cec*.

## Conclusions

The contribution of this work is an extension of the previous system presented in [5]. The general architecture of the previous system is maintained but there is a significant improvement in the process of learning cause-effects. In [5] the cause-effects were learned with the help of human explanations about relevant conditions that afford changes. Now this process is completely automatic permitting not only to learn of cause-effects without the help of the teacher but also to estimate the chances of producing the desired outcome for any set of conditions. Hence, the learning method could be now applied to stochastic and non-stationary environments. On the other hand, this probabilistic approach may constitute another step toward a possible formalization of the OAC: an object description is not restricted to a unique set of conditions but multiple set of conditions with different chances of affording the object functionality.

Nevertheless, there are still many other pending issues to treat like the evaluation of the method in real scenarios and the automatic explorations of actions whenever the planner fails to find a plan. Other topics are the definition of a criterion for plan memorization into rules and the integration of the learning method with an advanced planner module.

## References

[1] Newell, A. The knowledge level. Artificial Intelligence, 18(1), 87-127, 1982.

[2] Brooks, R. “Intelligence without representation”. Artificial Intelligence, 47, pp. 139-159, 1991.

[3] Mitchell, T. Machine Learning. McGraw Hill. 1997

[4] Piaget, J. The origins of intelligence in children. New York: International Universities Press. 1952.

[5] Agostini A., Celaya E., Torras C., Wörgötter F. Action Rule Induction from Cause-Effect Pairs Learned Through Robot-Teacher Interaction. In Proc. of the International Conference on Cognitive Systems, CogSys 2008. (Karlsruhe, Germany). April 2008, pp. 213-218.

[6] Wörgötter F., Agostini A., Krüger N., Shylo N., Porr B. Cognitive Agents - A Procedural Perspective relying on the Predictability of Object-Action-Complexes (OACs), Robotics and Autonomous Systems, 2008 (In press. doi:10.1016/j.robot.2008.06.011).

[7] Geib, C., Mourao, K., Petrick, R., Pugeault, N., Steedman, M., Krüger, N. and Wörgötter, F.. Object Action Complexes as an Interface for Planning and Robot Control, presented at IEEE RAS Int Conf. Humanoid Robot, Genova, Italy, 2006.

[8] http://www.paco-plus.org.

[9] Lemaître, M., Verfaillie, G.. Interaction between reactive and deliberative tasks for on-line decision-making. Presented at the 2007 International Conference on Automated Planning and Scheduling, Providence, Rhode Island, USA, 2007.

[10] Gat, E. On three-layer architectures. In D. Kortenkamp, R. P. Bonnasso, and R. Murphy, editors. Artificial Intelligence and Mobile Robots. MIT/AAAI Press, pp. 195-210, 1998.

[11] Schoppers, M. J.. Universal Plans for Reactive Robots in Unpredictable Environments. In Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI 87), Milan, Italy, 1987, pp. 1039-1046.

[12] Newton, M., Levine, J.. Evolving Macro-Actions for Planning. Presented at the 2007 International Conference on Automated Planning and Scheduling, Providence, Rhode Island, USA, 2007.

[13] Nicolescu, N., Mataric, M.. A Hierarchical Architecture for Behavior-Based Robots. In Proc. of the 1st Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems, Bolgna, Italy, 2002, pp. 227-233.

[14] Rao, D., Jiang, Z., Jiang, Y.. Learning Activation Rules for Derived Predicates from Plan Examples. Presented at the 2007 International Conference on Automated Planning and Scheduling, Providence, Rhode Island, USA, 2007.

[15] Yoon, S., Kambhampati, S. Towards Model-lite Planning: A Proposal For Learning & Planning with Incomplete Domain Models. Presented at the 2007 International Conference on Automated Planning and Scheduling, Providence, Rhode Island, USA, 2007.

[16] Wang, X. Learning planning operators by observation and practice. In Proceedings of the Second International Conference on AI Planning Systems, Chicago, IL, USA, 1994.

[17] Oates, T. and Cohen, P. Learning planning operators with conditional and probabilistic effects. In Proceedings of the AAAI Spring Symposium on Planning with Incomplete Information for Robot Problems, 1996, pp. 86-94.

[18] Benson, S. Inductive learning of reactive action models. In Proceedings of the 12th International Conference of Machine Learning, 1995, pp. 47–54.

[19] Sutton, R. and Barto, A. Reinforcement Learning. An Introduction. MIT Press, 1998.

[20] La Valle, S. Planning Algorithms. Cambridge University Press. 2006.

[21] Duda, R., Hart. P and Stork, D. Pattern Recognition. John Wiley & Sons, Inc., 2nd Edition, 2001.