Implementing Adaptive Capabilities on Agents that Act in a Dynamic Environment

Giuliano Armano
DIEE
Department of Electrical and Electronic Engineering
University of Cagliari
armano@diee.unica.it

Giancarlo Cherchi
DIEE
Department of Electrical and Electronic Engineering
University of Cagliari
cherchi@diee.unica.it

Eloisa Vargiu
CRS4
Center for Advanced Studies, Research and Development in Sardinia
eloisa@crs4.it

Abstract

Acting in a dynamic environment is a complex task that requires several issues to be investigated, with the aim of controlling the associated search complexity. In this paper, a life-cycle for implementing adaptive capabilities on intelligent agents is proposed, which integrates planning and learning within a hierarchical framework. The integration between planning and learning is promoted by an agent architecture explicitly designed for supporting abstraction. Planning is performed by adopting a hierarchical interleaved planning and execution approach. Learning is performed by exploiting a chunking technique on successful plans. A suitable feedforward neural network selects relevant chunks used to identify new abstract operators. Due to the dependency between abstract operators and already-solved planning problems, each agent is able to develop its own abstract layer, thus promoting an individual adaptation to the given environment.

1. Introduction

Intelligent agents are systems capable of autonomous sensing, reasoning and acting in a complex environment. The introduction of such systems has promoted investigation on new AI algorithms and techniques, with the aim of using them in real-world domains, as the classical assumptions no longer hold. In particular, real-world domains are usually not-completely controllable, not-completely accessible, and dynamic (see, for example, [13] for a discussion on this topic). Furthermore, some additional real-time constraints may hold. The ideal agent, devised for dealing with all these issues, should exhibit: (i) a suitable *planning* complexity, (ii) the capability of quickly *reacting* to environmental changes, and (iii) the ability of *adapting* itself to the given environment, as well as to the given planning problems.

1.1. Planning in complex environments

A complex environment is fairly intractable using traditional planning methods, since the search space can be quite large even for relatively simple tasks. As planners are used in increasingly complex domains, the ability to decompose problems and to identify their most abstract aspects becomes a critical issue. An effective approach used for dealing with the planning complexity is to build a set of abstractions for controlling the search. Abstraction is a technique that requires the original search space to be mapped into new abstract spaces, in which irrelevant details have been disregarded at different level of granularity.

Basically, two distinct abstraction mechanisms have been studied in the literature: *action*- and *state*-based. The former combines a group of actions to form macro-operators [10], whereas the latter exploits representations of the world given at a lower level of detail. The most significant forms of state-based abstraction rely on: (i) *relaxed* models, obtained by dropping operators' applicability conditions [15], and on (ii) *reduced* models, obtained by completely removing certain conditions from the problem space [8]. Both models, while preserving the provability of plans that hold at the ground level, perform a "weakening" of the original problem space, thus suffering from the problem of introducing "false" solutions at the abstract level [5].

Given a problem space and a set of abstraction spaces organized in a hierarchy, a hierarchical planner first solves a problem in the most abstract space and then uses the abstract solution to guide the search in the underlying space. The search proceeds recursively until a solution is found while searching the ground space.

1.2. Quickly reacting in a dynamic and not completely accessible environment

The capability, for an agent, of quickly reacting to any change is a major issue in dynamic environments; in particular, events that make the current activity needless, or impossible to be completed, must be identified as soon as possible. To this end, planning and execution monitoring can be interleaved, thus giving rise to a powerful control strategy (e.g., IPEM [1] and Rogue [6]).

Moreover, the planning process becomes harder when an agent does not have complete information about its environment, for multiple conditions have to be considered. To overcome this problem, an early execution of the actions could be used to obtain more information, useful for subsequent activities [14]. Another desired property of planners that act in a dynamic environment is the capability of avoiding global replanning when some exogenous conditions collide with the current plan, this property being particularly relevant for domains where additional quasi real-time constraints hold (e.g., computer games [13]).

1.3. Adaptation as integration between planning and learning

Self adaptation is a very powerful ability of agents that act in complex environments. Generally speaking, all adaptive techniques dedicated to speed up planning fall into the category of learning control knowledge. Several approaches aimed at learning control knowledge have been proposed; in particular, let us recall: a) learning abstractions from the domain knowledge (e.g., ABSTRIPS [15]), b) learning macro-operators [9], c) explanation-based learning techniques [7], d) learning abstractions from the problem to be solved (e.g., ALPINE [8]).

Perhaps, the two most important examples of full integration between learning and planning are the SOAR [11] and PRODIGY [4] systems, where the capability of embedding the learning activity within an adaptive framework that encompasses planning, learning, and execution is experimented.

1.4. Outline of the Paper

In this paper, an adaptive mechanism that allows an agent to discover abstract operators from successfully solved plans is described. In the proposed approach, planning is performed exploiting two levels of abstraction (i.e. *ground* and *abstrac*t). An abstract solution can be seen as a sequence of abstract operators, each embodying a subproblem to be solved by the underlying ground-level planner. Any new abstract operator becomes a candidate for being embedded into the abstract planner for other problems to be solved. Due to the dependency between abstract operators and already-solved problems, each agent is able to develop its own set of abstract operators, thus promoting an individual adaptation to the given environment.

The remainder of this paper is organized as follows: in section 2, after briefly depicting the system architecture and the HIPE (hierarchical interleaving planning and execution) approach, the proposed learning life-cycle and mechanism are described; in section 3 experimental results are discussed, and in section 4 conclusions are drawn and directions for future work are sketched.

2. A hierarchical adaptive approach for acting in a dynamic environment

To deal with the constraints imposed by the environment, a hierarchical approach has been adopted for designing and implementing agents, which encompasses three main interacting aspects: (i) their underlying architecture and domain knowledge, (ii) their proactive and reactive capabilities, and (iii) their adaptive behavior.

2.1. A layered architecture for hierarchical planning

Bearing in mind that our agents are aimed at implementing a goal-oriented behavior in a dynamic environment, we defined a two-pass vertically layered architecture that can be equipped with N layers. Each layer exploits a local knowledge base (KB), and is numbered according to its level of abstraction. In the proposed architecture all layers are –at least conceptually– identical, each being able to embody in the same way reactive, deliberative and proactive functionality. Only the responsibilities of a layer change, depending on the level of abstraction being considered. According to the basic features that characterize a two-pass vertically layered architecture, the information flows from level 0 up to level N-1, whereas the control flows from level N-1 down to level 0.

As for the domain knowledge, it has been organized at different levels of granularity, through three separate dimensions: types, predicates, and operators. In particular, predicates and types have been abstracted using the *is-a* and *part-of* meta-relations.

The ideal adaptive behavior, quite complex indeed, could be automatically obtained by a suitable learning algorithm able to handle types, predicates, and operators together. Here, we propose an automated mechanism devoted to generate only abstract operators, given an initial (hand-coded) hierarchical description of the domain.

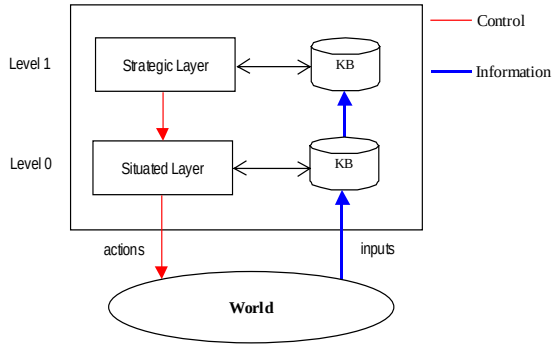

Figure 1. A layered agent architecture.

2.2. Hierarchical interleaving planning and execution

To efficiently handle a dynamic environment, a hierarchical interleaving planning and execution approach (HIPE) has been adopted, and the underlying architecture, as well as the overall planner, has been customized accordingly. In particular, we implemented agents equipped with two layers (i.e., *situated* and *strategic* as depicted in figure 1), each layer of the architecture being devoted to host a corresponding level of abstraction (i.e., *ground* and *abstract*, respectively). Each layer has been equipped with a corresponding planner, which follows an UCPOP-like approach [17], although –in principle– this choice does not affect the overall architecture, provided that the selected planning algorithm is sound and complete.

An agent repeatedly performs planning, executes actions, and monitors any change occurred in the environment, both at the situated and strategic layer. Note that the ground-level planner is devoted to perform planning on any goal imposed by the abstract-level planner, so that "executing" an action at the abstract level actually means creating a subgoal to be solved by the ground-level planner. Plan execution starts as soon as the first abstract operator has been refined. On completion, the next abstract operator (if any) is refined according to a classical depth-first approach (see [2] for further details).

Two main problems arise, concerning the soundness and completeness properties of an abstraction. In fact, for a given abstraction hierarchy, both the downward and upward solution properties may hold [16]. The Downward Solution Property (DSP) ensures that, for every abstract solution, at least one corresponding ground solution exists, thus yielding to an abstract-level planner able to find only "true" solutions. Conversely, the Upward Solution Property (USP) ensures that, for any ground solution, at least one corresponding abstract solution exists. Moreover, the Downward Refinement Property (DRP) makes the assumption that an abstract solution can always be refined without any backtracking. A weaker form of DRP (called near-DRP [3]) accepts the backtracking, provided that it occurs seldom.

Here, we are interested in dealing with a "quasi-sound" abstraction, i.e., with an abstraction in which the near-DRP holds. This property, although difficult to enforce automatically, can be easily implemented by a knowledge engineer while assessing the existence of default solutions for every abstract operator. It is worth pointing out that, in presence of near-DRP, backtracking may contrast the interleaving process, and greatly reduce the overall proactive performance of an agent.

On the other hand, no a-priori assumptions are made about completeness, except for the fact that it could be asymptotically obtained by repeatedly adding abstract operators to the planner located at the strategic layer.

2.3. Integrating learning with a HIPE approach

To be effective, the proposed HIPE approach requires an adaptive mechanism to be enforced, aimed at identifying new abstract operators (useful for any subsequent search at the abstract level). The three main sources of information for the learning mechanism are: world domain, planning problems, and solutions. We analyze successful plans, particularly those for which the abstract-level planner failed, in search of relevant sequences that could play the role of "supporting macro-operators" while devising new operators at the abstract level. When feasible, supporting macro-operators can also be used by the ground level planner during operator refinement; otherwise, local planning must be performed, thus laying this approach between case-based planning and "systematic" local search.

It is worth pointing out that the set of "solvable" problems at the abstract level grows according to the capability of embodying new solutions into the domain knowledge, in form of abstract operators. Thus, in principle, the abstract space should asymptotically become "complete", although the well known utility problem [12] must be taken into account. In fact, a trade-off between the desire to extend the "degree of completeness" of the planner and the number of operators made available to the abstract level must be adopted, since embodying a new operator negatively affects the branching factor.

2.4. The adaptive behavior life cycle

To identify abstract-operators, a "chunking" technique is exploited, which extracts sequences from successful plans. Relevant sequences are identified by a feedforward

neural network (FFNN), fed by a vector of suitable metrics evaluated for each given sequence (see table 1). As shown in figure 2, the adaptive life-cycle consists of repeatedly performing the following actions:

1. The HIPE planner is activated on the given goal (let us assume that a suitable solution is found and executed).
2. Plan subsequences are randomly extracted from the plan (chunking).
3. Relevant sequences are filtered out by a suitable FFNN. For every sequence, a corresponding vector of metrics is evaluated, to be used as input to the FFNN.
4. Each sequence considered relevant passes through a schematization process, which creates a corresponding macro-operator schema (MO). All MOs are temporarily stored into the MOS REPOSITORY.
5. Each MO passes through an abstraction process, which creates a corresponding abstract-operator schema (AO) out of any given MO. For every MO, the AO REPOSITORY is updated (if needed, the newly-created AO is added to the repository together with its supporting MO; otherwise the MO is associated with its corresponding AO already stored into the repository).
6. When an AO is stored into the AO REPOSITORY, a suitable procedure must be invoked that decides whether or not the AO has to be added to the set of active AOs (the ones used by the abstract-level planner), according to a

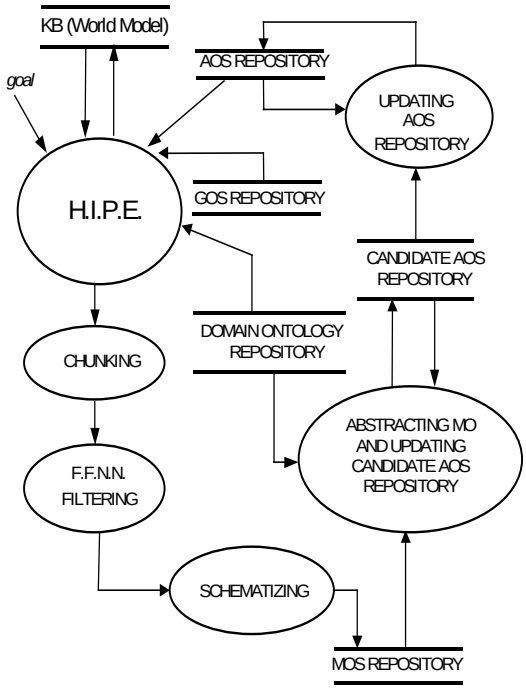

Figure 2. The adaptive behavior life-cycle.

Table 1. Metrics used for assessing sequences of ground operators.

Name	Formula	Brief Description
Chunk Length	$LC(s)=\begin{cases}\frac{(\lvert s\rvert-2)^2}{16}-\frac{\lvert s\rvert-2}{2}+1, & 2\le\lvert s\rvert\le 6\\ 0\ otherwise\end{cases}$	Penalizes long sequences by associating a different value to each length of the sequence.
Preconds-Length	$LPRE(s)=\frac{1}{\sqrt{\lvert pre(s)\rvert}} \quad \lvert pre(s)\rvert>0$	Penalizes sequences with a high number of corresponding preconditions.
Postconds-Length	$LPOST(s)=\frac{\lvert s\rvert}{L_0+\lvert post(s)\rvert}$	Penalizes sequences with a high number of postconditions. L_0 is the "preferred" (averaged) sequence length.
Establishment	$E(s)=\sum_{i=1}^{\lvert s\rvert-1}\frac{\lvert eff(op_i(s))\cap pre(sseq_i(s))\rvert}{\lvert pre(sseq_i(s))\rvert}$	Promotes sequences in which each operator establishes literals useful to the rest of the sequence. op_i extracts the i-th action of the sequence s. $sseq_i$ extracts the rest of the sequence s, starting from the i+1-th operator.
Redundancy	$R(s)=\frac{\lvert Ops(s)\rvert}{\lvert s\rvert}$	Penalizes sequences in which the same operator is replicated. ops(s) extracts the set of operators from s.
Cost	$C(s)=\frac{1}{\sqrt{\sum_{i=1}^{L}opc(op_i(s))}}$	Promotes low-cost sequences. The cost of a sequence is estimated according to the number of its pre- and post-conditions through the function opc(s). A different weight is given to pre- and post-conditions, as the "cost" of a sequence is mainly due to preconditions.
Weighted-Postconds	$WPOST(s)=\frac{\sum_{i=1}^{\lvert eff(s)\rvert}\lvert occur(eff_i(s))\rvert}{numOps}$	Promotes sequences according to their capability of instantiating effects that occur frequently as a precondition of the other operators. eff_i returns the i-th effect introduced by the sequence s. occur returns the set of operators that require a given effect. numOps is the total number of operators.

policy that takes into account the trade-off between the increase of complexity (due to the branching-factor) and the augmented capability of solving problems at the abstract level.

neurons, together with a single hidden layer of 3 neurons. Its output is a float value in [0,1]; the higher the output, the more significant the given sequence. The FFNN has been trained with 54 sequences (50% positive and 50% negative examples) extracted from a successful plan

Table 2. An example of sequence identified as relevant, together with its corresponding macro- and abstract-operator schemata.

Ground Level		
Supporting MO - #1	Actions	((OPEN-DOOR door house) (GO-INSIDE door house) (MOVE-TO-OBJ key house) (TAKE-OBJ key house) (MOVE-TO-OBJ friend house) (GIVE key friend))
	Preconds	((:NOT (opened door house)) (located friend house) (next-to self house) (:NOT (blocked road)) (located key house))
	Postconds	((owns friend key) (next-to-obj friend) (:NOT (located key house)))
	Metrics	(0.04 0.28 0.66 0.83 0.83 0.20 0.68)
	FFNN output	0.84
Supporting MO - #2	Actions	((MOVE-TO-OBJ key house) (TAKE-OBJ key house) (MOVE-TO-OBJ friend house) (GIVE key friend))
	Preconds	((located friend house) (inside self house) (:NOT (blocked road)) (located key house))
	Postconds	((owns friend key) (next-to-obj friend) (:NOT (located key house)))
	Metrics	(0.09 0.35 0.57 0.61 0.75 0.26 0.62)
	FFNN output	0.72
Abstract Level		
	Preconds	((located agent building) (is-near self building) (located object building) (movable object))
	Postconds	(owns agent object) (:NOT (located object building)))
Ontology		
	IS-A	(is-a house building) (is-a friend agent) (is-a key object) (is-a next-to is-near) (is-a inside is-near)
	Invariants	(movable key)

3. Experimental results

The described system has been implemented as a prototype of a computer game, mostly written in C++, although the AI kernel (planning, reactive and adaptive modules) is currently written in CLOS (Common Lisp Object System). The game lets the user play in a simulated environment inspired to the real-world, and represents a virtual city populated by two different kind of entities (physical and network ones), both implemented by intelligent agents. Players have an agent representative within the game (i.e., an avatar), which can be given a goal to be attained through a simple interface. Despite the presence of multiple agents within the game, interactions between agents are limited to simple actions (e.g., exchanging objects or information). The proposed architecture has been tested on case studies taken from the virtual world that characterizes the game.

Steps 1 to 5 of the adaptive life-cycle previously described have already been implemented. As for step 6, we are currently searching for an optimal balance between the augmented capability of solving problems at the abstract level, and the growth of the search complexity due to the higher average branching factor. In fact, this is a major point in the problem of deciding whether or not to let a further operator become available to the abstract planner. The FFNN that has been adopted to identify relevant sequences is composed by an input layer of 7

generated by the HIPE planner. The hierarchical planner solved the problem in 16 steps, from which the subsequences used as training set were extracted.

Some refinements of existing abstract operators manually crafted by a knowledge engineer, have been considered as positive examples during the training phase. A threshold μ_A has been used to separate relevant sequences from non-relevant ones. Being M the metrics-evaluator vector and f_{NN} the neural classifier, a sequence s has been considered relevant for abstraction iff $f_{NN}(M(s)) \geq \mu_A$. With $\mu_A = 0.65$, about 97% of the training sequences have been recognized as belonging to their proper class (i.e., relevant or not-relevant). Note that we relinquished to the idea of reaching a recognition rate of 100% on the training set (not necessary in this case), thus avoiding problems related to overfitting.

As test sets we have used some examples taken from tipical situations that occur in the virtual world. During the experiments carried out, the FFNN identified several sequences useful for abstraction and not yet obtained as a refinement of an existing abstract operator. Starting from these sequences, new abstract operators candidates have been automatically generated by the system. It is worth pointing out that all sequences filtered out by the FFNN were in fact relevant. An example is given in table 2 where two macro-operators, which support the abstract action "*take an object and give it to an agent*", are shown. For each supporting macro-operator (i) the sequence of

actions; (ii) pre- and post-conditions; (iii) the vector of metrics; (iv) and the FFNN output are reported.

4. Conclusions and future work

In this paper, a first step towards agents that can develop an adaptive behavior has been performed. The adaptive life-cycle, integrated within a hierarchical interleaving planning and execution approach, is driven by the given environment and by the history of problems that have been tackled by each agent.

As for the future work, we are willing to use domain ontologies for integrating planning and learning within a unifying framework. Furthermore, we are investigating how to close the learning life-cycle, i.e. how to assess when a candidate abstract operator can be embodied in the set of active abstract operators.

References

[1] J. A. Ambros-Ingerson, and S. Steel, "Integrating Planning, Execution and Monitoring", *Proc. of the 7th National Conference on Artificial Intelligence*. 1988, pp. 83-88.

[2] G. Armano, and E. Vargiu, "An Adaptive Approach for Planning in Dynamic Environments", *2001 International Conference on Artificial Intelligence, IC-AI 2001, Special Session on Learning and Adapting in AI Planning*, Las Vegas, Nevada, June 25-28, 2001, pp. 987-993.

[3] F. Bacchus, and Q. Yang, "Downward Refinement and the Efficiency of Hierarchical Problem Solving", *Artificial Intelligence*, Vol. 71(1), 1994, pp. 41-100.

[4] J. G. Carbonell, C. A. Knoblock and S. Minton, PRODIGY: An integrated architecture for planning and learning, In D. Paul Benjamin (ed.) *Change of Representation and Inductive Bias*. Kluwer Academic Publisher, 1990, pp. 125-146.

[5] F. Giunchiglia and T. Walsh, *A theory of Abstraction*. Technical Report 9001-14, IRST, Trento (Italy), 1990.

[6] K. Z. Haigh and M. Veloso, "Interleaving Planning and Robot Execution for Asynchronous User Requests", *Autonomous Robots*, Vol. 5(1), March 1998, pp. 79-95.

[7] S. Katukam, and S. Kambhampati, "Learning Explanation-Based Search Control Rule for Partial Order Planning", In *Proc. Of the 12th National Conference on Artificial Intelligence*, AAAI Press, vol. 1, July 31-August 4, 1994, pp. 582-587.

[8] C. A. Knoblock, "Automatically Generating Abstractions for Planning", *Artificial Intelligence*, Vol. 68(2), 1994.

[9] R.E. Korf. "Macro-operators: A weak method for learning", *Artificial Intelligence*, Vol. 26(1), 1985, pp. 35-77.

[10] R.E. Korf. "Planning as Search: A Quantitative Approach", *Artificial Intelligence*, 33(1), 1987, pp. 65-88.

[11] J.E. Laird, A. Newell, and P.S. Rosenbloom, "SOAR: Architecture for general intelligence", *Artificial Intelligence*, Vol. 33, 1987, pp. 1-64.

[12] S. Minton, "Quantitative Results Concerning the Utility of Explanation-Based Learning", *Artificial Intelligence*, Vol. 42(2-3), 1990, pp. 363-391.

[13] A. Nareyek, "A Planning Model for Agents in Dynamic and Uncertain Real-Time Environments", In *Proc. of the 1998 AIPS Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, AAAI Press, 1998, pp. 7-14.

[14] I. Nourbakhsh, *Interleaving Planning and Execution for Autonomous Robots*, Kluwer Academic Publishers, 1997.

[15] E.D. Sacerdoti, "Planning in a hierarchy of abstraction spaces", *Artificial Intelligence*, Vol. 5, 1974, pp. 115-135.

[16] J.D. Tenenberg, *Abstraction in Planning*, Ph.D. thesis, Computer Science Department, University of Rochester, 1988.

[17] D. Weld, "An Introduction to Least Commitment Planning", *AI Magazine*, 1994, pp. 27-61.