An Adaptive Approach for Planning in Dynamic Environments

Giuliano Armano
DIEE
Department of Electrical and Electronic Engineering
University of Cagliari
Piazza d'Armi, I-09123, Cagliari, Italy

Eloisa Vargiu
CRS4
Center for Advanced Studies, Research and Development in Sardinia
VI Strada OVEST Z.I.
Macchiareddu, I-09010 Uta (CA), Italy

Abstract

Planning in a dynamic environment is a complex task that requires several issues to be investigated in order to manage the associated search complexity. In this paper, an adaptive behavior that integrates planning with learning is presented. The former is performed adopting a hierarchical approach, interleaved with execution. The latter, devised to identify new abstract operators, adopts a chunking technique on successful plans. Integration between planning and learning is also promoted by an agent architecture explicitly designed for supporting abstraction.

Keywords

Learning Macro Operators, Learning Abstract Operators, Abstraction, Hierarchical Planning, Interleaving Planning and Execution.

1 Introduction

It is well known that the AI community has greatly promoted innovations in the field of autonomous agents, and that planning, learning, and reasoning are among the most important features to be implemented in an intelligent agent. Conversely, the introduction of agents has prompted investigations on AI algorithms and techniques, with the aim of using them in real-world domains, as the classical assumptions no longer hold for them. In particular, real-world domains are usually not-completely controllable, not-completely accessible, and dynamic (see, for example, [20] and [15] for a discussion on this topic). Furthermore, some additional real-time constraints may hold.

An "idealized" agent devised for dealing with these issues should exhibit: (i) suitable planning complexity, (ii) the capability of reacting quickly to environmental changes, and (iii) the ability of adapting itself to the given environment, as well as to the given planning problems.

In principle, for a given abstraction hierarchy, both the Downward and Upward Solution Properties may hold (DSP and USP [19]), as the counterpart –within the planning community– of the soundness and completeness properties, respectively.

The DSP ensures that, for every abstract solution, at least one corresponding ground solution exists, whereas the converse holds for the USP. Usually, abstraction methods allow one to assume that only the USP holds, thus introducing "false" solutions [1] at the abstract levels [8] (see also [5] and [17] for a description given from the problem-solving perspective). Furthermore, when the ratio between "false" and "true" abstract solutions becomes too high, exploiting the hierarchy yields a search that could be less effective than the one performed at the ground level only (see, for example, [2] for a thorough discussion on this topic).

1.1 Dealing with a Dynamic Environment

The capability 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

[1] Namely, abstract solutions that do not actually lead to ground solutions.

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]). 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 [15]). A further powerful framework has been investigated by Nourbakhsh [16], which integrates planning and execution with multiple abstractions, thus yielding a hierarchical approach able to enforce context-dependent control strategies.

1.2 Adaptation as Integration between Planning and Learning

Generally speaking, all adaptive techniques devoted to speed up planning fall into the category of learning control knowledge. For the sake of brevity, we will only recall some of the approaches that have been devised and experimented: a) learning abstractions from the domain knowledge (e.g., ABSTRIPS [18]), b) acquisition of heuristics for state-space [13] or plan-space search [4], c) learning macro-operators [10], d) use of classical learning techniques (e.g., explanation-based [7] and inductive [12] learning), e) learning abstractions from the problem to be solved (e.g., ALPINE [9]). The SOAR [11] and PRODIGY [3] systems, where the capability of embedding the learning activity within an adaptive framework that encompasses planning, learning, and execution is experimented, also deserve mention.

1.3 Outline of the Paper

In this paper an adaptive mechanism is proposed, which allows an agent to discover abstract operators from successful plans. Any such new operator becomes a candidate for being embedded into the agent's hierarchical planner and made available to the abstract level for any subsequent planning problem to be solved. 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.

The remainder of this paper is organized as follows: in section 2 , after briefly depicting the underlying 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 and Adaptive Approach to Interleaving Planning and Execution

It is worth pointing out in advance that our agents must act in a dynamic environment, which models a real-world domain where additional "quasi real-time" constraints hold. To deal with these constraints, a hierarchical approach has been devised that encompasses (i) their underlying architecture, (ii) their proactive and reactive capabilities, and (iii) their adaptive behavior.

2.1 A Layered Architecture for Hierarchical Planning

Bearing in mind that 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 one 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 characterizing 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 we are concerned with investigating the relations among abstraction, planning, and learning in a dynamic environment, for the sake of simplicity, agents have been equipped

with two layers, i.e., situated and strategic (see figure 1).

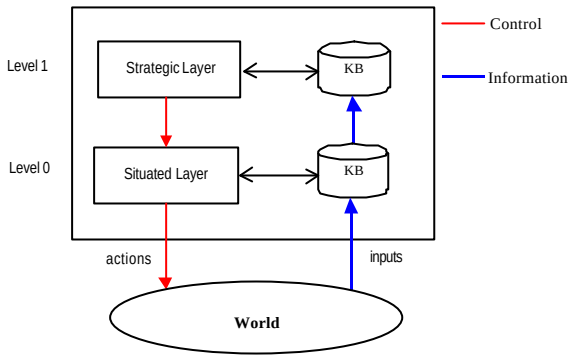

Figure 1 - A layered agent architecture.

2.2 Hierarchical Interleaving Planning and Execution

To efficiently deal with a dynamic environment, a hierarchical interleaving planning and execution approach (HIPE) has been adopted and customized according to the underlying architecture. In fact, the situated and the strategic layer host the ground- and abstract-level planner, respectively. Both planners act according to an UCPOP-like mechanism, although –in principle– this choice does not affect the overall architecture and vice versa.

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 planner is devoted to perform planning on any goal imposed by the abstract one, so that "executing" an action at the abstract level actually means creating a subgoal to be solved by the ground-level planner.

The overall planning strategy enforces the following behavior. First an abstract plan is created at the strategic layer, then the first (abstract) operator is refined at the situated layer, thus ending up with a corresponding plan. At this point, the actions of the ground plan can be executed, while the abstract planner concurrently refines the next abstract operator. When the current plan at the situated layer has been completed, the next abstract operator is executed, and so on. The general form of a layered plan is shown in figure 2.

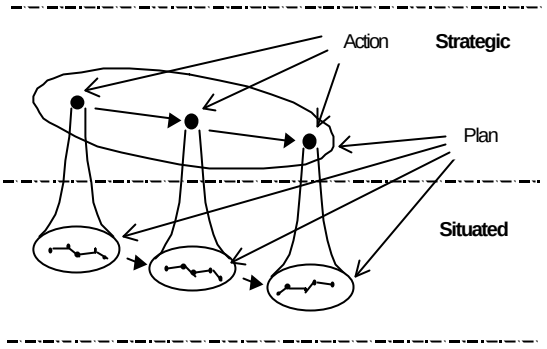

Figure 2 - Structure of a hierarchical plan.

It should be stressed that a plan does not need to be completely refined before starting its execution, the underlying strategy allowing an agent to elaborate a plan as a sequence of abstract operators, whose refinement can be deferred until actually needed. Note that this strategy is feasible under the assumption that no "false" solutions hold at the abstract level, or –at least– that no irreversible actions can be performed that prevent the agent from attaining the current goal.

Being more concerned on avoiding the introduction of "false" solutions at the abstract level, we assume that the resulting abstraction mapping is "sound", according to the definition given in [16], whereas no assumption at all is made on its "completeness". Of course, the absence of the latter property implies that not all ground solutions can be found starting from a high-level search. Therefore –when a failure occurs at the abstract level– a global search must be performed at the ground level, until a solution is found or the ground-level search ends with failure.

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). To this end, 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.

In this way, 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 mapping could asymptotically become both "sound" and "complete", although the well known utility problem [14] must be taken into account. In fact, a trade-off between the desire to extend the "degree of completeness" [2] 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 Learning Abstract Operators

Historically, abstract operators can be obtained by applying two different policies. One is to form a "relaxed" model by dropping their applicability conditions (e.g., ABSTRIPS [18]), the other is to form a "reduced" model by completely removing certain conditions from the problem space (e.g., ALPINE [8]). Both policies perform a "weakening" of the ground-level problem space, while preserving the provability of plans that hold at the ground level but suffering from the problem of introducing "false" solutions at the abstract level.

In this paper we adopt a different approach, being interested in dealing with abstract operators that are "sound", in the sense that for each valid substitution of abstract-level preconditions and postconditions performed according to the given domain ontology, at least a ground plan exists able to refine the abstract operator.

As formalizing the characteristics of a domain that allows to guarantee the "soundness" property is beyond the scope of this paper, we will simply mention that the underlying ontology is characterized by *is-a* and *part-of* relations, together with invariants.

2.5 The Adaptive Behavior Life Cycle

Let us assume that an initial domain ontology at both levels of granularity is supplied off-line, although nothing prevents us from learning abstract operators from scratch. The domain ontology represents, at different levels of detail (i) types of objects, (ii) predicates, and (iii) operators. In principle, all the above issues could be handled by a suitable learning algorithm, thus yielding a powerful, but rather complex, adaptive behavior.

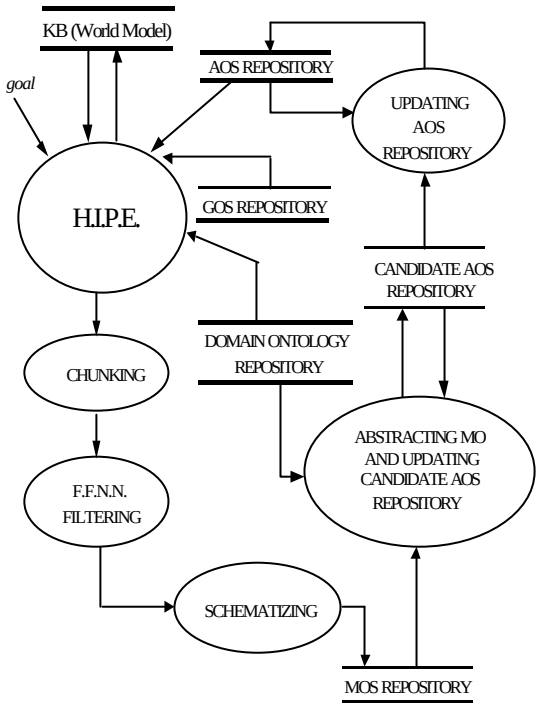

Figure 3 - The adaptive behavior life-cycle.

Less ambitiously, in this paper we are interested in learning abstract operators, given: (i) an immutable hierarchy of objects and predicates that hold at the abstract and ground level, and (ii) a set of ground operators with deterministic effects. The learning activity is driven by a "post-mortem" analysis of successful plans, aimed at identifying further abstract operators, to be subsequently embedded into the planner. To identify abstract-operators, a "chunking" technique is exploited, that extracts sequences from successful plans. Relevant sequences are identified by a feedforward neural network (FFNN), fed by a vector of suitable metrics

[2] For the sake of simplicity, let us denote as "degree of completeness" of an abstraction mapping the ratio between the number of problems that can be solved starting from the abstract level and the total number of problems.

evaluated for each given sequence. So far, the following metrics have been defined:

- *sequence-length-metrics*,
  penalizes lengthy sequences, according to a smoothed shape modeled by a global parameter that embodies the "preferred" length for macro-operators;
- *preconds-length-metrics*,
  penalizes sequences with a high number of preconditions;
- *postconds-length-metrics*,
  penalizes sequences with a high number of postconditions, according to the ratio between the number of postconditions and the length of the sequence;
- *establishment-metrics*,
  promotes sequences where each operator establishes predicates useful to the rest of the sequence (see [9] for the definition of establishment);
- *redundancy-metrics*,
  penalizes sequences where the same operator occurs repeatedly;
- *cost-metrics*,
  promotes low-cost sequences, the cost of a sequence being estimated according to the number of its preconditions and postconditions;
- *weighted-postconds-metrics*,
  promotes sequences according to their capability of instantiating effects that occur frequently as preconditions.

As shown in figure 3, 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 correspondent 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 correspondent 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 stored into the AO REPOSITORY becomes "sound", 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 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.

3 Experimental results

The system has been tested on a prototype (written in CLOS, Common Lisp Object System) of a computer game, where each agent is an *avatar* that represents a player in a simulated environment inspired to the real-world. Through a simple interface, avatars are given goals to be attained. Quasi real-time constraints hold, although –fortunately– an avatar that fails, or that cannot attain a goal in few seconds, usually does not compromise the outcome of the game for its player.

So far, we implemented steps 1-5 of the adaptive life-cycle previously described. As for step 6, we are currently tackling the problem of estimating the "degree of completeness" of the abstract planner, given a set of abstract operators. In fact, this is a major point in the problem of deciding whether making a further operator available to the abstract planner will be useful or not.

The FFNN adopted to identify relevant sequences has a single hidden layer (of 8 neurons), and outputs a real number in [0,1]. It has been trained with 180 sequences (about 60% positive and 40% negative examples) extracted from successful plans generated by the HIPE planner. Positive examples were refinements of abstract operators defined in an initial hierarchy manually crafted by a

knowledge engineer. A threshold m_A has been used for separating 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 $f_{NN}(M(s)) \geq m_A$. With $m_A = 0.65$, about 97% of the training sequences have been recognized as belonging to their proper class (i.e., relevant or not-relevant). Note that we abandoned the idea of reaching a recognition rate of 100% on the training set (not necessary in this case), thus avoiding problems related to overfitting.

During the experiments carried out, the FFNN identified several sequences useful for abstraction but not created as a refinement of an existing abstract-operator. It is worth pointing out that all sequences filtered out by the FFNN were in fact relevant. An example is given in figure 4 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 presented. Further information (i.e., the abstract-operator pre- and post-conditions, as well as a fragment of domain ontology) is reported to let the reader better understand the underlying mechanism.

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 0.62)
	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 0.56)
	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)

Figure 4 - An example of sequence identified as relevant for abstraction, together with its corresponding macro- and abstract-operator schemata.

4 Conclusions and Future Work

In this paper, an adaptive mechanism devised for identifying abstract operators, within a framework that supports a hierarchical interleaving planning and execution approach, has been described.

The overall adaptive life cycle, that encompasses plan chunking, sequence filtering, macro- and abstract-operators generation has been briefly depicted. In particular, starting from successful plans, relevant sequences are searched for by exploiting the capabilities of a feedforward neural network, trained with suitable positive and negative examples.

Newly created abstract operators can be made available to the abstract level for any subsequent planning problem to be solved, thus making it possible to evolve individual abstract operators according to the past experience of each agent.

As for the future work, we are currently investigating the problem of estimating the "degree of completeness" of an abstraction mapping, to give agents the capability of automatically updating their own set of "active" abstract operators. Furthermore, we are currently investigating under which constraints the "soundness" property can be

obtained in domain ontologies that allow both *is-a* and *part-of* relations.

References

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

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

[3] 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, pp. 125-146, 1990.

[4] Y. Gil and M.A. Perez. Applying a general-purpose planning and learning architecture to process planning. In *Planning and Learning: On to Real Applications: Papers from the 1994 AAAI Fall Symposium.* AAAI Press. pp. 48-52., 1994.

[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), pp. 79-95, March 1998.

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

[8] C. A. Knoblock. *Automatically Generating Abstractions for Problem Solving*. Ph.D. Thesis. CS Department. Carnegie Mellon University, 1991.

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

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

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

[12] C. Leckie and L. Zukerman. Learning search control rules for planning: An inductive approach. In *Proc. of Machine Learning Workshop*. pp. 422-426, 1991.

[13] R.A. Levinson. Exploiting the physics of state-space search. In *Proc. of AAAI Symposium on Games:Planning and Learning*. AAAI Press. pp. 157-165, 1993.

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

[15] 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. pp. 7-14, 1998.

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

[17] D.A. Plaisted. Theorem proving with abstraction. *Artificial Intelligence*. Vol.16(1), pp. 47-108, 1981.

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

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

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