Proceedings of the First International Conference on Machine Learning and Cybernetics, Beijing, 4-5 November 2002

ON LEARNING CONTROL KNOWLEDGE FOR A HTN-POP HYBRID PLANNER

SUSANA FERNÁNDEZ, RICARDO ALER Y DANIEL BORRAJO

Universidad Carlos III de Madrid, 28911 Leganés (Madrid), España
E-MAIL: sfarregu, aler@inf.uc3m.es, dborrajo@ia.uc3m.es

Abstract:

In this paper we present a learning method that is able to automatically acquire control knowledge for a hybrid HTN-POP planner called HYBIS. HYBIS decomposes a problem in subproblems using either a default method or a user-defined decomposition method. Then, at each level of abstraction, it generates a plan at that level using a POCL (Partial Order Causal Link) planning technique. Our learning approach builds on HAMLET, a system that learns control knowledge for a total order non-linear planner (PRODIGY4.0). In this paper, we focus on the operator selection problem for the POP component of HYBIS, which is very important for efficiency purposes.

1 Introduction

In this paper we present a system that learns control knowledge by generating a bounded explanation of the problem solving episodes. It is used in combination with a planner which solves real world problems from manufacturing systems (HYBIS [5]). HYBIS is a hierarchical and nonlinear planner with an automata-based representation of operators, which is able to obtain control sequences for manufacturing processes. It mixes hierarchical (HTN) [7] and Partial Order Planning (POP) techniques [19]. The description of the problems that appear in a manufacturing system consists of a set of transformations which must be performed on raw products in order to obtain the manufactured ones. A domain is a knowledge-based model of the manufacturing system. The model is divided into: a set of agents, which represents the set of actuators (devices); their operations and their interconnections described by the model of actions; and a set of axioms, which describe facts that are always true. Every agent is described hierarchically according to the different parts it is made of, which can also be other agents. In this context, a planning problem consists of: an initial state, which is represented as a conjunction of literals which describe both the manufacturing system and the raw products; and a set of goals, represented as a conjunction of literals that describe the desired manufactured products. The output will be a plan than defines the transformations to be performed to obtain manufactured products from raw ones.

As it is the case for all domain-independent planners, HYBIS is not necessarily fast when searching for plans, since it has to spend time studying in valid alternatives. To avoid this, we propose to automatically acquire knowledge to guide the planning process. This knowledge is based on the experience in solving previous real problems. In planning, several approaches have been used successfully in order to guide the search process by adding control knowledge to the planning procedure, either by learning this control knowledge [1,4,8,10,14,16], or by adding it directly by a human [3]. Perhaps, the most basic scheme for learning control knowledge has been deductive learning techniques that generate control rules from a single or a set of problem solving episodes and a correct domain theory. This is the case of pure EBL techniques [12,14], and techniques built on top of it [1]. These rules will be used in future situations to prune the search space. They allow to improve both the search efficiency of the problem solver and, in some cases, the quality of the generated plans. Another related work applies various learning algorithms to induce task hierarchies, instead of control knowledge [9,11,13,15].

The paper is organized in five sections. Section 2 overviews the planner and the manufacturing domains to which it has been applied. Section 3, discusses the learning process. Section 4 shows empirical results from different domains from manufacturing systems. Finally section 5 draws conclusions and describe future work.

2 The planner. HYBIS

The planner HYBIS mixes hierarchical and POCL techniques to approximate planning techniques to the way that control engineers reason to design control programs [5]. These control programs obtain real world solutions for manufacturing systems. The design of a correct and complete industrial control program is very complex, even for human experts. Traditionally control engineers have used different methodologies, standards, formal tools and computer utilities to carry out this task. The ISA-SP88 [2] standard is one of such methodologies used to hierarchically design control programs for manufacturing systems. This standard allows for a hierarchical specification of physical, process and control models of a manufacturing system. A planning domain is represented as a hierarchy of agents where the root (a *dummy* agent) represents the whole plant, leaf nodes are **primitive agents** corresponding to the field devices of the plant, and intermediate nodes are **aggregate agents.** The structure and behavior of the aggregate agents represent a composition of a set of agents at lower levels of abstraction. Each aggregate agent has knowledge on different alternatives for performing its activity at the next level of detail, so this is equivalent to the different methods used in HTN for decomposing a given operator. Each agent follows a finite automaton behavior. They are in a state, that can be changed through the actions (operators) that are defined inside the agent. Each aggregate agent has a **interface.** That is a set of rules to transform its activity into a problem at the next level in the planning hierarchy. Each aggregate activity can also have a property called **expansion**, to define different alternatives transforming the action into a problem of the next level. If an aggregate activity hasn't defined any expansion method, then the planer uses the interface of the corresponding agent to do this transformation.

2.1 Example of domain definition

The ITOPS domain, extracted from [18], can be used as an example of domain definition. Figure 1 shows a high level diagram of the plant. This domain contains the following primitives agents and products:

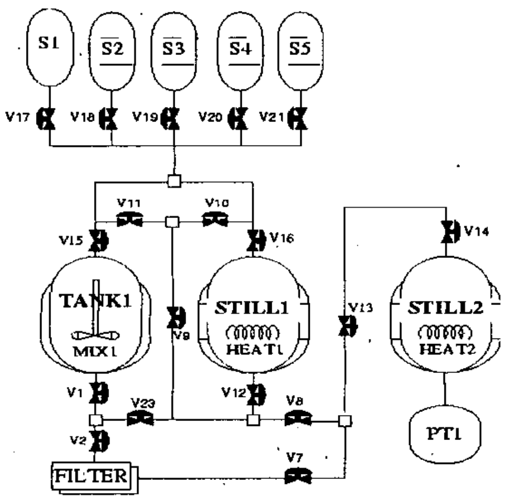

Figure.1. ITOPS Plant. An example of domain for HYBIS.

- Products: R1 to R5. They are initially in the tanks S1 to S5. I1 to I4 are the intermediate products obtained by the reaction of these products, following the scheme:
    - Mix R1, R2, and R3 to result in I1
    - Filter I1 results in I3
    - Heat R4, R5 and I3 resulting in I4
- Valves: V1 to V23
- Mixers: MIX1
- Heaters: HEAT1, HEAT2
- Distillers: DISTILLER1, DISTILLER2
- Filters: FILTER

The description of the hierarchical composition of the system is shown in Figure 2, that shows its three levels of abstraction.

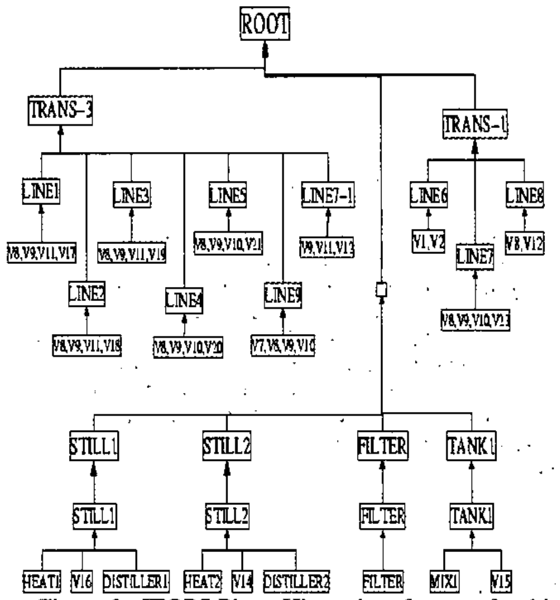

Figure.2. ITOPS Plant. Hierarchy of agents for this domain and problem.

2.2 The planning algorithm

The planning process is a generative and regressive planning algorithm at different levels of detail. Each plan at a given level of abstraction is refined into lower level plans, until no aggregate activities exist on the lowest abstraction level of a hierarchical plan. At each level, the plans are generated by MACHINE [6] using a POP approach. The input to the whole HYBIS planner is a domain description (hierarchy of agents), a problem to be solved (recipe at the highest abstraction level, or procedure level recipe in SP88) and the initial states of the devices and products. Then it proceeds as follows:

- First, by means of a generative POP process it obtains a sequence of control activities to be carried out by the highest level agents.
- Second, if the sequence obtained is only composed by primitive activities, then the problem is solved. Otherwise, the sequence is hierarchically refined; that is, the algorithm expands every aggregate activity, according to its agent interface and its default method or any other method specifically defined, obtaining a new lower level problem.
- Third, the algorithm recursively proceeds to solve the new problem by the agents at the next level.

Therefore, the final plan obtained by this algorithm is a hierarchy of control sequences at different granularity levels. The reader is referred to [5] for more details on the planning algorithm.

3 The learning mechanism

In order to learn control knowledge for this HTN-POCL planner, we follow a three step approach:

1. The planner is run on a planning problem. Then the planning search tree is labelled so that the successful decision nodes are identified.
2. At successful decision points, control rules are created in such a way that were the planner to be run again on this problem, only the right decision would be taken.
3. The control rules are parameterized (constants are converted to variables), so that they can be applied to other problems involving other objects with different names.

3.1 Labelling the search tree

The algorithm assigns four kinds of labels to the nodes:

- *success*, if the node belongs to a solution path
- *failure*, if it belongs to a failed path
- *abandoned*, the planner started to expand this node but the heuristic preferred other nodes and it was abandoned
- *unknown*, if the planner did not expand the node

When a node is generated it is labelled as *unknown*. The nodes which failed in the planning process are labelled as *failure*. When the planner finds a solution, all the nodes in the success path are labelled as *success*. After labelling these nodes, it labels the rest of the nodes bottom-up recursively:

- If all of the successors of a node have failed, then it is labelled as failure
- If an unknown node has, at least, one successor then it is considered abandoned

Once the search tree has been labelled, two kinds of decisions points (i.e. nodes) are considered as candidates for learning control rules:

- Failure-Success: these are nodes which have at least two branches, one with a success node and another with a failure node
- Abandoned-Success: the same as above but instead of a failure node it has an abandoned node

When it finds any of these decisions points a control rule is generated, as explained next section. Obviously, if all successor nodes are successful, no control knowledge is required.

3.2 Generating control rules

At decision nodes with some un-successful successors, control rules are generated so that the planner always selects in the future the successor node. More generally, control knowledge, can either select a node, reject it, or prefer one over another [17]. In this paper, we have focused on the most straightforward sort, namely select rules.

In hybrid HTN-POCL planners, there are also different types of nodes where rules can be learned:

1. HTN points: how to downward refine (which expansion method should be used?)
2. POCL points:
    1. Whether to use an already existing operator or a new one to achieve a goal
    2. In both cases, which operator should be selected?
    3. Whether to promote or demote an operator to solve a threat

In this paper we have studied the operator selection problem. Particularly, we learn SELECT OPERATOR-PLAN (to select an operator already present in the plan to achieve an unsolved goal) and SELECT OPERATOR-NEW (to select a hitherto unused operator to achieve a goal). The kind of rule to be learned depends on what the planner did.

The control rule has a template for describing its preconditions. The templates share a set of common features for both kinds of control rules, but each one has certain local features. Examples of common features, which become meta-predicates, of the control language, are: htn-level, true-in-state, current-goal, some-candidate-goals. Examples of local features for each one of the two kinds of control rules are: Operator-in-plan and Operator-not-in-plan. They are described below.

Variables may appear in the conditions of the control rules. Every variable can only match with a certain kind of objects, a type, which is coded as a prefix in the variable name (what appears before the mark %%). Typing preserves semantics and makes the matching process more efficient. The condition part of a control rule is made of the meta-predicates that were defined above.

- HTN-LEVEL meta-predicate to know the abstraction level in which the planner is working.
- A CURRENT-GOAL meta-predicate to identify which goal the planner is trying to achieve
- A SOME-CANDIDATE-GOALS meta-predicate to identify what other goals need to be achieved
- An OPERATOR-NOT-IN-PLAN meta-predicate so that an OPERATOR-NEW rule is activated only if the operator to insert was not already present. Similarly, OPERATOR-PLAN rules include the OPERATOR-IN-PLAN meta-predicate to make sure the action to be reused is already in the plan.
- Finally, there is a TRUE-IN-STATE meta-predicate for every literal which is true in the initial state. Actually, in order to make the control rules more general and reduce the number of TRUE-IN-STATE meta-predicates, a goal regression is carried out. Only those literals in the initial state which are required, directly or indirectly, by the preconditions of the operator are included. The regression of the preconditions is done by using the causal-link structure.

A control rule following the previous template would become activated only at the appropriate nodes. However, all the arguments of the predicates of the TRUE-IN-STATE meta-predicates are constants, because only particular objects appear in the initial state literals. To avoid that the rule depends on the names of the particular planning problem used for learning, constants are generalized into variables that belong to the same type as the constant.

Actually, not all constants are parameterized. In some cases, it makes no sense to generalize them. For instance, let us consider the literal (STATE TRANSPORTER-3 OFF). TRANSPORTER-3 is a good candidate for parameterization, but OFF is not, because in that case the meaning that a transporter object is off would be lost. Currently, we do not generalize the second argument of STATE predicates. In the future, we would like to detect such cases automatically, although it does not seem an easy task.

The next control rule is an actual example that has been generated after this process. It is an OPERATOR-NEW control rule that selects to use a new action not in the partial plan, FILTRATE(<FILTER>,<RESULT>), when the planner decides to work on goal CONTAINS(<I3>,<?SRC330>), and it is true in the initial state the literals that appear as arguments of the meta-predicate TRUE-IN-STATE.

```
(control-rule regla-1
 (if (and (current-goal (contains <i3>
<?src330>))
(some-candidate-goals
        ((state <line-3prod%%trans-3> off)
         (state <still%%still1-agg> off)
         (state <still%%still1-agg> ready)))
       (operator-not-in-plan
        (filtrate <filter-agg%%filteragg-agg>
<?result>))
       (true-in-state (state <line-3prod%%trans-
3> off))
       (true-in-state (state <tank%%tank1-agg>
off))
       (true-in-state
           (state <filter-agg%%filteragg-agg>
off))))
 (then select operator-new
    (filtrate <filter-agg%%filteragg-agg>
<?result>))))
```

4 Experiments and results

We have tested our approach in several domains. Basically, we want to check whether the rules are correct and save resources in the planning process. If the preconditions of the rules are not specific enough, a rule can be fired in a wrong point and the planner can choose the wrong operator. Because the rule discards the unselected alternatives, the correct ones might be pruned from the search tree, and the planner could not find the solution. This is specially relevant in the manufacturing domains where there are several agents belonging to the same type with the same named actions. For example, in the ITOPS domain there are 10 valves of type *valve-fwd* and all of them have 2 actions *OPEN* and *SHUT*. It is completely different to open one valve or another but it is difficult to distinguish that automatically. Thus, it is very important to determine whether the learning process is producing correct rules.

The experiments have been carried out with one problem in different domains. We obtained all the control rules for all the levels by running the planner with one problem. Then we run the planning process again with the same problem twice, one using the rules learned before and the other without the rules, and we compared the results. The characteristics of the domains and the problems are shown in Table 1. It displays the number of agents (Agents), the number of levels (Levels), the total number of actions (Actions), the number of initial states (Inits) and the number of goals (Goals).

Table 2 shows the results of running the planner with and without rules. It displays the number of nodes generated by the planner process, the time (in seconds) until it finds the solution, the savings in time to solve due to the use of rules, and the number of used rules.
It can be observed that nodes and time decrease when the rules are used. Also, control rules are correct, because they solve the problems.

5 Conclusions and future work

Nowadays, it is often claimed that the most commonly used planners in industry are HTN planners. In this approach, plans are built at different levels of a hierarchy, starting with a high level one and refining them towards the bottom, more specific, level. It is the task of the users to provide methods to step from one level to a lower level. Systems with higher autonomy can be devised. For instance, HYBIS is a hybrid hierarchical planner which provides a default method to step from one level to another. This plan refinement requires to solve a new planning problem, which is performed by a partial order planner (POP). However, although using a hierarchy limits the computational complexity, the process is still inefficient. Moreover, in a hybrid planner like HYBIS, efficiency can be gained both at HTN and POP decision points. Machine learning techniques have been used in older planners to improve the search process by means of previous experience. In this paper, we discuss some of the issues on machine learning applied to this kind of planners. We have extended some machine learning ideas, to deal with hybrid HTN-POP planners. In particular, we have focused in a decision point where the planner has to decide whether to apply an operator already in the plan or not, and in any case, which operator to apply.

In the near future we intend to carry out experiments to check that the control rules generalize to unseen planning problems in the same domain and similar domains (i.e. industrial plants that have more (or less) agents of the same type as in the original plant, or with common levels in the agent hierarchy). We also want to measure the effectiveness of the rules in terms of time and plan quality. It is also planned to extend this learning scheme so that control rules can be inductively specialized, generalized, and combined by using new learning planning problems.

There are many other issues that we would like to address in the future. In particular, we intend to learn control knowledge for all the decision points of HYBIS, including the HTN points. In addition, HYBIS is an agent-based planner, where some agents are made of some other agents. Capturing this part-of information would be useful to include more semantics into the control rules. Also, there is some other information about the connections between agents which is distributed in the domain description that would be interesting to capture as well. Finally, HYBIS has been extended to be able to generate conditional plans, which offers new learning opportunities.

Table 1. Domains characteristics.

Domain	AGENTS	LEVELS	ACTIONS	INITS	GOALS
ITOPS	42	3	92	63	1
BC-2	19	2	44	27	5
PORRIDGE	26	2	61	46	3
HANDLER	15	3	46	26	1
PLANT-3	10	2	20	16	2

Table 2. Results of the execution of HYBIS with rules and without rules

Domain	No rules		With rules			
	NODES	TIME	NODES	TIME	SAVINGS	RULES
ITOPS	898	244	639	212	13%	59
BC-2	637	60	319	34	43%	36
PORRIDGE	583	62	326	40	35%	40
HANDLER	235	21	185	15	29%	33
PLANT-3	198	9	125	4	55%	14

Acknowledgements

This work was partially supported by a grant from the Ministerio de Ciencia y Tecnología through project TAP1999-0535-C02-02. The authors would also like to thank Luis Castillo and Juan Fernández for their help on using HYBIS.

References

[1] Ricardo Aler, Daniel Borrajo, and Pedro Isasi. Using genetic programming to learn and improve control knowledge. Artificial Intelligence, 2002.

[2] ANSI/ISA. Batch Control Part I, Models & Terminology (S88.01), 1995.

[3] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. Artificial Intelligence, 116:123-191, 2000.

[4] Daniel Borrajo and Manuela Veloso. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review Journal*. Special Issue on Lazy Learning, 11(1-5): 371-405, February 1997.

[5] Luis Castillo, Juan Fernández-Olivares, and Antonio González. A hybrid hierarchical/operator-based planning approach for the design of control programs. In ECAI Workshop on Planning and configuration: New results in planning, scheduling and design, 2000.

[6] Luis Castillo, Juan Fernández-Olivares, and Antonio González. Mixing expressiveness and efficiency in a manufacturing planner. Journal of Experimental and Theoretical Artificial Intelligence, 13:141-162, 2001.

[7] Ken Currie and Austin Tate. O-Plan: the open planning architecture. Artificial Intelligence, 52(1): 49-86, 1991.

[8] Tara A. Estlin and Raymond J. Mooney. Learning to improve both efficiency and quality of planning. In Martha Pollack, editor, Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97), pages 1227-1232. Morgan Kaufmann, 1997.

[9] A. Garland, K. Ryall, and C. Rich. Learning hierarchical task models by defining and refining

exaples. In In First International Conference on Knowledge Capture, 2001.

[10] Yi-Cheng Huang, Bart Selman, and Henry Kautz. Learning declarative control rules for constraint-based planning. In Pat Langley, editor, Proceedings of the Seventeenth International Conference on Machine Learning, ICML'00, Stanford, CA (USA), June-July 2000.

[11] Okhtay Ilghami, Dana S. Nau, Héctor Muñoz-Avila, and David W. Aha. Camel:learning method preconditions for htn planning. In Proceedings of AIPS02, 2002.

[12] Subbarao Kambhampati. Improving graphplan's search with ebl & ddb techniques. In Thomas Dean, editor, Proceedings of the IJCAI'99, pages 982-987, Stockholm, Sweden, July-August 1999. Morgan Kaufmann Publishers.

[13] Amnon Lotem and Dana S. Nau. New advances in graphhtn: Identifying independent subproblems in large HTN domains. In *Artificial Intelligence Planning Systems*, pages 206-215, 2000.

[14] Steven Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach.* PhD thesis, Computer Science Department, Carnegie Mellon University, 1988. Available as technical report CMU-CS-88-133.

[15] M. van Lent and J Laird. Learning hierarchical performance knowledge by observation. In *Proceedings of the 16th International Conference on Machine Learning*, pages 229-238, San Francisco, CA, 1999. Morgan Kaufmann.

[16] Manuela Veloso. *Planning and Learning by Analogical Reasoning*. Springer Verlag, December 1994.

[17] Manuela Veloso, Jaime Carbonell, Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI*, 7:81-120, 1995.

[18] S. Viswanathan, C. Johnsson, R. Srinivasan, V. Venkatasubramanian, and K-E. Arzen. Procedure synthesis for batch processes: Part I. knowledge representation and planning framework. *Computers and Chemical Engineering*, 22:1673-1685, 1998.

[19] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27-61, 1994.