On Learning Control Knowledge for a HTN-POP Hybrid Planner

Susana Fernández, Ricardo Aler, and Daniel Borrajo

Departamento de Informática, Universidad Carlos III of Madrid, 28911 Leganés (Madrid), Spain email:{sfarregu@inf aler@inf 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, 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 problems solving episode applied to a planner which solves real world problems from manufacturing systems (HYBIS [5]). HYBIS is an 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 [15]. 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 represents a conjunction of literals which describe both the manufacturing system and the raw products; and a goal, which is a conjunction of literals that describe the transformations needed to obtain the manufactured products from the raw ones.

As it is the case for all domain-independent planners, HYBIS does not always finds the best plan fast, since it can spend time studying non valid alternatives, until it reaches the right solution. 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 [11,8,9,12,4,1], 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 underlying domain theory. This is the case of pure EBL techniques [11,10], and techniques built on top of it [1]. These rules are 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.

The paper is organised in six sections. Section 2 overviews the planner and the manufacturing domains to which it can be applied. Section 3, discusses the learning process. Finally section 4 draws conclusions and 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 programmers. Traditionally control engineers have been using 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. In this sense, the planner employs hybrid POCL and hierarchical planning techniques in order to:

- represent an industrial plant as a device hierarchy at different levels of granularity, which accepts SP88 descriptions, providing a friendly input level for control engineers, and
- autonomously develop control programs for manufacturing systems following a hybrid planning process (POCL+hierarchical), which results in a hierarchy of control sequences (plans) at different levels of detail, closer to the way that humans develop modular industrial control programs and, thus, providing a more understandable output.

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 behaviour 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 behaviour.

They are in a state, that can be changed through the actions (operators) that are defined inside the agent. A problem description is a specification of a process on products, i.e., a recipe. A problem is represented as an ordered set of literals which represents the process to be carried out by the aggregate agents of the highest abstraction level.

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) and a problem to be solved (recipe at the highest abstraction level, or procedure level recipe in SP88). That recipe is preprocessed in order to build a hierarchical plan H-Plan with a single abstraction level, containing a set of literals which represent the problem stated by the recipe. Then, the inputs to HYBIS are the hierarchical `Domain`, the initial abstraction `Level` (the highest one is 1), an initialised task `Agenda` and the initial hierarchical `H-Plan`. 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.

2.1 Example of domain definition

An industrial plant is conceived as a multi-agent domain where every agent represents the knowledge about the relevant properties and behaviour of every factory device. In the planner, the behaviour of every agent is described as an automaton, and every transition of the automaton is represented as a control activity. The planner uses an expressive and rich language in order to represent actions as intervals and, in addition, to handle different kinds of conflicts and interferences which may arise in complex domains like manufacturing systems.

The ITOPS domain, extracted from [14], 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:

- 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:

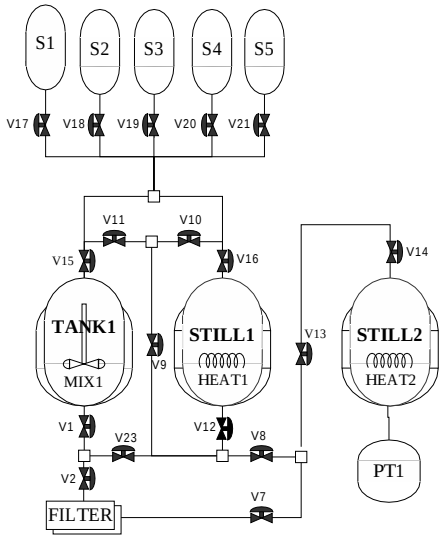

Figure 1. ITOPS Plant. An example of domain for HYBIS.

  - 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: DISTILER1, DISTILLER2
- Filters: FILTER

The description of the hierarchical composition of the system is shown in Figure 2. It has three levels of abstraction and the highest level is composed by the following agents:

- Transports: TRANS-1 and TRANS-3. The components are valves and circuits. Both agents can perform two actions: `transport` and `stop-transport`. But, while TRANS-1 only transports one product, TRANS-3 transports any combination of three products
- Transformers: TANK1, STILL1, STILL2 and FILTER

The agents that appear in the next hierarchical level are: circuits, LINE1 to LINE9; and transformers, the same as the higher level but with different primitive agents. The process to carry out consist in obtaining product I4 from the distiller STILL1.

3 The learning mechanism

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

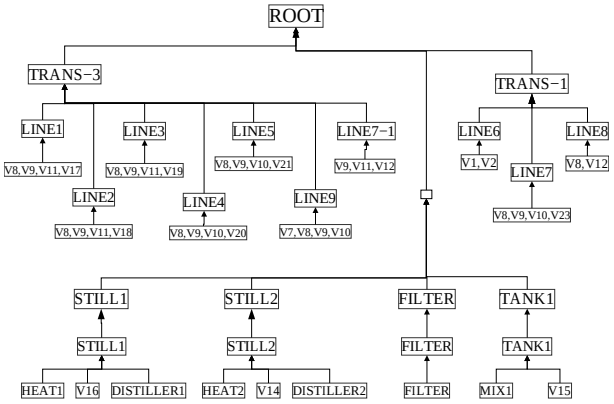

Figure 2. ITOPS Plant. Hierarchy of agents for this domain and problem.

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 tried.
3. Constants in the control rules are generalized, so that they can be applied to other problems involving other objects with different names.

In the future, we plan to extend this learning scheme so that control rules can be inductively specialized, generaliced, and combined by using new planning problems.

Finally, learning can take place at two different moments:

1. Just before a downward refinement so that rules are learned only for one level of abstraction, the one before the refinement (and the highest).
2. After the planner finds a complete solution. In that case, the planner returns a complete search tree involving all levels in the hierarchy, and control rules can be learned for all of them.

In this paper, we have followed the first approach, although it is not difficult to use the whole search tree in a similar way.

3.1 Labelling the search tree

In order to label the whole search tree, the leaf nodes are labelled first. The algorithm assigns four kinds of labels to the leaf 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.

After labelling the leaf nodes, it labels the rest of the nodes bottom-up recursively:

– If a node has a successful succesor, then it is considered successful.
– If all its successor nodes have failed, then it is labeled as faillure
– Otherwise, it is considered unknown

Once the search tree has been labelled, two kind 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 other with a faillure node
– Abandoned-Success: the same as above but instead of a failure node it has an abandoned node

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

3.2 Generating control rules

At decision nodes with some non-successful succesors, control rules are generated so that the planner always selects the successor node. More generally, control knowledge, can either select a node, reject it, or prefer one over another [13]. In this paper, we have focused on the most direct 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:
   (a) Whether to use an already existing operator or a new one to achieve a goal
   (b) In both cases, which operator should be selected?
   (c) 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 (to select an operator already present in the plan to achieve an unsolved goal) and SELECT NEW-OPERATOR-PLAN (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 predicates, or rather metapredicates, [1] of the control language, are:

[1] Actually, they are called metapredicates, because their arguments are predicates themselves.

– True-in-state <assertion>: tests whether the <assertion> is true in the initial state of the planning problem.
– Current-goal <goal>: tests whether the <goal> is the one that the planner is trying to achieve.
– Some-candidate-goals <goals>: tests whether any of the goals in the <goals> set is a pending goals.

Examples of local features for each one of the two kinds of control rules are:

– Operator-in-plan <action>: tests whether the <action> is already in the plan (for OPERATOR-PLAN)
– Operator-not-in-plan <action>: tests whether the <action> is not in the plan (for OPERATOR-NEW)

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.

Finally, the condition part of a control rule is made of three main parts:

– A CURRENT-GOAL metapredicate to identify which goal the planner is trying to achieve
– A SOME-CANDIDATE-GOALS metapredicate to identify what other goals need to be achieved
– A OPERATOR-NOT-IN-PLAN metapredicate so that an OPERATOR-NEW rule is activated only if the operator to insert was not already present. Similary, OPERATOR-PLAN rules include the OPERATOR-PLAN metapredicate to make sure the action to be reused is already in the plan.
– Finally, there is a TRUE-IN-STATE metapredicate for every literal which is true in the planning initial state. Actually, in order to make the control rules more general and reduce the number of TRUE-IN-STATE metapredicates, 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 metapredicates 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 as explained. 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 have 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 metapredicate `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>))))
```

So far, we have tested our approach in the ITOPS domain, and observed that they prune the search tree as expected. 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 that some levels in the agent hierarchy are preserved). We also want to measure the effectiveness of the rules in terms of time and plan quality.

4 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.

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, for which we plan to use case based learning techniques. 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 conections 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.

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. Also in the book "Lazy Learning", David Aha (ed.), Kluwer Academic Publishers, May 1997, ISBN 0-7923-4584-3.
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. 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.

10. 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.
11. 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.
12. Manuela Veloso. *Planning and Learning by Analogical Reasoning*. Springer Verlag, December 1994.
13. 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.
14. 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.
15. Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.