MACHINE LEARNING IN HYBRID HIERARCHICAL AND PARTIAL-ORDER PLANNERS FOR MANUFACTURING DOMAINS 

Susana Fernández, Ricardo Aler, and Daniel Borrajo

Departamento de Informática, Universidad Carlos III of Madrid, 28911 Leganés (Madrid), Spain
email:{susana.fernadez, ricardo.aler, daniel.borrajo}@uc3m.es

Abstract The application of AI planning techniques to manufacturing systems is being widely deployed for all the tasks involved in the process, from product design to production planning and control. One of these problems is the automatic generation of control sequences for the entire manufacturing system in such a way that final plans can be directly used as the sequential control programs which drive the operation of manufacturing systems. HYBIS is a hierarchical and nonlinear planner whose goal is to obtain partially ordered plans at such a level of detail that they can be used as sequential control programs for manufacturing systems. Currently, those sequential control programs are being generated by hand using modelling tools. This document describes a work whose aim is to improve the efficiency of solving problems with HYBIS by using machine learning techniques. It implements a deductive learning method that is able to automatically acquire control knowledge (heuristics) by generating bounded explanations of the problem solving episodes. The learning approach builds on HAMLET, a system that learns control knowledge in the form of control rules.

1 Introduction

A manufacturing system consists of a set of processes, machines and factories where raw products are transformed into higher value manufactured products. The design of the sequential control program for a manufacturing system, i.e the ordered sequence of actions with all of the actions of devices needed to transform the raw products into the manufactured ones, is a difficult task which is traditionally carried out by engineers by hand using modelling tools. Artificial intelligence techniques are proving to be very useful in solving this task, allowing for an error-free, fast and low cost building process of these control programs [23,26,41,42,47,50,52]. In particular, it is becoming an area of growing interest for researchers from the field of AI planning systems [4,15,37,63]. The most commonly used planners in industry are based on hierarchical planning techniques [21,27,55] which are useful to represent the hierarchical structure of

devices and their operation in manufacturing systems and are closer to the way control engineers represent a control program (modular and hierarchical). A hierarchical planner has the effect of increasing the level of abstraction at which a human user can operate since it can allow problems to be posed as high-level goals and work out the detailed implications. This makes planning quicker and also less prone to error as low-level interactions between actions are dealt with automatically. A good example is HYBIS [16], a hybrid planner which solves real world problems from manufacturing systems. It mixes Hierarchical Task Network (HTN) approaches and Partial Order Planning (POP) techniques [65]. It 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 POP.

In this paper we propose to improve the efficiency of solving problems with HYBIS, by automatically acquiring knowledge to guide the planning process. This knowledge is based on the experience in solving previous real problems. HYBIS has two main steps. The first one decomposes an action into another set of actions in a lower level of the hierarchy (this is the HTN part). Although the user can program specific decomposition methods, by default, HYBIS poses the decomposition problem as a planning problem for a lower level in the hierarchy. In any case, decomposition can be done in many different ways, and it is useful to learn control knowledge at this point so that good methods are tried first. The second step calls a partial order planner at every level in the HTN hierarchy. POP planners include decision points such as whether to insert a new operator in the plan, or to use an already existing one, which operator to insert/reuse, etc. Our approach starts by obtaining a trace of HYBIS after solving a planning problem. This trace is a search tree, whose nodes are tagged by the algorithm as successful, failure, abandoned or unexplored. At successful nodes, our method learns control rules by goal regression and generalisation. If the planner is executed again with the learned control rules, only the successful nodes would be explored, with the subsequent efficiency gain.

The paper is organised in six sections. Section 2 explains the manufacturing systems, the planning problem and compares our approach with previous related work. Section 3 overviews the planner. Section 4 discusses the learning process. Section 5 shows empirical results for different domains from manufacturing systems. Finally section 6 draws conclusions and comments on future work.

2 Background on planning in manufacturing systems

The aim of this section is twofold. First, the manufacturing system domain is explained by adopting a representation of the manufacturing resources based on agents. Agent-based representation is an approach commonly used, although there are others [2,6,9,28,36,53]. Second, it compares our approach with previous related work, not only within the AI planning field, but also within the context of the wider scientific literature, particularly in Operation Research and hyper-heuristics.

2.1 Manufacturing systems domains

The definition of manufacturing systems and their main characteristics are widely explained in [15]. In this section, we will point out to the main ideas. A manufacturing system is the set of processes, machines and factories where raw products are transformed into higher value manufactured products. These transformations are carried out by the devices of the manufacturing system. In an agent approach, every device can be represented as an agent whose operation is described by a finite state automata. Thus, every agent has a set of possible states and a set of actions. Each action describes a transformation as well as a change of state in the agent. Additionally, an agent has a name, which must be unique, a set of variables which are used to represent the objects related to the operation of the agent (like products, chemicals, interconnection points between agents or constants) and a set of codesignation constraints defined on the set of variables, which defines the set of valid values for every variable. Every action of every agent is defined by a unique name, a set of effects, which is represented by means of an addition list, and a deletion list of literals that represent the transformation made by the action, and a set of requirements. These are divided into a list of previous requirements, that must hold before the action, a list of simultaneous requirements which must hold during the execution of the action and a list of later requirements, that must hold after the action.

The description of the problems that appear in a manufacturing system consists of a set of transformations which must be made on raw products in order to obtain the manufactured ones. A problem is defined by the following components:

- Domain: a knowledge-based model of the manufacturing system. It is comprised of: a set of agents, which represent the set of devices, their operations and their interconnections described by the model of action; and a set of axioms, which describe facts that are always true
- Initial state: a conjunction of literals which describe what is true at the starting point of both the manufacturing system, and the raw products
- Goal: a conjunction of literals which describe the transformations needed in order to obtain manufactured products from raw ones. This is known in manufacturing as a recipe [3]. In AI planning a goal is a conjunction of literals which describes a desired partial state instead of a partially ordered set of subgoals which describes a desired behaviour, as it is in manufacturing

The solution of a problem is an ordered sequence of actions of the agents of the domain which achieves the goal, starting from the specified initial state. This is called a control sequence (or a plan in planning) and it is a difficult task which is usually carried out by engineers. Traditionally control engineers have been using different methodologies, standards, formal tools and computer utilities to carry out this task. The ISA-SP88 [3] 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.

Manufacturing planning researchers typically want to solve a particular manufacturing problem, and present their research results within the context of this problem, without discussing how the approach might generalise to other planning domains. This can lead AI planning researchers to view manufacturing planning as a domain full of ad-hoc, domain-specific programs rather than general principles and approaches [50]. But manufacturing systems are evolving into a new generation of systems driven by the demands of a constantly changing market and they must adapt to their changes in a timely fashion. There is no other alternative, but to generalise to other similar planning domains. This is specially certain nowadays where the tendency is to manufacture under demand, flexibly and quickly, allowing to produce different products in the same plant even if there are faults such as broken machines or any other type of failure.

2.2 Related work

There are close synergies of the research carried out here on planning and other scientific areas. In particular, the field of Operations Research has an extensive area in manufacturing scheduling [10,30,31,35,59]. AI planning and scheduling complement each other, and overlap to some extent, but tackle different problems. AI planning is the application of artificial intelligence technologies to the problem of generating a correct and efficient sequence of activities, chosen from a knowledge base of generic actions, which, when executed, will move a system from some initial state to some desirable end-state. This sequence is typically a partial order one; only essential ordering relations between the actions are considered, so that actions allow a pseudo-parallelism and can be executed in any order for achieving the desired goals. On the other hand, scheduling is applied to the related problem of efficiently, or sometimes optimally, allocating time and other resources to an already-known sequence of actions (plan). Scheduling can therefore be seen as selecting among the various action sequences implicit in a partial-order plan in order to find the one that meets efficiency or optimality conditions with respect to time constrains, and filling in all the resources details to the point at which each action can be executed.

In planning, several approaches have been successfully used in order to guide the search process by adding control knowledge to the planning procedure, either by learning this control knowledge [1,8,22,32,45,62], or by adding it directly by a human [5]. 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 [34,45], and techniques built on top of it [1]. These rules can be used in future situations to prune the search space. Others use case based reasoning for production scheduling [7,19,57], but they focus on scheduling rather than on the planning aspect. As far as we know, our approach is the only one which learns control knowledge for HTN-POP planners and it can be applied to solve real world problems from manufacturing systems.

Machine learning has also been extended to hierarchical planners. The system HICAP (*Hierarchical Interactive Case-based Architecture for Planning*) [46] inte-

grates the SHOP hierarchical planner [49] together with a case-based reasoning (CBR) system named NaCoDAE [11]. Hicap has been used to assist with the authoring of plans for noncombatant evacuation operations. KnoMic(Knowledge Mimic) [60] is a machine learning system which extracts hierarchical performance knowledge by observation to develop automated agents that intelligently perform complex real world tasks. The knowledge representation learned by KnoMic is a specialised form of Soar's production rule format [38]. The rules implement a hierarchy of operators with higher level operators having multiple sub-operators representing steps in the completion of the high level operator. These rules are then used by an agent to perform the same task.

CaMeL [33] uses an incremental algorithm to learn expansions methods in a HTN planner under supervision of an expert. In [24] a technique called programming by demonstration is used to build a system in which a domain expert performs a task by executing actions and then reviews and annotates a log of the actions. This information is then used to learn hierarchical task models.

GIPO-II is a GUI able to create and maintain hierarchical domain specifications, and verify them using a structural property checker, and plan using the forward hybrid task-reduction planner HyHTN [43]. It is based on the previously released GIPO [56], the GUI and tools environment for building classical planning domain models, providing help for those involved in knowledge acquisition and the subsequent task of domain modelling. It integrates the algorithm *opmaker* to induce operator schema from example sequences [44]. Although all these systems are applied to hierarchical planners that solve problems of the real world, they differ from our approach in the learning method. While we used deductive learning, they apply inductive learning. Also, they focus on learning the domain model, not the heuristics as it is our case. One advantage of deductive learning approaches is that it is enough to provide a few solved problems, opposed to inductive methods that require a big amount of examples.

A recent application of machine learning and rule-based techniques on planning has been done to build an adaptive planning system, called HAP that can automatically fine-tune its parameters based on the values of specific measurable characteristics of each problem [64]. Adaptation is guided by a rule-based system, whose knowledge has been acquired through machine learning techniques. It uses the DM-II program [39] that performs classification based on association rules [40] in order to discover useful and interpretable rules from the data. Neither they learn heuristics nor they use deductive learning.

An extensive survey of research work related to machine learning applied to automated planning can be found in [66].

Other works related to our learning approach are those that have been devoted to hyper-heuristics. The term hyper-heuristic was first used in early 2000 to describe heuristics (or meta-heuristics) which choose heuristics [18]. The hyper-heuristics manage the choices of which lower-level heuristic method should be applied at any given time, depending on the characteristics of the region of the solution space currently under exploration. Hyper-heuristics could be clas-

sified depending on the learning mechanism employed, such as genetic algorithms [12,29], reinforcement learning [48], case-base learning [13], ... These systems work with the idea that there are known heuristics, and they use inductive learning methods that require many training examples. Our learning method is able to acquire heuristics in the form of control rules without considering the heuristics that might (or not) use the planner. It implements a deductive learning algorithm that is able to generate the control rules from one sample problem.

3 The planner. HYBIS

HYBIS is a hierarchical and nonlinear planner with an automata-based representation of operators, which is able to obtain control sequences for manufacturing processes [16]. The aim of HYBIS is to provide a tool for helping humans on defining such programs. Also, it allows to easily adapt to changes in the manufacturing plant requirements.

A domain model in HYBIS represents an industrial plant as a device hierarchy at different levels of granularity, which accepts ISA-SP88 descriptions. 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. Properties and behaviour of a given aggregate agent are related with those of its components by means of the **interface** of the aggregate agent. The user can also program specific decomposition **methods** defining the **aggregate action**. The expansion slot of an aggregate activity is used to specify different ways to carry out that activity. These alternative ways are described as a set of different methods where every method is represented by a set of ordered literals representing a problem to be solved by the agents of the next lower abstraction level. All the aggregate actions have a default method which is obtained from the interface of the agent.

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 to be carried out by the aggregate agents of the highest abstraction level, to obtain the manufactured products from the raw ones.

3.1 Example of domain definition

The MIXTURE domain, which is inspired in a real factory, can be used as an example of domain definition. Figure 1 shows a high level diagram of the plant. It is composed of 26 varied interconnected agents (pumps, valves, mixers, heaters, belts and a bottler), 46 different actions and 2 abstractions levels. The goal in this domain consists of adding an ingredient (flour), initially contained in `ADDITIVE-1`, to the milk, initially contained in `MILK-TANK`, and then proceed to bottle the mixture. The milk and an enzyme are transformed into ferment in

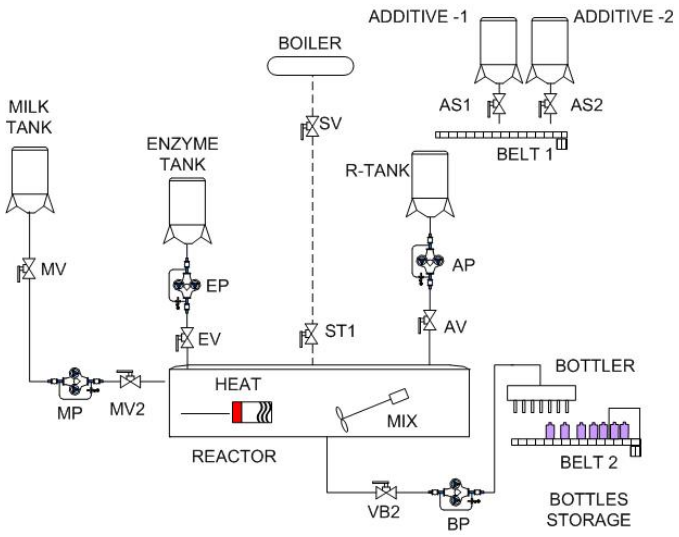

Figure1. A real world manufacturing system, MIXTURE, that produces an elaborated mixture from milk, enzyme and flour.

the `REACTOR`. The result is mixed with the flour to produce the final mixture. The design of a low level control program for such a system is difficult, even for a control engineer, due to the number of agents and the total number of available actions.

3.2 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 a POP system, MACHINE [17]. 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). A hierarchical plan `H-Plan`, initially only with the highest abstraction level, is partially builded, containing the set of literals which represent the problem stated by that recipe. The inputs to the hybrid algorithm 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, as in MACHINE, it obtains a sequence of control activities to be carried out by the highest level agents
- Second, if the sequence obtained is only composed of 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. In a POP approach, planning is conceived as a search through plan space where nodes denote plans [65]. A plan is represented as a tuple $< A, O, L >$ in which A is a set of actions, O is a set of ordering constraints over A, and L is a set of causal links. A causal link [58] is a data structure with three fields: two contain pointers to plan actions (the link's producer, A_p, and its consumer, A_c); the third field is a proposition, Q, which is both an effect of A_p and a precondition of A_c. Such a causal link is written as $A_p \rightarrow^Q A_c$ and store a plan's links in the set L. Causal links are used to detect when a newly introduced action interferes with past decision. It is called a *threat*. There are some ways to protect against threats but the most common are demotion and promotion. POP also requires another data structure called *Agenda*. This is composed of a set of tasks, each of which describes a pending problem in the plan, a flaw. The different pending problems which may be found in *Agenda* are: pending subgoals that represent unmet preconditions; threats from actions to causal links; interferences between actions; and order inconsistencies caused by a loop in the order structure, which must be a strict order. Flaws in the *Agenda* are ordered following a LIFO strategy. Unsolved subgoals are solved by an existing action in the partial plan or by the addition of a new action. Threats are solved by promotion or demotion of operators. Order inconsistencies can not be solved. The search process to solve the tasks in the *Agenda* is the depth first engine over the set of choices to solve every task and the well known heuristic evaluation function which accounts for the number of steps and the number of open conditions [25,54].

The start point is a null *Plan* with two dummy actions *Start* and *End* which encode the planning problem $P =< D, I, G >$ in a domain D. The initial state I is encoded as the addition list of the action *Start*, the goal G is encoded as an unsatisfied preconditions of action *End*. The *Agenda* initially contains only these pending subgoals specified in the goal G and the causal links set is initially empty. Some changes were necessary to integrate the HTN part with the POP approach in order to inherit all the data structures through the different levels of abstraction. When an action must be refined in a lower abstraction level, this is considered as a new flaw to be inserted in the *Agenda* and it will be solved by any of its decomposition methods. The heuristic function that guides the search also needs to be adapted for integration with the HTN part.

The reader is referred to [16] for more details on the planning algorithm and to [65] for partial order planning.

4 The learning mechanism

The planner defined in the previous section is complete, but not necessarily efficient even with the domain-independent heuristics it uses. In some situations, the heuristics do not guide the planner towards the best search path. Learning from experience can help to identify the particular situations for which the domain-independent heuristics need to be overridden. In this section, we propose a machine learning mechanism to improve HYBIS efficiency based on previous experience. It generates explanations for the local decisions made during the search process. These explanations become control rules that are used in future situations to prune the search space. In order to learn control knowledge for this HTN-POP planner, we follow a three step approach:

1. The planner runs 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 generalised, so that they can be applied to other problems involving other objects with different names

Now, we will present each step in more detail.

4.1 Labelling the search tree

As many other planners, HYBIS generates a search tree for every planning problem. There are four kinds of nodes in the search tree:

- *success*, if the node belongs to a solution path. It represents one of the actions of the control sequence that constitutes the solution of the problem
- *failure*, if it is a failed node or all its successor nodes have failed. The failure takes place when a flaw cannot be solved, i.e there is no action to solve a pending subgoal, there is no way of solving a threat, or the order relation is violated
- *abandoned*, the planner started to expand this node but the heuristics of HYBIS preferred other nodes and it was abandoned before it failed
- *unknown*, if the planner did not expand the node. When a task is removed from the *Agenda*, it generates as many nodes as ways to solve it are. Depending on the values of the heuristic function, only one node every time is chosen to be expanded. It can happen that there are nodes whose heuristic value is worse than the values of the rest of the nodes, so they never get expanded.

All nodes are initially labelled as *unknown*. If a node fails during the planning process, its label changes to *failure*. Once the planner finds a solution, all the nodes that belong to the solution path are labelled as *success* and a bottom-up recursive algorithm assigns the *failure* or *abandoned* label to the rest of the nodes.

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 children; one a *success* node and another a *failure* node
- *Abandoned-Success*: the same as above, but, instead of a failure node, they have an *abandoned* node

Figure 2 shows an example of a labelled search tree and the decision points where the rules are generated. For example, N6 represents a flaw with three possible ways to solve it, which generate three nodes N7, N8 and N9. N8 and N9 fail during the planning process and N7 belongs to the solution path. So, it is a decision point in which one children is a successful node (N7) and another is a failure one (N8), so a control rule is generated (Rule-1). Another example is N21; there are three possible ways to solve the corresponding task: N22 was never chosen to expanded (the value of the heuristic function for this node is worse than the value of other nodes) and it remains with the initially *unknown* label; N23 is expanded but none of its children fail nor become part of the solution path, so the labelled algorithm assigns it the *abandoned* label; and finally, N24 is a successful node. Therefore, this is the other type of decision points from which we learn, one children is an *abandoned* node (N23), and another is a *success* one (N24), so Rule-3 is generated.

4.2 Generating control rules

At the learnable decision nodes, control rules are generated so that the planner will select the successful node on later problem solving episodes. More generally, control knowledge can either select a node, reject it, or prefer one over another [61]. We have focused on the most direct sort, namely select rules.[1]

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

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

In this work we have studied the operator selection and the downward refine problems. Particularly, we learn the following kinds of control rules:

- `SELECT OPERATOR-PLAN`, to select an operator already present in the plan to achieve an unsolved goal

[1] Selecting a node means also rejecting any other alternative, according to the semantics of our control rules.

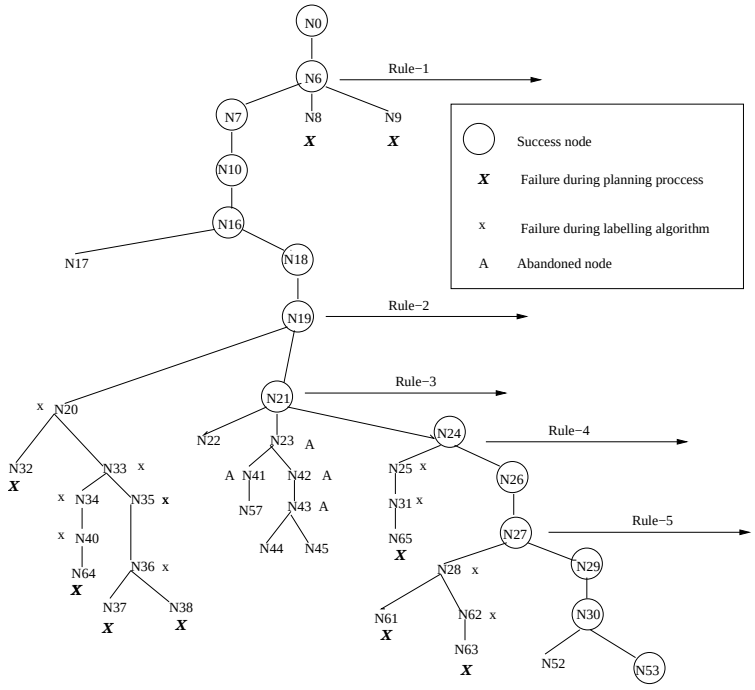

Figure2. An example of a search tree and some of its decision points. The learning algorithm would learn from nodes N6, N19, N21, N24 and N27.

- `SELECT OPERATOR-NEW`, to select a hitherto unused operator to achieve a goal, and
- `SELECT EXPANSION`, to select the expansion method to refine an aggregate action

The kind of rule to be learned depends on what the planner did. All control rules have a typical rule format (for instance, *IF conditions THEN consequent*) but the condition part and the consequent are different in each type of rule.[2] One of the most important decisions to be made in learning systems refer to the language to be used to represent target concepts. In our case, the condition part is made of a conjunction of meta-predicates which check for local features of the search process:

- `HTN-LEVEL` to know the HTN-LEVEL in which the planner is currently working
- `CURRENT-GOAL` to identify which goal the planner is currently trying to achieve
- `SOME-CANDIDATE-GOALS` to identify what other goals need to be achieved

[2] This is based on the meta-predicates defined by the PRODIGY planner [14].

- `OPERATOR-NOT-IN-PLAN` so that a `SELECT OPERATOR-NEW` rule is activated only if the operator to insert was not already present. Similarly, `SELECT OPERATOR-PLAN` rules include the `OPERATOR-IN-PLAN` meta-predicate to make sure the action to be reused is already in the plan. These meta-predicates also check that the arguments of the operator verify the constraints of the agent (this information is in the domain description). All the operators have at least one argument which represents an agent. The rest of arguments, if they exist, must be included in the constraints of that agent
- `TRUE-IN-STATE` to test that a literal 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` meta-predicates, a goal regression is carried out as in most EBL techniques [20]. Only those literals in the initial state which are required, directly or indirectly, by the preconditions of the operator involved in the rule are included. The regression of the preconditions is done by using the causal-link structure of the partial ordered component of the HYBIS planner. As we said before, a causal link is represented as $A_p \rightarrow^Q A_c$ where A_p is a plan action that produces a literal Q, which is a precondition of another plan action A_c. The algorithm to obtain the regressed state R of an action A_i from the causal link structure L is:

Q = preconditions (A_i)
$R = \varnothing$
while $Q \neq \varnothing$
    $q = 1^{st}$ element of Q
    if $q \in$ Initial State
    then $R = R \cup q$
    else
        A_q = action which adds q according with L $((A_q \rightarrow^q A_i) \in L)$
        Q_r = preconditions (A_q)
        $Q = Q \cup Q_r$
    remove q from Q

  For example, if
  $L = \{A_0 \rightarrow^{Q_1} A_1, A_0 \rightarrow^{Q_2} A_4, A_2 \rightarrow^{Q_3} A_3, A_1 \rightarrow^{Q_4} A_3, A_4 \rightarrow^{Q_5} A_2\}$ is the set of causal links and A_3 the action that was selected in the success node we are learning from, the goal regression would proceed as follows: Q_3 and Q_4 are the preconditions of A_3 as the causal links $A_2 \rightarrow^{Q_3} A_3, A_1 \rightarrow^{Q_4} A_3$ show. A_2 and A_1 are different from A_0 so their preconditions would be computed, Q_1 from causal link $A_0 \rightarrow^{Q_1} A_1$ and Q_5 from causal link $A_4 \rightarrow^{Q_5} A_2$, $Q1$ belong to the initial state but $Q5$ don't, so the precondition of A_4 would be computed from causal link $A_0 \rightarrow^{Q_2} A_4$. Therefore, the initial literals required would be (Q_2, Q_1)
- `OPERATOR-TO-EXPAND` to identify which action the planner is trying to expand in order to do a downward refine. This is only for the rules of `SELECT`

EXPANSION. This meta-predicate also checks that the arguments of the operator verify the constraints of the agent, in the same way as OPERATOR-IN-PLAN does
– COMPONENTS-OF to identify the agents of a lower level that compound an aggregate agent

An expansion method is a set of literals representing subgoals to be achieved for the agents of the next level of abstraction. Sometimes, these subgoals are nothing but states to be reached for the component agents of the aggregate agent whose action is trying to refine. In that case, a COMPONENTS-OF meta-predicate is necessary. This led us to have two kinds of expansion rules; one when the expansion method is a set of literal of the type (STATE <AGENT> state) and another kind for the rest of situations.

Summarising, there are four different types of control rules, two related to the POP points and two concerned with the HTN decision points, with an specific template:

1. POP nodes (o decisions) to use a new operator to achieve a goal:

```
(control-rule name1
   (IF (AND (HTN-LEVEL (level))
            (CURRENT-GOAL (goal))
            (SOME-CANDIDATE-GOALS (list-of-subgoals))
            (OPERATOR-NOT-IN-PLAN (operator arguments))
            (TRUE-IN-STATE (literal1))
            ....
            (TRUE-IN-STATE (literaln))))
   (THEN SELECT OPERATOR-NEW (operator arguments)))
```

   Figure 3 shows an example of a rule which selects to use a new action not in the partial plan, (ONLINE ADDITIVE-TRANSPORT1), when the planner is at level 1, it is working on goal (CONTAINS FLOUR RTANK), there is at least one of the pending subgoals which appears as argument of the meta-predicate SOME-CANDIDATE-GOALS, and in the initial state the literals (CONTAINS FLOUR ADDITIVE1) and (STATE ADDITIVE-TRANSPORT1 OFF) are true.
2. POP nodes to use an operator already existing in the partial plan to achieve a goal:

```
(control-rule name2
   (IF (AND (HTN-LEVEL (level))
            (CURRENT-GOAL (goal))
            (SOME-CANDIDATE-GOALS (list-of-sugoals))
            (OPERATOR-IN-PLAN (operator arguments))
            (TRUE-IN-STATE (literal1))
            ....
            (TRUE-IN-STATE (literaln))))
   (THEN SELECT OPERATOR-PLAN (operator arguments)))
```

   Figure 4 shows an example of a rule which selects to reuse the action START already in the partial plan when the planner is at level 1, it is working on

```
(control-rule rule-1
 (IF (AND (HTN-LEVEL (1))
          (CURRENT-GOAL (CONTAINS FLOUR RTANK))
          (SOME-CANDIDATE-GOALS ((STATE ADDITIVE-UNIT OFF)
                                 (CONTAINS FERMENT REACTOR)
                                 (STATE  ADDITIVE-UNIT OPEN)
                                 (STATE  REACTOR READY)
                                 (BOTTLED MIXTURE)))
          (OPERATOR-NOT-IN-PLAN (ONLINE  ADDITIVE-TRANSPORT1))
          (TRUE-IN-STATE (CONTAINS FLOUR ADDITIVE1))
          (TRUE-IN-STATE (STATE ADDITIVE-TRANSPORT1 OFF))))
 (THEN SELECT OPERATOR-NEW
          (ONLINE ADDITIVE-TRANSPORT1)))
```

Figure3. Control rule. Example 1

goal (STATE REACTOR READY), there is at least one of the pending subgoals which appears as argument of the meta-predicate SOME-CANDIDATE-GOALS, and in the initial state the literal (STATE REACTOR READY) is true.

```
(control-rule rule-2
 (IF (AND (HTN-LEVEL (1))
          (CURRENT-GOAL (STATE REACTOR READY))
          (SOME-CANDIDATE-GOALS ((ADDED MILK ENZYME)
                                 (CONTAINS MIXTURE  REACTOR)
                                 (BOTTLED MIXTURE)))
          (OPERATOR-NOT-IN-PLAN (START))
          (TRUE-IN-STATE (STATE REACTOR READY))))
 (THEN SELECT OPERATOR-PLAN (START)))
```

Figure4. Control rule. Example 2

3. HTN nodes to select an expansion method which is a set of literal representing the state of several lower level agents:

```
(control-rule name3
   (IF (AND (HTN-LEVEL (level))
            (OPERATOR-TO-EXPAND (operator arguments))
            (TRUE-IN-STATE (literal1))
            ....
            (TRUE-IN-STATE (literaln))
            (COMPONENTS-OF (agent-agg (agent1 ... agentr)))))
   (THEN SELECT EXPANSION ((STATE agent1 ESTADO1)
                           ...
                          (STATE agentm ESTADOm))))
```

Figure 5 shows an example of rule which selects to use the expansion method ((STATE HEAT1 OFF) (STATE ST1 OFF) (STATE SV OFF)), when the planner decides to work on the downward refinement of the action (OFF REACTOR ?RESULT171),

the compounded agents of REACTOR are (HEAT1 ST1 SV) and in the initial state the literals that appear as arguments of the meta-predicate TRUE-IN-STATE are true. This rule would force the HTN component of the planner to refine the (OFF REACTOR ?RESULT171) into ((STATE HEAT1 OFF) (STATE ST1 OFF) (STATE SV OFF)) at the next level of abstraction.

```
(control-rule rule-3
 (IF (AND (HTN-LEVEL (2))
          (OPERATOR-TO-EXPAND (OFF REACTOR ?RESULT171))
          (TRUE-IN-STATE (STATE SV OFF))
          (TRUE-IN-STATE (STATE ST1 OFF))
          (TRUE-IN-STATE (STATE HEAT1 OFF))
          (COMPONENTS-OF (REACTOR
                          (HEAT1 ST1 SV)))))
 (THEN  SELECT EXPANSION
          ((STATE HEAT1 OFF) (STATE ST1 OFF) (STATE SV OFF))))
```

Figure5. Control rule. Example 3

4. HTN nodes to select an expansion method different from previous case:

```
(control-rule name4
   (IF (AND (HTN-LEVEL (level))
            (OPERATOR-TO-EXPAND (operator arguments))
            (TRUE-IN-STATE (literal1))
            ....
            (TRUE-IN-STATE (literaln))))
   (THEN SELECT EXPANSION (expansion)))
```

Figure 6 shows an example of a rule which selects to use the expansion method (CONTAINS FERMENT REACTOR), when the planner decides to work on the downward refinement of the action (REACT REACTOR FERMENT) and in the initial state the literal (STATE REACTOR READY) is true.

```
(control-rule rule-4
 (IF (AND (HTN-LEVEL (2))
          (OPERATOR-TO-EXPAND (REACT  REACTOR FERMENT))
          (TRUE-IN-STATE (STATE REACTOR READY))))
 (THEN SELECT  EXPANSION
          ((CONTAINS FERMENT REACTOR))))
```

Figure6. Control rule. Example 4

4.3 Generalising

To avoid that rules depend on the names of objects or agents of the particular planning problem used for learning, constants are generalised into variables that belong to the same type as the constants. 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 %%). For example, if during the planning process a constant `REACTOR` appears it would be generalised into `<REACTOR-MIXTURE %%REACTOR>` because `REACTOR` is defined in the domain as an agent of type `REACTOR-MIXTURE`. In fact, only constants which represent agents of the domain are typed. The rest of planning constants (like products, chemicals or interconnection points between agents) are generalised without the type part. Typing preserves semantics and makes the matching process more efficient.

Actually, not all constants are parameterised as explained. In some cases, it makes no sense to generalise them. For instance, let us consider the literal `(STATE REACTOR READY)`. `REACTOR` is a good candidate for parameterisation, but `READY` is not, because, in that case, the meaning that a reactor object is ready would be lost. Currently, we do not generalise the second argument of `STATE` predicates, because they usually have this meaning.

The `SELECT EXPANSION` rules cannot be completely generalised either. In the meta-predicate `OPERATOR-TO-EXPAND` only the first argument of the operator, i.e the agent, is generalised because the rest of the arguments influence the selection of the correct expansion method. For example, there are two possible ways to expand the operator `REACT` depending on if it is `(REACT REACTOR FERMENT)` or `(REACT REACTOR MIXTURE)`, so we only generalise `REACTOR`. In other words, the control rule might be generalised for any `REACTOR` but not for every product to be generated.

This generalisation scheme is very planner dependent, but it is domain independent. The example 1 rule of figure 3 would be generalised as:

```
(control-rule rule-1
 (IF (AND (HTN-LEVEL (1))
          (CURRENT-GOAL (CONTAINS <FLOUR> <RTANK>))
          (SOME-CANDIDATE-GOALS ((STATE <NOZZLELINE-SIMPLE%%ADDITIVE-UNIT> OFF)
                                 (CONTAINS <FERMENT> <REACTOR-MIXTURE%%REACTOR>)
                                 (STATE  <NOZZLELINE-SIMPLE%%ADDITIVE-UNIT> OPEN)
                                 (STATE  <REACTOR-MIXTURE%%REACTOR> READY)
                                 (BOTTLED <MIXTURE>)))
          (OPERATOR-NOT-IN-PLAN (ONLINE  <TRANSPORTLINE%%ADDITIVE-TRANSPORT1>))
          (TRUE-IN-STATE (CONTAINS <FLOUR> <ADDITIVE1>))
          (TRUE-IN-STATE (STATE <TRANSPORTLINE%%ADDITIVE-TRANSPORT1> OFF))))
 (THEN SELECT OPERATOR-NEW
          (ONLINE <TRANSPORTLINE%%ADDITIVE-TRANSPORT1>)))
```

4.4 Use of learned control knowledge

The learned control rules are used in future planning episodes in the following way. The planning algorithm cycle starts by removing a task from the *Agenda* and then finds the possible ways to solve it, generating a list of choices which

will become new nodes of the search tree. Depending on the type of task, the choices can be: existing actions in the partial plan or new actions (if the task is to achieve a subgoal); demotion o promotion actions (if the task is to solve a threat); or expansion methods (if the task is an action downward refine). Then all the control rules are matched against the current meta state of the search, generating another list of choices by means of the consequents (RHS) of the matched rules. The intersection of both lists will constitute the possible ways to solve the task and the new nodes to add to the search tree. In case that the intersection is empty, the first list is used (no control rule matched or they selected an incorrect choice). A rule matches when all its preconditions (LHS) are fulfilled. Actually, a control rule can be fulfilled in many different ways. The matching algorithm generates all of them. Each possible matching is represented by a list of bindings (i.e lists of pairs variable/value). The list of bindings are incrementally generated. This is a straightforward implementation of an unification procedure. For example, the meta-predicate `(TRUE-IN-STATE (STATE <VALVE-FWD%%SV> OFF))`, solving a problem whose initial state are the literals:
`((STATE SV OFF), (STATE SV1 OFF), (STATE HEAT1 OFF))`,
would generate two pairs of bindings:
`(<VALVE-FWD%%SV>.SV) and (VALVE-FWD%%SV>.SV1)`, assuming that `SV` and `SV1` are of type `VALVE-FWD`. If the precondition has another meta-predicate, for example `(TRUE-IN-STATE (STATE <HEATER-3PROD%%HEAT1> OFF))`, a new binding `(<HEATER-3PROD%%HEAT1>.HEAT1)` would be added (if `HEAT1` is of type `HEATER-3PROD`) and the list of bindings would become:

```
(((<VALVE-FWD%%SV>.SV),(<HEATER-3PROD>.HEAT1))
 ((VALVE-FWD%%SV>.SV1), (<HEATER-3PROD>.HEAT1))),
```

which represented two possible ways of satisfying the rule.

5 Experiments and results

The experimentation has been divided in two phases. In the first one, we have used our approach in several domains to test its validity. Basically, we wanted to check whether the rules were correct and saves resources in the planning process. If the preconditions of a rule are too general, the rule can be triggered at a wrong point and the planner may choose the wrong decision. Because the rule discards the alternatives not selected, the correct one might be pruned from the search tree, and the planner will never find the solution. This is specially relevant in manufacturing domains where there are several agents belonging to the same type with the same named actions. Thus, it is very important to determine whether the learning process is producing correct rules. If the preconditions are too specific they might work for just a very small set of problems.

In the second phase, we have checked that the control rules generalise and improve efficiency to unseen planning problems in a similar domain, i.e. industrial plants that have a different number of agents of the same type as in the original plant. We have also studied the utility of the generated rules.

5.1 Correctness of rules

The firsts experiments have been carried out in different domains with a particular problem each one. The domains are composed of interconnected devices (agents), tanks, pumps, valves, mixers, heaters, belts, bottler ..., similar to the ones explained in section 3 (MIXTURE problem). There is one defined problem for each domain consisting of a high level goal that specifies a process to be performed. For example, the ITOPS problem [63] consists of five raw products, r_1, r_2, r_3, r_4 and r_5, initially in five different tanks. The goal is to obtain a final product, i_4, by allowing a reaction to occur among them, according to the following scheme: mix r_1, r_2, r_3 to obtain i_1; filter i_1 to obtain i_3 and heat r_4, r_5 and i_3 to obtain i_4. The other problems are of the same type. A complete description can be found in section 2.8 at [51].[3] From the viewpoint of planning, it is enough to consider that the domains have different characteristics as number of hierarchical levels, number of agents, and number of total actions. And the problems differ in the number of literals in the initial state and goals. 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 literals in the initial states (Inits) and the number of goals (Goals).

Table1. Domains characteristics.

Domain	Agents	Levels	Actions	Inits	Goals
ITOPS	42	3	92	63	1
BC-2	19	2	44	27	5
MIXTURE	26	2	61	46	3
HANDLER	15	3	46	26	1
PLANT-3	10	2	20	16	2

Table 2 shows the results of running the planner with and without rules in a problem per domain. We obtained all the control rules, HTN and POP, for all the levels by running the planner with one problem and one domain. Then we ran 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. It displays the number of nodes generated for the planner process, the time (in seconds) until it finds the solution, the savings (%) in time to solve due to the use of rules, the total matching time of the rules (in seconds) and the number of used rules.

It can be observed that the control rules are correct, because they do not restrain the planner from finding the solution. Also, nodes and time decrease

[3] The problems and domains are available under request for the purpose of scientific comparison.

Table2. Results of the execution of HYBIS with and without rules.

	No rules		With rules				
Domain	NODES	TIME	NODES	TIME	SAVINGS	MATCHING	RULES
ITOPS	995	418	751	410	2%	51	72
BC-2	637	57	304	41	28%	9	40
MIXTURE	583	70	329	56	20%	8	43
HANDLER	235	21	184	20	5%	2	33
PLANT-3	198	7	185	8	0%	1	14

when the rules are used. Finally, the planner found the same solutions with and without rules in all the domains, so there was no variation in plan quality.

5.2 Generalisation of rules to unseen problems

In order to test the generalisation power of the algorithm we have performed experiments in the MIXTURE domain shown in Figure 1. It represents a real manufacturing plant of a dairy product industry which has collaborated in the accomplishment of this project.[4] We considered that this domain is sufficiently complex and significant to verify the total validity of our approach. The goal is to test whether the control rules are also useful if the industrial plant changes (by adding new agents, for instance). We generated all the control rules by running the planner with the problem described in section 3, obtaining 43 rules. Afterwards, we modified the domain by adding more agents to check the generality of the rules. The new domains and problems tested are:

- MIXTURE_2B: a second bottler is added to solve the same problem but the first bottler is not ready to work (in the initial state, one of the preconditions of the operator who activates the bottler is missing). The planner does not detect its no operation until much later in the search tree, so it expands a non-useful part of the search tree, therefore needing more time to find the solution. If the second bottler would be ready to work the planner would find the same solution and we wouldn't test the generalisation.
- MIXTURE_3B: two more bottlers are added to solve the same problem but bottler1 and bottler2 are not ready
- MIXTURE_4B: three more bottlers are added to solve the same problem but only bottler4 is ready
- MIXTURE_2B2R: a second bottler and a second reactor are added. The problem consists of bottling mixtures obtained in each reactor
- MIXTURE_3B2R: same as above but with one more bottler

We also generated all the control rules by running the planner with the MIXTURE_2B problem obtaining another 43 rules. We run the planning process

[4] The company is PULEVA and the project TIC2001-4936-E from the *Ministerio de Ciencia y Tecnología* in Spain.

with each problem three times; first using the rules learned from the original problem (rules_1), second using the MIXTURE_2B rules (rules_2) and the third one without rules.

No control knowledge was extracted from the other problems due to memory space limitations. The learning algorithm explores the search tree, generated during problem resolution, after solving the problem. It then finds the right decision points in the solution path of the tree to generate the control rules. Therefore, the training problems must be of small size to guarantee that the planner will find a solution and that it will be able to keep in memory the search tree. The planner HYBIS offers the posibility of solving a problem either keeping the history of the resolution process (the search tree) or without keeping it. In order to solve the last and more complex problems, the history option was disabled to avoid failures of allocating memory because of a maximum address space limitation. Also, according to our previous experience on learning for other planners and domains, rules learned from complex problems will be less general. A control rule is generated by extracting the meta-state, and performing a goal regression for finding which literals from the initial state were needed. So, the simpler the training problem is, the simpler the meta-state will be (fewer literals in the regressed state, and, therefore, fewer conditions in the control rule). Then, the control rules obtained will be simpler and more general. We plan to use inductive techniques to correct the control-rules, as it has been done for other planners [1,8].

Table 3 shows these results with the two sets of rules. It displays the number of nodes generated for the planning process, the time (in seconds) until it finds the solution and the savings (%) in time to solve due to the use of the rules.

Table3. Results of the execution of rules to unseen problems.

	No rules		**With rules_1**			**With rules_2**		
Domain	NODES	TIME	NODES	TIME	SAVINGS	NODES	TIME	SAVINGS
MIXTURE_2B	2506	630	1463	393	38%	1539	429	32%
MIXTURE_3B	4458	1350	2604	750	44%	2753	862	36%
MIXTURE_4B	6410	1952	3745	1082	45%	3967	1090	44%
MIXTURE_2B2R	4057	1094	808	770	30%	842	911	17%
MIXTURE_3B2R	6821	1455	1091	842	42%	1125	966	34%

It can be observed that nodes and time decrease when the rules are used. The average saved time due to the rules is 36%. We have also measured the total matching time of the rules and it appears not to be too high as Table 4 shows. Column TIME represents the matching time in seconds and column %TOTAL represents the % with respect the total search time. So, it seems there is no big utility problem in this case.

Table4. Total matching time of rules in the different domains

	With rules_1		With rules_2	
Domain	TIME	%TOTAL	TIME	%TOTAL
MIXTURE_2B	50	13%	43	10%
MIXTURE_3B	63	8%	80	9%
MIXTURE_4B	85	8%	84	8%
MIXTURE_2B2R	44	6%	68	7%
MIXTURE_3B2R	55	6%	37	4%

The results using the first rules are better than the results using the other ones in all the domains. This shows the convenience of training with simple problems and then applying the learned knowledge to more complex problems.

6 Conclusions and future work

Manufacturing systems are evolving into a new generation of systems driven by the demands of a constantly changing market and they must adapt to their changes in a timely fashion. AI planning techniques have been very useful to satisfy these needs, and specially HTN planners where the hierarchical semantics of this kind of planning gives us the ability to model planning problems in domains that are naturally hierarchical, such as industry environments. 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. Given that machine learning techniques have been successfully used in older planners to improve the search process by means of previous experience, we have extended them for learning planning heuristics for HTN-POP planners.

The concept of domain in HYBIS differs from other planners. For HYBIS , a domain is an industrial plant, designed specifically to solve a single problem, that is to say, to generate a manufactured product from initial raw materials. However, at some times the plant might undergo a decision modification so that new (but similar) problems can be solved. An example of modification is adding a new bottler, having a broken machine, etc. In that case a new plan for the plant would have to be found and time would be wasted due to inefficiencies of the planner. Although the modification might be minimal, the planning process must begin again from scratch. If learned control rules in the first execution are obtained and these are correct, this second planning episode will be solved much more efficiently.

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. We have also studied hierarchical refinement. Each task can be decomposed into several subtasks using predefined methods. Each possible decomposition represents a new branch in the search space of the problem and control rules can be learned to prune the unsuccessful branches. We have measured the effectiveness of the rules in terms of time and nodes expanded. It has been found that the rules generated are correct and useful. They improve the efficiency of the planner by saving resources (time and memory) during the search process. We have verified the convenience of acquiring the control knowledge from small problems to use it for solving more complex ones afterwards. In case of having an unsolved problem (for the planner) a strategy for solving it could consist on defining a relaxed problem to train the system with. For instance, one in which we define fewer agents or machines in the plant. Then, we allow the learning system to acquire some control rules. Finally, we try to solve the original problem using the learned control rules in the simpler problem.

In the future, we plan to extend the learning scheme so that control rules can be inductively corrected by specializing, generalizing, or modifying them. Here, we want to follow previous approaches that have been successful for other planners (in particular, HAMLET [8] and EVOCK [1]). Also, 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 projects TAP1999-0535-C02-02, TIC2001-4936-E, and TIC2002-04146-C05-05. The authors would also like to thank Luis Castillo and Juan Fernández-Olivares for their help on using HYBIS.

References

1. R. Aler, D. Borrajo, and P. Isasi. Using genetic programming to learn and improve control knowledge. *Artificial Intelligence*, 141(1-2):29–56, October 2002.
2. J. Algeo, M.E.A.and Fowler and K. Jurrens. Computerized representations of manufacturing resources: validation and standardization efforts. In K. Stelson and F. Oba, editors, *Proceedings of the Japan-USA Symposium on Flexible Automation*, volume 1, pages 699–702. ASME, New York, NY, USA, 1996.
3. ANSI/ISA. *Batch Control Part I, Models & Terminology (S88.01)*, 1995.
4. R. Aylett, G. Petley, P. Chung, J. Soutter, and A. Rushton. Planning and chemical plan operation procedure synthesis: a case study. In *Proceedings in Fourth european conference on planning*, pages 41–53, 1997.
5. F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191, 2000.

6. L. Becker and C. Pereira. SIMOO-RT an object-oriented framework for the development of real-time industrial automation systems. *IEEE Transactions on Robotics and Automation*, 18(4):421–30, 2002.
7. A. Bezirgan. An application of case-based expert system technology to dynamic job-shop scheduling. In R. M.A.Bramer, editor, *Proceedings of Expert Systems 92, The Twelfth Annual Technical Conference of the British Computer Society*, pages 225–235, 1992.
8. D. Borrajo and M. 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.
9. A. Braatz. Development of a UML-based function block-model for object-oriented control-design. *Automatisierungstechnische Praxis*, 45(6):38–44, 2003.
10. L. Braccesi, M. Monsignori, and P. Nesi. Monitoring and optimizing industrial production processes. In *Proceedings of the Ninth IEEE International Conference on Engineering of Complex Computer Systems*, pages 213–22, Florence, Italy, 2004.
11. L. Breslow and D. W. Aha. NaCoDAE: Navy conversational decision aids environment. Technical Report AIC-97-018, Washington, DC: Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence, 1997.
12. E. K. Burke, P. Cowling, J. D. Landa Silva, and S. Petrovic. Combining Hybrid Metaheuristics and Populations for the Multiobjective Optimisation of Space Allocation Problems. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 1252–1259, San Francisco, California, 2001. Morgan Kaufmann Publishers.
13. E. K. Burke, S. Petrovic, and R. Qu. Case-based heuristic selection for examination timetabling. In 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02). Nayang Technology Umiversity, NTU Press. CD-ROOM, 2002.
14. J. G. Carbonell, J. Blythe, O. Etzioni, Y. Gil, R. Joseph, D. Kahn, C. Knoblock, S. Minton, A. Pérez, S. Reilly, M. M. Veloso, and X. Wang. PRODIGY4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Department of Computer Science, Carnegie Mellon University, 1992.
15. L. Castillo, J. Fdez-Olivares, and A. González. Automatic generation of control sequences for manufacturing systems based on nonlinear planning techniques. *Artificial Intelligence in Engineering*, 4(1):15–30, 2000.
16. L. Castillo, J. Fdez-Olivares, and A. 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.
17. L. Castillo, J. Fernández-Olivares, and A. González. Mixing expressiveness and efficiency in a manufacturing planner. *Journal of Experimental and Theoretical Artificial Intelligence*, 13:141–162, 2001.
18. P. Cowling, G. Kedall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *Selected Papers of the Third International Conference PATAT 2000*, Lecture Notes in Computer Science, pages 176–190, Konstanz, Germany, 2000. Springer.
19. P. Cunningham and B. Smyth. Case based reasoning in scheduling: reusing solution components. *Interantional Journal of Production Research*, 35:2947–2961, 1997.
20. G. DeJong and R. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.

21. K. Erol, J. Hendler, and D. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Artificial Intelligence Planning Systems*, pages 249–254, 1994.
22. T. A. Estlin and R. J. Mooney. Learning to improve both efficiency and quality of planning. In M. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1227–1232. Morgan Kaufmann, 1997.
23. M. Fabian, B. Lennartson, P. Gullander, S. Andreasson, and A.Adlemo. Integrating process planning and control for flexible production systems. In *Proceedings of ECC'97*, Brussels, Belgium, July 1997.
24. A. Garland, K. Ryall, and C. Rich. Learning hierarchical task models by defining and refining examples. In *In First International Conference on Knowledge Capture*, 2001.
25. A. Gerevini and L. Schubert. Inferring state constraints for domain-independent planning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 905–912, Menlo Park, California, 1998. AAAI Press.
26. Y. Gil. A specification of manufacturing processes for planning. Technical Report CMU-CS-91-179, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, August 1991.
27. F. Giunchiglia. Using Abstrips Abstractions - Where do we stand? *Artificial Intelligence Review*, 13:201–213, 1999.
28. H. Guang-hong and L. Zu-shu. Intelligent control system based on industrial PC for middling and small coal boiler. *Control Engineering China*, 10(4):339–63, 2003.
29. L. Han, G. Kendall, and P. Cowling. An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem.
30. A. Haskose, B. Kingsman, and D. Worthington. Performance analysis of make-to-order manufacturing systems under different workload control regimes. *International Journal of Production Economics*, 90(2):169–86, 2004.
31. J. Heinonen and F. Pettersson. Scheduling a specific type of batch process with evolutionary computation. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 966–70, Canberra, ACT, Australia, 2003.
32. Y. Huang, B. Selman, and H. Kautz. Learning declarative control rules for constraint-based planning. In P. Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning, ICML'00*, Stanford, CA (USA), June-July 2000.
33. O. Ilghami, D. Nau, H. Muñoz-Avila, and D. Aha. Camel: Learning method preconditions for HTN planning. In M. Ghallab, J. Hertzberg, and P. Traverso, editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS-02)*, pages 131–141, Toulouse (France), 23-17 April 2002. AAAI Press.
34. S. Kambhampati. Planning graph as a (dynamic) CSP: Exploiting EBL, DDB and other CSP search techniques in graphplan. *Journal of Artificial Intelligence Research*, 12:1–34, 2000.
35. S. Kaparthi, N. Suresh, , and R. Cervaney. An improved neural network leader algorithm for part-machine grouping in group technology. *European Journal of Operational Research*, 69(3):342–356, 1993.
36. D. Kiritsis, P. Xirouchakis, and C. Gunther. Petri net representation for the process specification language. 1. manufacture process planning. In *First International Workshop on Intelligent Manufacturing Systems (IMS-Europe 1998)*., pages 595–608. Ecole Polytech. Federale de Lausanne, Lausanne, Switzerland, 1998.

37. I. Klein, P. Jonsson, and C. Backstrom. Efficient planning for a miniature assembly line. *Artificial Intelligence in Engineering*, 13(1):69–81, 1998.
38. J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.
39. B. Liu, W. Hsu, Y. M, and S. Chen. Discovering Interesting Knowledge using DM-II. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99)*, San Diego, CA, USA, 1999. Industrial Track.
40. B. Liu, W. Hsu, and Y. Ma. Integrating Classification and Association Rule Mining. In *Knowledge Discovery and Data Mining*, pages 80–86, 1998.
41. G. e. Luger, editor. *AAAI Special Interest Group in Manufacturing Workshop: Artificial Intelligence and Manufacturing, State of the Art and Sate of the Practice*. AAAI Press, 1998.
42. P. Maropoulos. Review of research in tooling technology, process modelling and process planning. Part II: Process planning. *Int. Journal of Computer Integrated Manufacturing Systems*, 8(1):13–20, 1995.
43. T. McCluskey, D. Liu, and R. Simpson. GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment. In N. M. Enrico Giunchiglia and D. Nau, editors, *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS-03)*, pages 92–101, Trento, Italy, June 2003. AAAI Press.
44. T. McCluskey, N. Richardson, and R. Simpson. An interactive method for inducing operator descriptions. In M. Ghallab, J. Hertzberg, and P. Traverso, editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS-02)*, Toulouse (France), 23-17 April 2002. AAAI Press.
45. S. 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.
46. H. Muñoz-Avila, D. W. Aha, L. A. Breslow, R. Weber, and D. Nau. HICAP: An interactive case-based planning architecture and its application to noncombatant evacuation operations. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*, pages 879–885, 1999.
47. H. Muñoz-Avila and J. Huellen. Retrieving cases in structured domains by using goal dependencies. In M. Veloso and A. Aamodt, editors, *Proceedings of the 1st International Conference on Case-Based Reasoning Research and Development*, volume 1010 of *LNAI*, pages 241–252, Berlin, 1995. Springer Verlag.
48. A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning, 2001.
49. D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP: Simple hierarchical ordered planner. In *Proceedings of the IJCAI-99*, pages 968–973, Stockholm (Sweden), August 1999.
50. D. Nau, S. Gupta, and W. C. Regli. AI planning versus manufacturing-operation planning: A case study. In *Proceedings of IJCAI-95*, pages 1670–6, 1995.
51. J. F. Olivares. *Planificación Híbrida para el diseño automático de programas de control industrial*. PhD thesis, Universidad de Granada. E.T.S de Ingeniería Informática. Departamento de Ciencias de la Computación e Inteligencia Artificial, 2001.
52. S. Park, M. Gervasio, M. Shaw, and G. D. Jong. Explanation-based learning for intelligent process planning. *IEEE Transactions on Systems Man and Cybernetics*, 23(6):1597–1616, 1993.

53. J. Peschke. Real-time java for industrial controls in flexible manufacturing systems. In *Proceedings of IEEE International Conference on Industrial Informatics (IEEE Cat. No.03EX768)*, pages 325–31, 2003.
54. M. E. Pollack, D. Joslin, and M. Paolucci. Flaw selection strategies for partial-order planning. *Journal of Artificial Intelligence Research*, 6:223–262, 1997.
55. E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.
56. R. Simpson, T. McCluskey, W. Zhao, A. R.S., and C. Doniat. GIPO: An integrated graphical tool to support knowledge engineering in AI Planning. In *Procedings of the 6th European Conference on Planning (ECP-2001)*, 2001.
57. E. Szelke and G. Markus. A learning reactive scheduler using CBR. *Computer Industry*, 33:31–36, 1997.
58. A. Tate. Project planning using a hierarchic non-linear planner. Research Report 25, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland, 1976.
59. M. Valyov, C. Potts, and V. Strusevich. Batching decisions for assembly production systems. *European Journal of Operational Research*, 157(3):620–42, 2004.
60. 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.
61. M. Veloso, J. Carbonell, A. Pérez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI*, 7:81–120, 1995.
62. M. M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer Verlag, December 1994.
63. 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.
64. D. Vrakas, G. Tsoumakas, N. Bassiliades, and I. Vlahavas. Learning Rules for Adaptive Planning. In N. M. Enrico Giunchiglia and D. Nau, editors, *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS-03)*, pages 82–91, Trento, Italy, June 2003. AAAI Press.
65. D. S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.
66. T. Zimmerman and S. Kambhampati. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, 24(2):73–96, 2003.