Automatic Synthesis of Control Sequences: A Nonlinear Planning Approach*

Luis Castillo and Juan Fdez-Olivares and Antonio González
Departamento de Ciencias de la Computación e Inteligencia Artificial
E.T.S. Ingeniería Informática. Universidad de Granada
18071 Granada, SPAIN.
{L.Castillo,Faro,A.Gonzalez}@decsai.ugr.es

Abstract

This paper presents an approach to the application of artificial intelligence planning techniques to the generation of control sequences for manufacturing systems. These systems have some special features that must be considered in the planning process, but usual models of action present difficulties to deal with them. Therefore, a model of action derived from the classic model of STRIPS is defined and a nonlinear planning algorithm is derived from POP, both able to deal with these features.

Keywords: Nonlinear planning, model of action, manufacturing systems, control sequences.

Introduction

The application of artificial intelligence planning techniques to the automatic synthesis of control sequences for manufacturing systems is becoming an area of increasing interest (Gil, 1991; Klein et al., 1993; Nau et al., 1995; Park et al., 1993; Soutter, 1996). The reason is that they allow for an error-free, fast and low cost building process of such control sequences.

However, the results obtained are not as realistic as one could expect because either plans obtained have not the necessary level of detail, that is, there are actions which should have also been included and they are missing, or the plan is only focused in a small part of the overall manufacturing system. The goal of this paper is twofold, on the one hand, the design of a planning system which obtains plans at a sufficiently level of detail such as to be considered as a control program, what will be called a control sequence, and, on the other hand, to extend these plans to the whole of the manufacturing system in order to obtain more realistic plans, closer to what an engineer would call a control program.

This goal may be approached by increasing the expressiveness of the classical planning systems in order to cope with the actions that take place in a manufacturing system. It is clear that classical planning systems lacks of the necessary expressivity to solve real world problems and that this subject has been widely studied (Pednault, 1989; Penberthy and Weld, 1992; Sandewall and Ronnquist, 1986; Allen, 1984). However, there are some features of manufacturing systems which are difficult to deal with usual models of action. Therefore, a specialized model of action, and in consequence a specialized planning scheme, will be presented which take these features into account in order to obtain more realistic results.

In the next section, these features and their motivation are presented. The next sections are devoted to explaining how both the basic model of action of STRIPS (Fikes and Nilsson, 1971) and the general nonlinear planning scheme described in (Weld, 1994) may be extended in order to deal with these features, configuring a planning scheme called MACHINE. The last sections show some experimental results and how some other interesting features should be included in this planning scheme.

*This work has been supported by the CICYT under project TIC-0453.

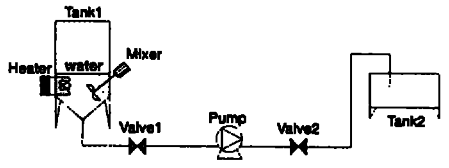

Figure 1: A sample manufacturing system

Description of the Problem

A manufacturing system is the set of processes, machines and factories where raw products are transformed into higher value manufactured products. A very simple manufacturing system is shown in Figure 1.

These transformations are made by the machines of

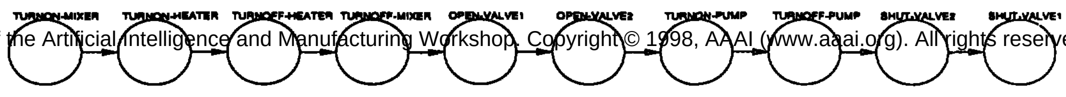

Figure 2: A small control sequence.

the manufacturing system, called actuators. The operation of every actuator is defined by a finite state automata where the states of the automata represent all the conditions in which the actuator is intended to be, and every arc from one state to another represents an action of the actuator. Hence, every action of every actuator implies both, a transformation in the manufacturing system and a change of state in the actuator.

There are many ways of writing a control program for a manufacturing system (GRAFCET, Ladder), but for our purposes, the necessary level of detail to describe a control program is a control sequence[1], that is, an ordered sequence of *all of the actions of actuators needed to transform raw products into manufactured ones.* For example, a possible control sequence to heat and carry the water from *Tank1* to *Tank2* could be like the one shown in Figure 2.

Apparently, a sequence of actions like this could have been generated by any of the state of the art planners, however it has some interesting features that makes it difficult to obtain mainly due to some inherent features of manufacturing domains which must be taken into account. Let us see these features.

Actions as Intervals

Every action of every actuator executes as usual, but it is somehow *active*, that is, it could maintain its effects, until the next change of state in the automata of the actuator. For example, let us consider the action `TurnOn-Mixer`. It is executed in the sequence and the water will become in agitation, but it will be active until the execution of `TurnOff-Mixer`. Therefore, it seems reasonable to consider actions as intervals instead of as isolated points in the sequence, as in classical planners. This is the interval in which the action is considered to be *active.* We will call this interval its interval of execution. For example, the interval of execution for the action `TurnOn-Mixer` would be [`TurnOn-Mixer`, `TurnOff-Mixer`].

There are many approaches in the literature which consider actions as intervals like for example (Allen, 1984; Dean and McDermott, 1987; Rutten and Hertzberz, 1993; Sandewall and Ronnquist, 1986). In some of them, the end of this interval is defined by the achievement of all of its effects and in the others the end of the interval has an implicit relation with the action itself (like for example a known duration). However, none of these conceptions adequately fit in this problem. For example, let us consider the action of opening `Valve1`. It achieves its effects at some point before the starting of `Pump` and it continues *active* until the shutting of `Valve1`, later in the sequence, that is its interval of execution is [`Open-Valve1`, `Shut-Valve1`]. This shows that the interval of `Open-Valve1` is neither defined by the achievement of its effects (it is later) nor has a fixed relation with it, but that it is *active* while there is no change of state in the actuator, that is, until the execution of another action that produces a change of state. If such an action doesn't exist, then the action will continue active until the end of the program.

This is an important feature of the actuators in a manufacturing system. One can plan about the execution of an action and the action will execute, but it will continue *active*, and its effects maintained, if no change of state is produced. The immediate consequence is that if the end of the interval of execution of an action comes from the execution of another action then it must also be planned.

Requirements During the Actions

The second one, and very related to the former, is that, if actions are to be considered as intervals instead of as points, then the requirements which must hold in order to guarantee a correct execution of an action, should also take into account this interval. That is, in addition to classical preconditions, as conditions which must hold *before* the action, it is necessary to define some kind of *simultaneous requirements* as conditions which must hold *during* the interval of execution of the action. These requirements are a form of the *during* relation in (Allen, 1984). For example, let us consider the action of pumping the water. It requires the valves to be open before the pumping starts, but it is also necessary for them to remain open until the end of pumping. This kind of requirements is also present in the literature. They also appear in (Sandewall and Ronnquist, 1986) and later in an application by (Klein et al., 1993), but the difference here is that the interval which defines the protection for these *simultaneous requirements* doesn't end with the achievement of the effects of the action, but rather while the action is ac-

[1]In (Castillo et al., 1998) we describe a method to translate these control sequences into GRAFCET charts and Petri nets, as true representations of a control program.

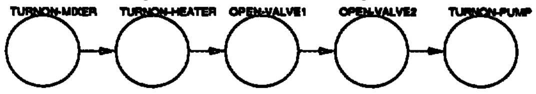

Figure 3: An alternative control sequence

tive, that is until the next change of state produced by the execution of another action. If actions are not considered like explained above, it is difficult to guarantee that valves should be open during the interval of execution of the pump, that is, in the interval [`TurnOn-Pump`, `TurnOff-Pump`], or that the water should be in agitation during the heating of the water or that `Valve1` should be closed during the agitation of the water.

Safe States

And finally, if one thinks about the example from a causal viewpoint, then a strictly correct plan would have also been the one shown in Figure 3 because there is nothing that tells the actuators to be off once the water is hot and it is in *Tank2*.

However, the truth is that there are ***safe states*** in the automata which describes the operation of actuators and that these states must be reached by every actuator before the end of the sequence, so the correct program is actually the one shown in Figure 2. One way to introduce this feature in the process for the building of programs could just be by including these safe states in the goal of the problem. Although this achieves a safe state for every actuator it seems too global, that is, it may be difficult to decide the point in the program in which the actuator reaches a safe state, or even if it would be necessary to use the same actuator later in the program and return it to a safe state. It seems that this decision is specific to each action that doesn't leave it in a safe state. Therefore, the need to leave the actuator in a safe state can be modelled as a later requirement of some actions, that is, as a condition that must hold *after* the action. In the example of the valve, the safe state is the one in which the valve remains shut, so the need to shut it as soon as possible could be modelled like a later requirement of the action which opens the valve.

These are the basic features which must be taken into account in order to enable a planning system to reason about the actions that take place in a manufacturing system. Since they do not fit adequately either into known models of action which consider actions as a point in a sequence, or in others which consider actions as intervals, the following model for actions and plans has been defined.

A Model for Actions and Plans

The model of action needed to deal with the previous features may be obtained by extending the basic model of STRIPS (Fikes and Nilsson, 1971) in order to give actuators a higher importance and to allow for the inclusion of the new types of requirements. Every actuator in a manufacturing system is represented as an *agent* whose operation is described by a finite state automata. Thus, every agent has ***a set of states*** $\mathcal{E}$, which describe all the possible conditions in which it is intended to be, and ***a set of actions*** $\mathcal{A}$, each of whom describe a transformation as well as a change of state in the agent. Additionally, an agent has a *name* $\mathcal{N}$, which must be unique, ***a set of variables*** $\mathcal{V}$, which are used to represent the objects related to the operation of the agent (like for instance products, chemicals, interconnections points between agents or constants) and ***a set of codesignation constraints*** $\mathcal{C}$ defined on the set of variables, which define the set of valid values for every variable.

$$Agent = \langle \mathcal{N}, \mathcal{E}, \mathcal{V}, \mathcal{C}, \mathcal{A} \rangle$$

Every ***action*** of every agent is defined by a unique ***name*** $\mathcal{N}$, a set of effects, which is represented by means of an ***addition list*** $\mathcal{ADD}$, and a ***deletion list*** $\mathcal{DEL}$ of literals that represent the transformation made by the action, and a set of requirements, divided into a list of ***previous requirements*** $\mathcal{ANT}$, that must hold before the action, a list of ***simultaneous requirements*** $\mathcal{DUR}$ which must hold during the interval of execution of the action and a list of *later requirements* $\mathcal{POST}$, that must hold after the action.

$$Action = \langle \mathcal{N}, \mathcal{ADD}, \mathcal{DEL}, \mathcal{ANT}, \mathcal{DUR}, \mathcal{POST} \rangle$$

Example 1 *This example roughly shows how* `Valve2` *seen previously could be described by this model (using a Lisp-based notation).*

```
(AGENT
  (N      Valve2)
  (E      OPEN SHUT)
  (V      ?SOURCE ?IN ?OUT ?CHEM)
  (C      (?SOURCE NIL) (?IN (PUMP))
          (?OUT (TANK2)) (?CHEM NIL))
  (A
    (ACTION (N    Open-Valve2)
      (ADD  (STATE Valve2 OPEN)
            (OPEN-FLOW ?CHEM ?SOURCE ?OUT))
      (DEL  (STATE Valve2 SHUT))
      (ANT  (STATE Valve2 SHUT)
            (OPEN-FLOW ?CHEM ?SOURCE ?IN)
            (CONTAINS ?CHEM ?SOURCE))
      (DUR  (OPEN-FLOW ?CHEM ?SOURCE ?IN))
      (POST (STATE Valve2 SHUT)))
    (ACTION (N    Shut-Valve2)
      ...))
)
```

As may be seen, the interval of execution of an action of an agent is not included in its definition because it doesn't depend on itself but on the inclusion of another action of the same agent that changes the state reached by 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. Although most of the manufacturing processes are quite complex, in this paper only simple transformations are considered; however, they are expressive enough to show the main difficulties during the building process of a control sequence. Thus, a problem $\mathcal{P} = \langle \mathcal{D}, \mathcal{I}, \mathcal{G} \rangle$ is defined by the following components.

A domain $\mathcal{D}$ is a knowledge-based model of the manufacturing system and it is divided into a set of agents, which represents the set of actuators, their operation and their interconnections described by this model of action, and a set of axioms, which describe facts which are always true.

The initial state $\mathcal{I}$ is a conjunction of literals which describe the initial state of both the manufacturing system, and the raw products.

A goal $\mathcal{G}$ is a conjunction of literals which describe the transformation needed to obtain manufactured products from raw ones.

Plans

The solution to these problems consists in an ordered sequence of actions of the agents of the domain which achieves the goal starting from the specified initial state. This can be called a control sequence or an operation procedure (Soutter, 1996), but in this paper it will also be called a plan.

Example 2 *Let us consider the manufacturing system shown in Figure 4. Let us suppose that we already*

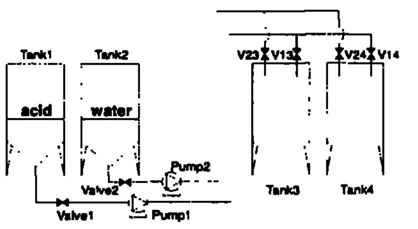

Figure 4: A second manuf. system

have a knowledge-based model of this system, built by means of the previous model of action, and that the initial state shows the valves closed, and the pumps off. Then, Figure 5 shows a plan to carry the water to TANK3*, where* START *and* END *are two dummy actions defined with the same meaning as in SNLP (McAllester and Rosenblitt, 1991) or UCPOP (Weld, 1994).*

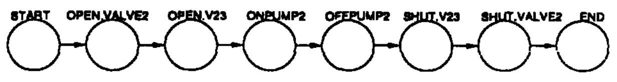

Figure 5: A plan for Example 2

This is only a structural description of what we consider a plan, the following explains in detail the semantics behind this conception of plan.

Actions have an interval of execution, but this interval is defined between every two consecutive actions of the same agent in the plan, thus it may change during the building process of the plan as actions are included in the plan. Let us consider the plan shown in Figure 6 as an intermediate step during the building process of the plan in Figure 5.

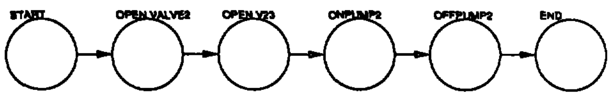

Figure 6: An intermediate plan

One may see that the action `Open.Valve2` will execute before `Open.V23`, but it will continue active until the end of the plan. Hence, its interval of execution is [`Open.Valve2`, `END`]. Now let us consider that the action `Shut.Valve2` is included in the plan obtaining the following one.

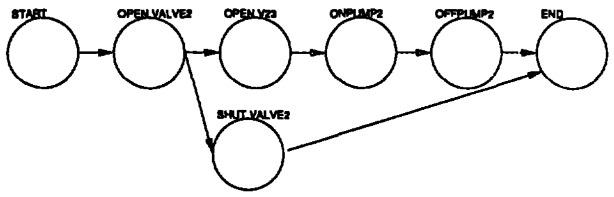

Figure 7: A step forward from Figure 6

This action produces a change of state in `Valve2` that affect the state reached by `Open.Valve2`, so the interval of execution for `Open.Valve2` is revised and redefined as [`Open.Valve2`, `Shut.Valve2`]. At once, the interval of execution of `Shut.Valve2` appears as [`Shut.Valve2`, `END`].

Although actions have an interval of execution, this interval is not previously defined by the own action but that it depends on the building process of a plan. This feature will be very important during the building process of the plan.

Concurrent execution of several actions is something natural in manufacturing systems and it is straightforwardly modelled in the plan. Actions whose intervals of execution overlap are concurrently executing. Let us consider the intermediate plan in Figure 6. Actions `Open.Valve2`, `Open.Valve23` and `TurnOn.Pump2` are concurrently executing.

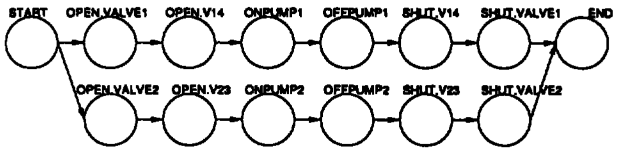

Figure 8: A partially ordered plan

Although the examples seen so far show a total order of actions, plans can have a partial order structure. A partial order is used not only to represent a class of total order plans, but also to represent possible concurrency. For example let consider again the manufacturing system shown in Figure 4. A plan to carry `ACID` from `TANK1` to `TANK4` and `WATER` from `TANK2` to `TANK3` could be the one shown in Figure 8. The fact that both branches of the plan are unordered means that there is no commitment between them; therefore the intervals of execution of actions in both branches could possibly overlap, that is, they could be possibly executing concurrently.

An immediate consequence of the possible concurrent execution of actions is that if the intervals of two actions can possibly overlap, then both actions should not interfere, that is, they must not have any opposite effect. This will be called an interference.

Causal links are also considered in the plan. They define intervals of protection for the literals that appear in the requirements lists of an action. Since there are three lists of requirements of different nature, the interval which defines a causal link can differ depending on the type of requirement. The causal link associated with a previous requirement is defined from the producer of the literal until the consumer (in terms of (Weld, 1994)). When the requirement is a simultaneous one, then the causal link must be defined during all the interval of execution of the action, that is, from the producer until the end of the interval of the consumer. Since the end of an interval of execution may change during the building process of a plan, causal links related to these requirements may also change. Later requirements have a different nature, they only need to be satisfied and they do not need to be protected throughout the plan like previous or simultaneous ones. Therefore, causal links with respect to later requirements are not considered.

An action threatens a causal link if the literal associated to the causal link appears in the deletions list of the action. Since causal links represent intervals of

```
MACHINE(Domain, Agenda, Plan, Links)
1. When Agenda is EMPTY Return SUCCESS
2. Task ← SelectTask(Agenda)
3. Choices ← HowToDoIt?(Task, Domain, Plan)
4. Iterate over Choices until it is empty
   (a) How ← ExtractFirst(Choices)
   (b) DoIt(How, Domain, Agenda, Plan, Links )
   (c) When MACHINE (Domain, Agenda, Plan, Links)
       Return SUCCESS
5. Return FAIL
```

Figure 9: The algorithm of MACHINE

protection for these literals, the interval of execution of an action which threatens a causal link and the interval of the causal link must never overlap. Neither interferences nor threats are allowed in a valid plan and they must be avoided by the usual methods of promotion and demotion.

The only notion of time in a plan like the ones in Figures 5 and 8 is the relative ordering between its actions, and this is only a qualitative notion like in (Allen, 1984) or (Sandewall and Ronnquist, 1986). The inclusion of a metric notion of time would be, of course, useful, however the main problem that appears in the building process of such a plan is the search for a correct interleaving of the actions and a qualitative notion of time is quite enough, although it is more conservative than a metric time, which would surely provide a more precise interleaving. This must also be taken into account to avoid considering this work like a scheduling approach since, in this paper, a solution to a problem is solely a correct interleaved plan.

Bearing in mind these conceptions of actions, problems and plans, the following section describes a nonlinear planning scheme, called MACHINE, designed by adapting the general nonlinear algorithm POP presented in (Weld, 1994) to this model of actions and plans, and is able to obtain the plans seen so far.

MACHINE: A Nonlinear Planner for the Building of Control Sequences

MACHINE is a generative refinement planning scheme (Weld, 1994) whose algorithm is described in Figure 9.

It uses four data structures to store the information during the planning process: an `Agenda`, the `Plan` and its `Links`, and the `Domain` in consideration. The `Domain` is the knowledge-based model of the manufacturing system, that is, the set of agents, their actions and the set of axioms. The `Plan` is a partially ordered set of nodes, where every node may be an instantiated action from the `Domain` or a subgoal, together with its

Links, that is, the set of the existing causal links, which describes the causal structure of the plan, the *plan rationale*. And the Agenda which is a set of tasks each of which describes a pending problem in the plan.

The start point is a null Plan with two dummy actions, START and END which encode the planning problem, and Agenda which initially contains only the pending subgoals specified in the goal of the planning problem and Links which is initially empty. Over this initial plan, a refinement process is applied which, at every step, solves a pending problem in the plan until there are no more pending problems or the problem cannot be solved. The different pending problems which may be found in a plan are pending subgoals (motivated by unsatisfied requirements) threats, interferences, and order inconsistency (motivated by a loop in the order structure, which must be a strict order). They all are included in Agenda, which drives the refinement process. The search process to solve the tasks in the Agenda is a basic depth first engine over the set of choices to solve every task.

Description of Modules

The three basic modules of MACHINE are shown in boldface. They are similar to the ones of POP (Weld, 1994), except that they have been adapted to work on the model of action and plans described in the previous section, thus only the most important differences will be described.

SelectTask. This module selects the first task in Agenda in order to solve it. Tasks in Agenda are ordered using the following scheme: first, order inconsistency, then interferences, threats and subgoals. Order inconsistency is the first one because it has no solution and it always leads to backtracking. Subgoals are the last ones because they are delayed until all the interferences and threats are solved. In addition to this, subgoals are also ordered amongst them by their relative ordering in such a way that subgoals closest to START are solved before the furthest ones.

HowToDoIt?. This module analyses a selected task from the Agenda and it builds a list with all the possible choices to solve it. An inconsistent order has no solution, so the list will be empty. The choices to solve interferences and threats are the known methods of promotion or demotion, nondeterministically. Subgoals may be solved by either the axioms, an existing action in the plan or a new action from the domain. Since this process is based on a most general unifying algorithm, the codesignation constraints defined on the variables of the agents will play an important role by rejecting undesirable unifications.

DoIt. This module applies one of the existing choices in the list built by **HowToDoIt?** to solve a problem, that is, it tries to solve the problem. Pending subgoals related to simultaneous and previous requirements are solved by producers which must be before the consumer action, and later requirements are solved by actions which must be after the consumer. The inclusion of a new action in Plan to solve a pending subgoal implies the following tasks. First, the inclusion of all of its requirements as pending subgoals in Agenda and the inclusion of the new causal link in Links (if an existing action were reused, the causal link were also included).

Second, when a new action is included in Plan, the end of its interval of execution is unknown, so, by default, it is assumed that its end is the dummy action END. However, the true end of all of the actions in the plan is continuously searched as shown in the previous sections. Since the end of the interval of an action implies a change of state in the agent which carries out the action, every time a requirement of change of state is solved by an action then the end of the interval of this action has been found, and it is updated in Plan.

And third, if new interferences or threats have appeared, then include it in Agenda also as pending tasks. Interferences and threats are found when there is harmful overlapping between the intervals of execution of actions and the intervals defined by a causal link as explained in the previous section. Promotions and demotions to solve interferences and threats are not applied between actions but between their intervals of execution. When an action is promoted over another action, it is promoted over all its interval of execution, not only the action. For example, let us consider the intermediate plan shown in Figure 7. If the action Shut.Valve2 would interfere the action TurnOn.Pump2, then its promotion will be over the interval of execution of TurnOn.Pump2 defined by [TurnOn.Pump2, TurnOff.Pump2], that is, Shut.Valve2 will be ordered after TurnOff.Pump2.

However MACHINE can delay the solution of some threats and interferences for a later moment in the resolution process. The reason is that, as mentioned before, not all of the intervals of the actions in Plan are known, some of them are known and some of them will be known as pending subgoals are solved. Therefore, if a threat or an interference is related to an undefined interval then it should be delayed until the end of the involved intervals are known. In order to do that, these kinds of threats and interferences are ordered in Agenda after pending subgoals giving them less priority. Later, if the solution of some of these subgoals finds the end of some of these problematic intervals, Plan will be up-

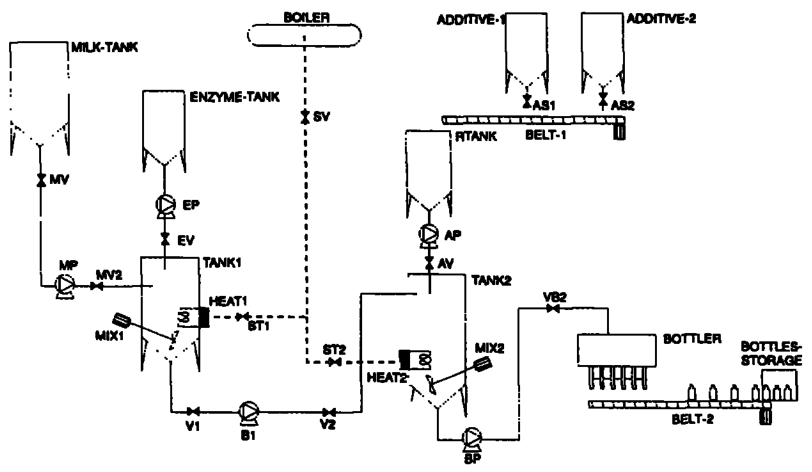

Figure 10: A real-size manufacturing system

dated and the threat of interference will be back at the beginning of **Agenda** and, so, solved appropriately. This is also a least commitment heuristic which could more or less say the following: *"I don't try to solve a problem if I don't know it exactly"*.

Experimental Results

MACHINE has been implemented in COMMON LISP and has been tested using the problems shown throughout this paper. It found the correct plan, i.e. control sequence, for all of them and its behavior is shown in Table 1.

Table 1: Some experimental results

Plan	Generated Nodes	Explored Nodes	Time	Size
Figure 2	56	39	11 s	12
Figure 5	38	30	5 s	8
Figure 8	69	51	16 s	14
Figure 10	217	144	374 s	40

This table also includes the result of the real-size problem shown in Figure 10. This problem consist in adding an ingredient (initially contained in ADDITIVE-1) to the milk initially contained in MILK-TANK and then proceed to bottle the mixture. Since there are many involved intermediate operations like carrying the milk and the ingredient, heat the mixture, etc, the final plan is too large as to be included here due to space limitations.

The final result of MACHINE is a control sequence, it is not exactly a control program but it has the necessary level of detail to be considered as such. Furthermore, in (Castillo et al., 1998) we show in detail how these control sequences may be translated into GRAFCET charts (Gruver and Boudreaux, 1993) and Petri nets (Peterson, 1981), as true representations for a control program, and very useful tools in the design and modelling of manufacturing systems.

Conclusions and Extensions

This work has presented MACHINE, a nonlinear planning scheme adapted from POP (Weld, 1994), motivated by the need to apply artificial intelligence planning techniques to the design of control sequences for manufacturing systems. The domain of manufacturing system has some basic properties which must be taken into account in order to ensure a correct reasoning about the actions which take place in such a domain. MACHINE deals with these features and it is able to obtain control sequences for manufacturing systems. However, it can only be considered as a step forward in the resolution of the problems which appear in manufacturing systems. The reason is that this is a very rich domain with many problems of different natures which should be taken into account by an autonomous problem solver, although the truth is that the core of that problem solver is actually a planning system and that all of these problems may be built like folders or extensions over this planning core. Some of these important problems, which will be dealt with in the near future, are the following ones.

Perhaps the most important problem is the inclusion of a metric time in order to both, quantify the intervals of actions, and allow for the representation of gradual achievement of effects along this interval. In real problems these intervals are not perfectly known and they are affected of some kind of vagueness. Time map managers (Dean and McDermott, 1987; Rutten and Hertzberz, 1993) seem to be very promising in this task.

A classic conjunctive goal is not expressive enough to represent the transformations needed for real manufactured products. It is necessary to define of goals which express a behaviour instead of a state, something as an *ordered* set of transformations on raw products, this is known as a *recipe*. At present, MACHINE does work with goals whose literals have a partial order structure.

In these domains, there are what could be called procedures: complex problems which can be decomposed into an ordered sequence of smaller subproblems. The planning system must know these procedures and it must also know how to work with them. This problem points directly to HTN techniques (Erol et al., 1994b; Erol et al., 1994a).

The control programs seen in this paper are intended to work in an open loop manner, that is, with no feedback from the environment. Real control programs have feedback from sensors in the environment and the planning system must be able to include the information supplied by these sensors in the planning process. This seems the most challenging problem because it implies both: (a) the ability of the planning system to adapt its behavior to the different ways in which sensors may appear (case-based and analogical techniques seem very promising in this task because they also seem to be the techniques used by humans in the same role) and (b) the ability to include some kind of conditional behavior in the plan because the information given by sensors is not always available at planning time.

References

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial intelligence*, 23:123–154.

Castillo, L., Fdez-Olivares, J., and González, A. (1998). An application of artificial intelligence techniques to the implementation and validation of control programs for manufacturing systems. Technical Report DECSAI-980111, University of Granada.

Dean, T. and McDermott, D. (1987). Temporal database management. *Artificial intelligence*, 32:1–55.

Erol, K., Hendler, J., and Nau, D. (1994a). UMCP: A sound and complete procedure for hierarchical task-network planning. In *AIPS-94*.

Erol, K., Hendler, J., and Nau, D. S. (1994b). HTN planning: complexity and expresivity. In *AAAI-94*, pages 1123–1128.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2:189–208.

Gil, Y. (1991). A specification of manufacturing processes for planning. Technical Report CMU-CS-91-179, Carnegie Mellon University.

Gruver, W. A. and Boudreaux, J. C. (1993). *Intelligent manufacturing: programming environments for CIM*. Springer-Verlag, London.

Klein, I., Lindskog, P., and Backstrom, C. (1993). Automatic creation of sequential control schemes in polynomial time. Technical Report LiTH-ISY-I-1430, Linkoping University.

McAllester, D. and Rosenblitt, D. (1991). Systematic nonlinear planning. In *AAAI-91*, pages 634–639.

Nau, D., Gupta, S. K., and Regli, W. C. (1995). AI planning versus manufacturing-operation planning: A case study. In *IJCAI-95*, pages 1670–1676.

Park, S. C., Gervasio, M. T., Shaw, M. J., and DeJong, G. F. (1993). Explanation-based learning for intelligent process planning. *IEEE Transactions on systems, man and cybernetics*, 23(6):1597–1616.

Pednault, E. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Knowledge Representation 1989*.

Penberthy, J. S. and Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *3rd. Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 103–114.

Peterson, J. L. (1981). *Petri nets theory and the modelling of systems*. Prentice-Hall.

Rutten, E. and Hertzberz, J. (1993). Temporal planner = nonlinear planner + time map manager. *Artificial intelligence communications*, 6:18–26.

Sandewall, E. and Ronnquist, R. (1986). A representation of action structures. In *AAAI-86*, pages 89–97.

Soutter, J. (1996). *An integrated architecture for operating procedure synthesis*. PhD thesis, Loughborough University.

Weld, D. (1994). An introduction to least commitment planning. *AI Magazine*, 15(4).