EAAI : 537 - PROD.TYPE: COM ED: C. MAMATHA
pp.1–16 (col.fig.: NIL) PAGN: SANDHYA HU SCAN: VENKI

PERGAMON

Engineering Applications of Artificial Intelligence 0 (2000) 1–16

Engineering Applications of ARTIFICIAL INTELLIGENCE

www.elsevier.com/locate/engappai

Planning plant operating procedures for chemical plant

R.S. Aylett[a,*], J. Soutter[b], G.J. Petley[a], P.W.H. Chung[c], D. Edwards[d]

[a] *Centre for Virtual Environments, University of Salford, Salford M5 4WT, UK*
[b] *BG Technology Centre, Loughborough, UK*
[c] *Department of Computer Science, Loughborough University, UK*
[d] *Department of Chemical Engineering, Loughborough University, UK*

Received 17 May 2000; received in revised form 8 December 2000; accepted 20 December 2000

Abstract

All industrial plants require an extensive set of operating procedures. This paper discusses the use of hierarchical nonlinear least-commitment AI planning technology to generate plant operating procedures for chemical process plant. It considers the handling of flow through the interfacing of a valve sequencing subplanner, the handling of safety through the mechanism of goals of prevention, and the use of *pairs* as a way of mutually constraining planning variables and increasing planning efficiency. It concludes with some results and discussion of the advantages of the approach. © 2001 Published by Elsevier Science Ltd.

Keywords: Plant operating procedures; AI planning; Chemical process plant

1. Introduction

All industrial plants require an extensive set of operating procedures which define the steps required—for example—to start the plant up, to shut the plant down, to isolate pieces of equipment for maintenance or to deal with emergency situations. Steps may be carried out manually by human operators, or some of them may be embodied in the plant control system, depending on the level of automation. It is clearly vital for reasons both of safety and efficiency that procedures are of a high quality.

In the chemical process industry, a multi-disciplinary commissioning team containing skilled engineers is normally responsible for defining sets of procedures, taking of the order of two man-years of effort. If operability problems are uncovered during this work, late changes to the design of the plant may result, sometimes while the plant is actually being constructed. These are the motivations for the development of computer-based tools to aid in the authoring of operating procedures. In the INT-OP project,[1] which ran between 1996 and 1999, state-of-the-art hierarchical non-linear partial-order AI Planning technology (Weld, 1994) was applied to the task of operating procedure synthesis (OPS) (Soutter, 1997), as far as we are aware, for the first time. The project produced the chemical engineering planner (CEP) as part of an overall system described below in Section 2.

Because of the importance of creating high-quality operating procedures, and the amount of manual effort currently required to do this, the problem has been studied by a number of workers in the field of chemical engineering. However, none of them were able to produce a general purpose system since the algorithms used were either oriented towards solving the problem of flow in a plant by organising the opening or shutting of valves, or were aimed at reasoning about the operation of reaction vessels such as filling a tank or starting a heater. The work of the early 1980s (Ivanov et al., 1980; Kinoshita et al., 1981) using state-graphs limited sample problems to plants containing a handful of valves because of the number of states they generated: 20 valves each with 2 states produces 1,048,576 nodes in a state graph. Other workers used larger plant (Rivas and Rudd, 1974) but only considered valves and not vessels. A real-world nuclear fuel processing plant was used in (Crooks and Macchietto, 1992), but this work concentrated on optimising a hand-generated plan. Only

*Corresponding author. Tel.: +44-161-295-2912; fax: +44-161-295-2925.
E-mail address: r.s.aylett@salford.ac.uk (R.S. Aylett).

[1] Funded by the UK Engineering and Physical Sciences Research Council: Academic Partners, University of Salford & Loughborough University; Industrial partners BG Plc, BP, ICI, Cogsys Ltd., TCCL.

PII: S0952-1976(01)00012-4

Aelion and Powers (Aelion and Powers, 1991) have seriously considered AI planning technology, and in this case a linear STRIPS type engine was used, dating back to the 1970s (Fikes et al., 1972), which was therefore unable to deal with unfavourable interactions between actions in the plan.

AI planning is a technology that has developed representations and algorithms specifically to handle combinatorial sequencing problems. Its approach matches the requirements of OPS very closely. A planning problem is usually defined by a domain model and by two states of that model: the initial state and the goal state. The domain model describes the objects in a domain, the actions that can be performed with the objects and the constraints on these actions. Actions are normally defined by *planning operators*. The initial state describes the state of the domain immediately before any actions have been carried out, with the goal state describing the facts which must be true after the plan has been completed. The output is a set of ordered steps (instantiated planning operators) which, if executed, take the domain model from its initial to its desired final state.

The task of producing a plan can be split into two closely related and intertwining subtasks. The first subtask involves correctly selecting and instantiating the planning operators needed to solve each goal in the final state. For example, consider three blocks A, B, and C all on a table; a single planning operator which allows a robot to move any block from the table onto another block if both blocks are clear; and a final goal of a tower with A on B on C. Of all the steps possible as the first in the plan, only that of moving B from the table onto C will meet the final goal state. All other planning operator instantiations, if chosen, require the planner to backtrack and undo them

The second subtask involves detecting and resolving conflicts between the steps needed to achieve different objectives. For example, in the above problem, if A is first put onto B in order to solve one of the end-goals, then it will not be possible to move B onto C to solve the other goal. Resolving conflicts can be carried out by reordering the conflicting actions, inserting new actions, or by replanning, as discussed in Chapman, (1987).

During planning, the search space can become enormous if no techniques are used to limit its size, and it is here that modern AI planning techniques have made substantial advances. Least-commitment planning (Penberthy and Weld, 1992) is an approach to reducing search spaces. It encompasses non-linear planning, in which only essential ordering constraints between actions are introduced, leaving all others unordered (in pseudo-parallel), allowing a whole set of plans to be represented at once. It also includes constraining the possible instantiations of an object used in the plan rather than committing to a particular instantiation. Hierarchical planning (Sacerdoti, 1974) also reduces the search space by representing a problem as a hierarchy of tasks that need to be achieved, allowing a plan or part of a plan to be represented by a high level of abstraction, with the lower levels, and more detailed part of the plan, left for later expansion.

AI planning, in its modern and relatively mature form, is thus a set of representations and algorithms specifically intended to deal with exactly the type of combinatorial sequencing problems that arise in OPS. Yet as the discussion above shows, it is a technology that has rarely been applied outside its own research community, though this is beginning to change with a number of successful real-world applications especially in Space (Aarup et al., 1992; Chien, 1994; Chien et al., 1997; Bernard et al., 1998). This paper discusses in detail what was required in order to apply AI planning technology to OPS in the hope that this will encourage other workers to apply it to similar industrial problems.

2. Planning characteristics of process plant and OPS

Fig. 1 shows an engineering line diagram (ELD) of the type that would be used by commissioning engineers generating operating procedures manually. The ELD presented is that for a real piece of plant—the double effect evaporator (DEE)—located in Loughborough University and used there for teaching purposes and also as a case-study in the INT-OP project. Though it is not therefore a real-world industrial plant, it is of equivalent complexity to a 'chunk' of real-world plant—that is a functional sub-division of a plant often considered by engineers in the field. Two such 'chunks'—the backend loop of an ammonia plant, and the metals extraction subsystem of an acetic acid plant—were also made available to the project as case studies by its industrial collaborators (ICI and BP, respectively) but industrial confidentiality prevents a detailed consideration of their topology and procedures.

One basic characteristic of the domain is revealed by the ELD. This is the high degree of interconnection (Aylett and Jones, 1996), obvious even if the particular symbols used are not familiar to the reader. This is not merely a static topological consideration. In a robot blocks world such as that discussed above, removing one block normally has no effect on the other blocks in the domain (as long as blocks are only taken from the top of piles). In a process plant, the significant effect of opening or shutting a valve is not that the state of the valve changes, but that, depending on the state of the rest of the plant at the time, one or more chemical flows may be started or stopped. The interconnectedness of the domain is reflected dynamically in the particular properties of flow.

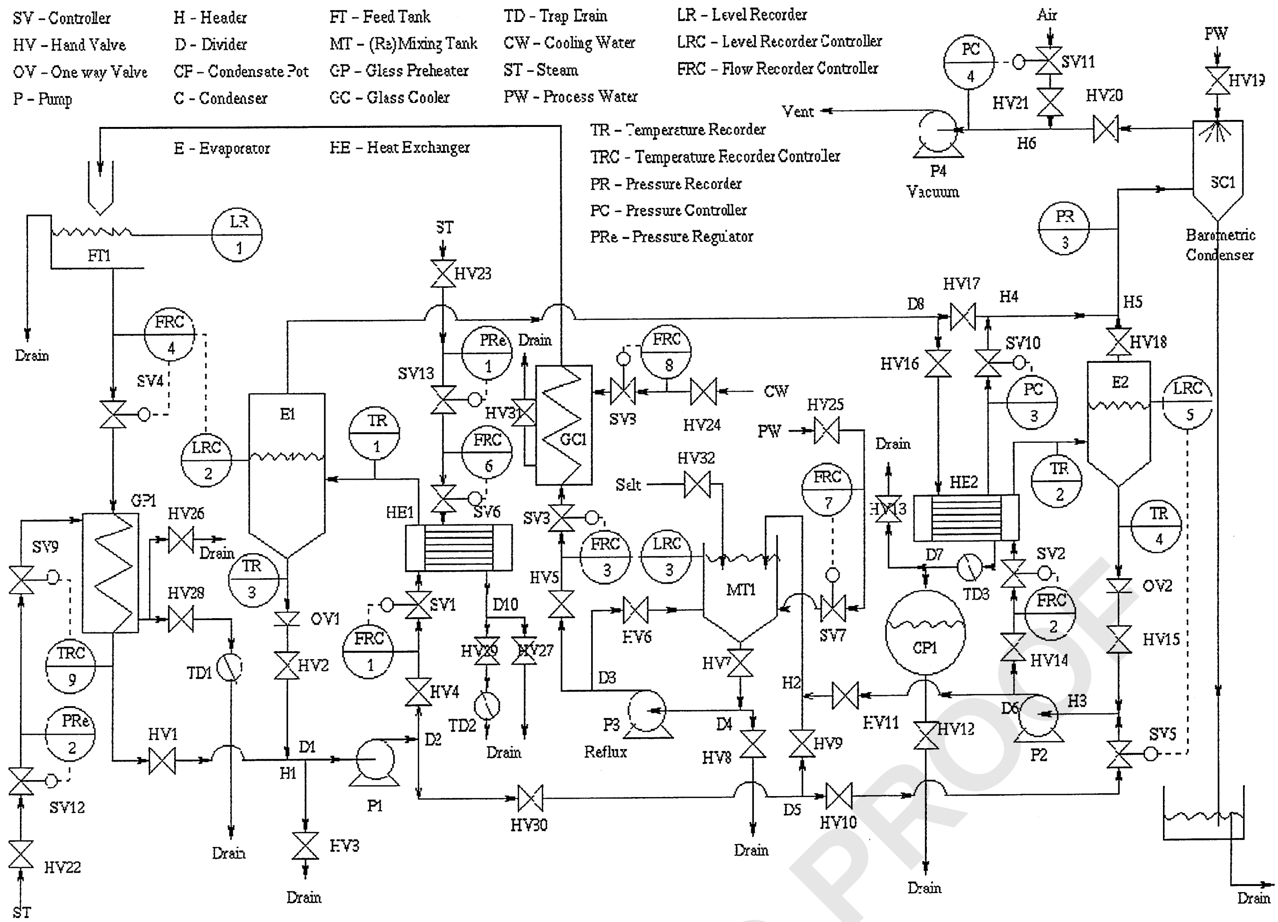

Fig. 1. The double-effect evaporator.

Flow plays as fundamental a role in a chemical plant domain as movement does in a robotic domain. Just as a robot delivering a letter to an office can be decomposed into a sequence of MOVE operations, so could the Haber process for making ammonia be decomposed into FLOW of chemicals. Does this mean that FLOW should be handled in much the same way as robot route planning?

In fact FLOW differs from robot route planning in significant ways. While a MOVE action can be thought of as directly moving the robot along a portion of its route, FLOW is produced entirely as a side-effect of valve (and pump) operations. While a robot moves *only* along the route planned for it, flow will occur at all junctions off the chosen route not specifically blocked. Thirdly, a flow continues once started until explicitly stopped. Fourthly, more than one chemical may pass down the same route simultaneously: in the Haber process hydrogen and nitrogen are merged into one flow. Finally, a flow route is contaminated with chemical even after the flow has stopped. These differences are expressions of interconnectedness and must be handled if planning is to be successful. While STRIPS operators are capable of handling side-effects, the approach taken in this project was to interface the planning mechanism to a specific subsystem for valve sequencing, using a specialised algorithm (Soutter and Chung, 1997; Aylett et al., 1998). This will be discussed and justified in Section 3.

A second characteristic of the process plant domain is the importance of safety. A plan which moves the plant to a desired end-state is unacceptable if—for example—along the way explosive gases have been mixed together, poisonous gases have been vented to the atmosphere or corrosive liquids have been flushed down a drain. In planning terms, intermediate states are important as well as end-states. The approach taken here was the introduction of 'Goals of prevention' (Soutter and Chung, 1996) defining safety restrictions as part of the overall description of the plant. These will be discussed in Section 4.

A third characteristic which is important from a planning point of view is the fact that flow delivers chemicals to components of the plant quite distant from where they originate as inputs. For example, in Fig. 1, at the top right corner the symbol PW shows process water entering the system which will eventually flow through the two heat exchangers, HE1 and HE2, in the centre portion of the ELD. It is important if the problem is to remain tractable that the planning operators used in operating these heat exchangers should not have to try every possible path in the plant in order to establish the source of this flow. At the same time, as we discuss below, it is also important that the specific topology of this plant is not encoded into the generic behaviour of a heat exchanger (Petley et al., 1998) described by these planning operators. Section 5 shows how this problem was tackled with an extension to the representation used known as *pairs*, with an associated modification of the planning algorithm (Aylett et al., 1999).

This paper therefore demonstrates that by taking into account the specific characteristics of the domain, very general mechanisms can be applied to OPS. Like all knowledge-based approaches, there are vital issues concerning the acquisition, validation and maintenance of the knowledge required. These issues are not discussed here (see Aylett et al., 1997), but lie behind the overall architecture of the whole system shown in Fig. 2. CEP-Tool is an intelligent front-end which handles knowledge acquisition for a particular plant, and delivers a domain model to CEP-Run, the core of which is CEP itself, the chemical engineering planner discussed in this paper.

CEP-Run encompasses a number of other functions not discussed in this paper. In particular, it includes some facilities allowing a user to interact with the planning process if they so require. It also includes a process known as linearisation, which turns the partially ordered plan net produced by the planner itself into a sequence of instructions forming an operating procedure. These have been discussed elsewhere (Aylett et al., 1997). At the heart of CEP-Run is the system shown

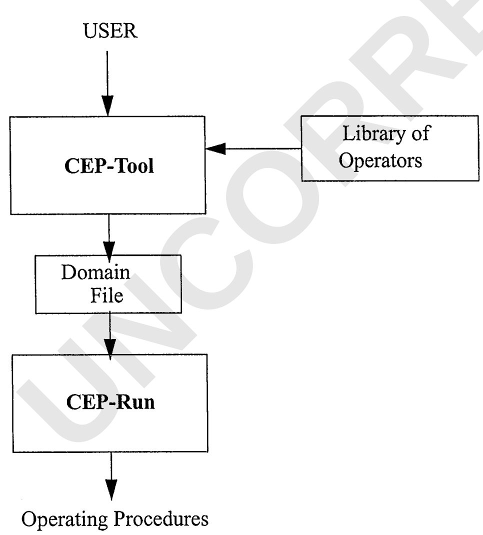

Fig. 2. The CEP architecture.

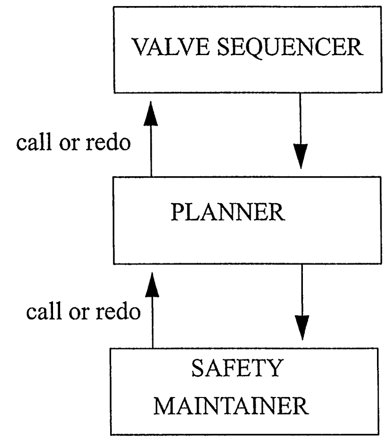

Fig. 3. CEP components.

```
operator OperateControlValve
{
    aperture ?state1;
    aperture ?state2;
    controller ?c;

    ?state1 != ?state2;

    achieve
      * aperture of ?c is ?state2;
    using
      aperture of ?c is ?state1;
    end
```

Fig. 4. CEP planning operator for operating a valve.

in Fig. 3, forming CEP, the chemical engineering planner, itself. It can be visualised, as in this figure, as a state-of-the-art hierarchical non-linear least commitment planner with specific facilities for handling flow and safety.

CEP works with planning operators of the basic form of the example shown in Fig. 4, the planning operator for toggling the state of a control valve. Here, all quantities prefixed with '?' represent variables, which can be instantiated with specific names in a particular plant. The achieve section shows what the effect of this action is when executed—to change the `aperture` of a valve to `?state 2`. The using section gives the preconditions for applying this action—in this case that the initial `aperture` is in `?state1`. Setting `?state1 != ?state2` constrains these states to be instantiated with different values. We have used this planning operator as an example because it is valve operations which produce flow—yet as we see in Fig. 4, the only direct result of operating a valve is to change its state from open to shut or vice-versa. How CEP can reason about flow and decide which valve operations to include in its plan is the subject of the next section.

3. Planning flow

It was argued above that flow—achieved by sequencing opening and closing valves—is fundamental to all continuous process plant domains. Previous OPS work in chemical engineering either dealt with valve sequencing alone (Foulkes et al., 1988), or planned the operation of reaction vessels to the exclusion of valve sequencing (Fusillo and Powers, 1987). CEP is the first OPS system to combine both.

Several approaches are possible. Flow may be handled as a post-processing step to the allocation of reaction vessels. Alternatively, it may be dealt with like other aspects of planning through planner operators. Thirdly, flow may be handled via a sub-planner integrated into the planner. CEP has adopted this third solution.

3.1. Flow as post-processing

This approach was adopted in Crooks and Macchietto (1992), who viewed OPS as a resource allocation rather than a planning problem. The objective was to produce given quantities of given products by choosing a process route (sequence of reactions) to produce each of the required products and then allocating resources in the form of reaction vessels to each. In a second phase, steps were added to the procedure to create the necessary flow paths between the reaction vessels used.

The problem from a planning perspective is that the pipes which carry a flow are themselves resources. If flow paths are chosen independently of each other then two flow paths may end up sharing the same pipes, mixing chemicals in dangerous or undesirable ways. Such flows need not overlap temporally: a flow can contaminate a pipe with a chemical which may have to be removed before a second flow can be created with another chemical. Removal of a chemical often involves washing a pipe out with some neutral substance like water and in order to get this to the required location in the plant, sometimes the substance will have to flow through a vessel. This causes problems because the system is designed to allocate vessels to tasks only in the first phase of procedure creation. Thus flow cannot be treated as a post-processing step.

3.2. Flow using operators

While flow could be modelled using planner operators, because of the characteristics discussed above this turns out to be clumsy and inefficient. While a `MOVE` operator with pre-conditions `at (?Robot, ?`X`)`, `next-to (?`X`, ?`Y`)` and effect `at (?Robot, ?`Y`)` can be used to find a route for a robot between two locations, an operator FLOW cannot take this form since not only must valves along the chosen flow-route be opened, valves off the flow-route must also be closed to stop flow into other parts of the plant. With no *explicit* representation of the valves in the flow-route, it is hard to close the correct valves.

However, modelling flow through planning operators OPEN and CLOSE for valves is also problematic. While in principle such operators can deal with side-effects, flow is a side-effect *which depends on the configuration of the plant*, so that the standard assumption that all effects of an action are declared in the planning operator is very hard to meet. The alternatives are: first, a separate planning operator to describe the operation of each valve in each interesting plant state. Second, enhance the planning operator representation to allow the effects of opening a valve to be a function of the state of the plant, for example using conditional effects. Third, use planning operators to represent the opening of each interesting flow route rather than representing each individual valve operation.

Each of these three strategies produces planning operators specific to a particular plant which then cannot be used in a different plant, a fundamental objection in a project trying to produce a system that can be configured to a new plant by a non-expert. They differ only in the trade-off between complexity and brevity. In the first and third cases each operator is simple but a huge number is required to describe a complex plant. In the second case, each operator is very complex but only one is required for each valve in a plant. In all cases, the valve sequencing problem must be solved anew for each new plant.

The same objection can be raised against a planning operator which handles all the flows in a section of plant. This operator would only work on that section of that plant—and due to its complexity and size would be extremely difficult to formulate and test. Nor is it clear that flow patterns would support such sectioning on any but the largest scale of granularity, with shared flows, circular flows and reversing flows giving particular problems.

In conclusion, it does not appear possible to create a planning operator model of opening a valve that is independent of the specific valve to be opened. Similarly, it does not appear possible to create a planning operator model to start a chemical flowing through a plant independent of the process plant that the chemical will flow though.

3.3. Flow with a subplanner

Noticeably all three of the AI planning systems currently used for real-world problems—OPLAN (Tate et al., 1994), SIPE (Wilkins, 1988), and PRODIGY (Veloso et al., 1995)—provide support for subplanners, suggesting that domain dependent algorithms of this type are not unusual. This section proposes a domain dependent but plant independent mechanism for creating a flow using a specific flow reasoning algorithm. The implication of this work is that some difficult problems are best solved through the development of domain dependent modules that live within otherwise domain independent planners.

The algorithm used by the OPS community to create a flow of chemical is based on Foulkes et al. (1988). It applies a maze searching algorithm; the valves around this route are closed and then the valves along the route are opened. Thus each flow of chemical has an easily determined effect—the contamination of all the units along the flow route by the chemical being transported. Clearly the subplanner should be called when the planner has a goal to create a flow between two points. It will find a flow route which must be communicated back to the planner in terms of valves to be opened and closed.

All of OPLAN, SIPE and PRODIGY approach subplanning as a mechanism for performing mathematical reasoning during planning. Hence it is not surprising that their subplanners are constrained to behave as mathematical functions, taking a fixed length sequence of arguments as input and producing a result completely determined by these input parameters. The input parameters are planning objects, that are variables or constants. A planning variable is used to hold the return value. But these interfaces cannot be adopted for a flow subplanner which differs in significant ways from a straightforward mathematical function:

1. Flow subplanners are non-deterministic. There may be many flow routes for a particular chemical and any suitable one may be chosen arbitrarily. All possible routes should be considered during backtracking to ensure completeness.
2. Flow subplanners may implement partial functions. There may be no feasible flow paths between two points or all feasible paths may be blocked by other flows of chemical. Hence it may not be possible to find a route for a particular flow of chemical at a particular point in a plan.
3. Unbound input parameters can be important. For example, if the destination point for a flow is unbound but constrained to the set of possible drains for a particular chemical, then the subplanner could opportunistically choose a suitable drain when looking for a flow route.
4. Flow subplanners return partial plans, not a single variable. The interface must support their incorporation into planning.

A principled way of creating fragments of plan can be created through the use of a macro—a piece of code which itself creates a piece of code when it runs. CEP had already included macro planning operators as a facility since this allows a user to specify sequencing

```
macro Flow{
  valve *?opened, *?closed;
  unit *?contaminated, ?source, ?destination;
call     /* subplanner call */
   flow(?source, ?destination)
     [*?opened, *?closed, *?contaminated];
solve    /* Use this operator for */
   flow(?source, ?destination, ?chem);
nodes    /* Node definitions */
   1 instant close;
order    /* Temporal ordering of nodes */
   1, @;
require  /* preconditions */
   1, @ aperture of *?closed is closed;
   @aperture of *?opened is open;
   @aperture of *?pumps is open;
achieve /* effects */
   1contains(*?unit, *?port, ?chem);}
```

Fig. 5. The flow macro planning operator.

information about the order in which goals should be met and is very useful where at some higher level of abstraction, 'chunks' of plant are to be started up one after the other.

Thus a specific CEP macro operator, `flow`, seen in Fig. 5 was used to implement the interface between the valve sequencer and CEP's general planning mechanism. One should note that this mechanism is general enough to support other subplanners of like complexity (for example, a computational geometry subplanner in the case of component assembly planning) and is therefore more powerful than the interfaces of SIPE, OPLAN and PRODIGY.

The `call` section of this macro specifies `flow` as the subplanner to be called from the CEP table relating names to available subplanners. The arguments in round brackets are input parameters: question marks show that they are variables rather than constraints. The arguments in square brackets are return parameters: the star in front of them indicates they represent a set of objects rather than a single object. One should remember here that the planning process is proceeding 'backwards', that is, from a goal requiring a flow to the actions needed to produce the flow.

The `call` above will find a route between `?source` and `?destination` and then constrain the variables `*?opened`, `*?closed` and `*?contaminated` with the details of the flow route found. For example, `*?opened` will be constrained to the set of valves that are to be opened. The call may as a side effect bind some of the input parameters—a particular flow must start from a particular source and end up at a particular destination.

The rest of the macro describes the handling of the variables constrained by the call. For example, at the start of the plan fragment being produced by the subplanner, and shown in Fig. 6, the valves around the flow path must be closed. This "close point" (Fig. 6) is referred to in the macro as `node1`. These valves must

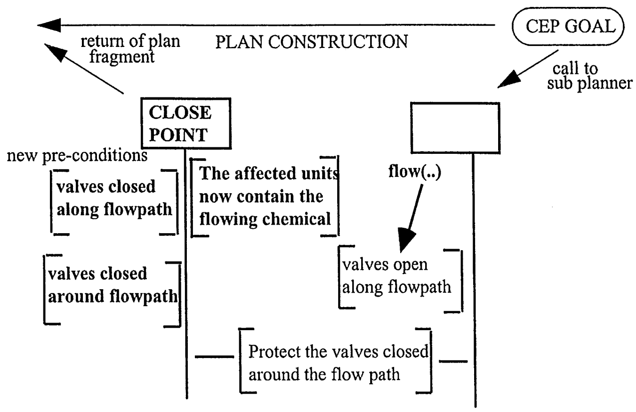

Fig. 6. Creating a plan fragment for CEP from the valve sequencer.

remain closed until the point in the CEP plan which produced the flow goal in the first place, which is given the special symbol @. CEP can say then that the flow operator has the precondition shown, amounting to "all the `*?closed` valves must be closed at `node1` and remain closed until `@`". Thus the macro now performs the translation of the subplanner output into a plan fragment.

The following steps are required within CEP in order to apply this macro:

(1) A new set of variables are created according to the type definitions at the macro head.
(2) These variables are constrained so the `solve` section matches the goal to be solved.
(3) The flow subplanner is called, further constraining the `opened`, `closed` and `contaminated` variables.
(4) The domain of all starred variables is fixed.
(5) New nodes are added as described in `nodes` and their order is constrained as in `order`.
(6) Preconditions, effects and causal links are added as described in `achieve` and `require`.
(7) On backtracking, steps 6, 5 and 4 are undone and the subplanner is asked for an alternative solution.

The use of this macro interface gives CEP a neat and generic interface for allowing a specialised subplanner to solve problems for which a standard AI planning algorithm is not the best solution and has successfully allowed CEP to cope correctly with all the problems posed by flows outlined above.

4. Dealing with safety through goals of prevention

CEP applies a method for generating safe plans by actively monitoring and protecting the safety of a plan during planning. Its approach may be described as an explicit strategy for planning with goals of prevention. The strategy is 'explicit' because the responsibility for plan safety is taken from the planner and given to a separate algorithm explicitly designed to maintain it.

A goal of prevention describes a set of states that must not occur during a plan. Expressed in predicate calculus, a goal of prevention has the general form shown in statement (1).

$$\forall(?v_1 \in \Re_1, \ \dots, ?v_n \in \Re_n, [\text{var constraints}].$$
$$\neg(p_1 \wedge p_2 \wedge \cdots \wedge p_n). \quad (1)$$

In statement (1), $?v_1 \dots ?v_n$ represent planning variables; $\Re_1 \dots \Re_n$ represent the sets of possible values for each of the variables; and $p_1 \dots p_n$ represent literals. The only variables in $p_1 \dots p_n$ are $?v_1 \dots ?v_n$.

For example, the goal of prevention shown in statement (2) represents the blocks world constraint "each block can have no more than one other block stacked on top of it". Note that $?b_1 \neq ?b_2$ should be read as "?b1 does not codesignate (that is, become instantiated to the same value) with ?b2".

$$\forall(?b_1, ?b_2, ?b_3 \in \text{Blocks}, ?b_1 \neq ?b_2). \neg(\text{on}(?b_1, ?b_3)$$
$$\wedge \ \text{on}(?b_2, ?b_3)). \quad (2)$$

We will say that a goal of prevention is violated if it becomes false during a plan. A plan is considered safe if every goal of prevention in the domain remains unviolated at every point in the plan.

4.1. Implicit and explicit strategies

The concept of goals of prevention is not new in the planning literature. Similar concepts include: 'hazardous conditions' in Rivas and Rudd (1974), 'preservation goals' in Schank and Abelson (1977), 'goals of maintenance' and 'goals of prevention' in Georgeff (1987), 'local constraints' and 'global constraints' in Fusillo and Powers (1987), and 'don't-disturb' goals in Weld and Etzioni (1994). However only two papers were found that discussed planning with such goals. On the one hand, Weld and Etzioni (1994) look at encoding goals of prevention into the planning operators of a domain so that the safety of the plan is maintained implicitly by the planner. On the other, the paper of Fusillo and Powers (1987) examines the explicit maintenance of goals of prevention during planning though only in the context of very simple state space planners much less powerful than CEP.

The implicit strategy used by Weld and Etzioni (1994) encodes the goals of prevention into the planning operator set of the domain. In a basic implementation of this strategy, whenever each new action is added to the plan, the action is made safe against all the goals of prevention by adding new preconditions and by constraining the variables within it. The planner must consider all possible ways of making each action safe. This can be done through backtracking or through the use of disjunctive preconditions.This is efficient and easy to implement but suffers from inflexibility. In this approach, the way that goals of prevention are handled is defined by the way that the planner works.

An explicit strategy is based on a safety maintenance algorithm that is run at regular intervals during planning. The method the safety maintenance algorithm uses to resolve a violation is similar to the algorithm used in the implicit strategy to make an action safe but has two major advantages which have led to its adoption in CEP.

Firstly, the planner is better able to explain a plan if the explicit strategy is used since the planner knows which possible violation each modification is added to protect against. The ability to explain a plan is important if a person and a computer are to cooperate to create a plan. It is of particular significance in a domain such as chemical process plant where the engineer using the software must always be able to justify the choices made, for example in the formal hazard assessment procedure known as HAZ-OP.

Secondly, the explicit strategy is flexible and can support a more sophisticated handling of goals of prevention. For some plant, goals of prevention must be constrained to a region of time and the planner must allow new goals of prevention to be added during planning. For example, consider a plant in which a chemical h needs to be continuously added to a reaction vessel during a reaction lasting 10 h. There are six routes that h can flow along from its supply tank to the reaction vessel. At all times during the reaction at least one route must be open to the flow of h. However, the route that is used during the first 5 h may be different from the route that is used in the last 5 h. This may happen if one of the pipes in the original flow route is needed in the achievement of some other objective of the planner. Protecting the flow of h during the reaction is the same as maintaining a goal of prevention which guards against closing all six flow routes during the reaction.

The explicit strategy provides access to the algorithm which detects a goal of prevention violation. It is possible to change this algorithm so that some goals of prevention are only noticed in certain periods of time during a plan. It is also possible to have the algorithm look for all the violations of a new goal of prevention rather than just the violations resulting from the newest action. CEP follows the approach taken by Fusillo and Powers (1987) but extends it to least commitment planning.

An explicit strategy for planning with goals of prevention has two parts: an algorithm to monitor the safety of a plan and detect violations of goals of prevention, and an algorithm to restore the safety of a plan after a goal of prevention violation is found. In CEP these two algorithms are run at the end of each planning cycle.

4.2. Detecting violations and repairing them

The safety of a plan can be evaluated incrementally by examining each action as it is added to the plan. This strategy was first suggested by Rivas and Rudd (1974). This evaluation is based on the idea of an action causing a goal of prevention violation. In CEP, a new action is said to *necessarily cause* the violation of a goal of prevention $\neg(p_1 \wedge p_2 \wedge \cdots \wedge p_n)$ if the new action necessarily achieves $p_j \in p_1 \ldots p_n$ at some point s and the goal of prevention is violated at s. This is not quite the intuitive idea of causation because, by the definition, an action can cause a violation that already existed before that action was added to the plan.

A new action is said to *possibly cause* a goal of prevention violation if there is some completion of the plan in which the action necessarily causes a goal of prevention violation. This definition is sufficient for our needs. If the initial state of a plan is safe and each action does not *possibly cause* a goal of prevention violation then the plan as a whole is safe for two reasons. Firstly, if no action causes a violation then the plan cannot contain a violation to which any action contributes. Secondly, part of the STRIPS assumption is that the world state will not change except as the result of an action. If a goal of prevention is violated by a plan then at least one of the actions must contribute to this violation.

`DetectViolation()` (Fig. 7) is the procedure in CEP to find whether a new action *possibly causes* a goal of prevention violation. The heart of this procedure is a routine to examine a point s in a partial plan and decide whether a goal of prevention is violated at that point. This routine can be implemented using a simplified planner which cannot add new actions to the plan. The simplified planner is given the task of achieving all the terms in the goal of prevention at the point s. Each term is treated as an end goal. We reason that the simplified planner can achieve its end goals if and only if the goal of prevention is violated at s in some completion of the plan.

At worst the simplified planner will have to consider every completion of a partial plan in order to achieve the goals that it has. In CEP each partial plan has a finite number of completions because all planning variables are constrained to sets of possible values and because a plan can be ordered only in a finite number of ways. Hence the simplified planner has a finite search space. The search space does not contain loops, mainly because solving a goal does not create new subgoals, and so planning is deterministic.

For this implementation a set of sensible values for s was needed. The possible values were limited to the set of actions achieving literals in the goal of prevention. The reasoning is that one of these achievers must come last, or at least no latter than any other achiever. At this

1. c = choose a condition from the set of conditions achieved by new-action.
2. p = choose a goal of prevention from the domain description.
3. p_j = choose a term from p such that p_j possibly codesignates with c.
4. Create a new set of plan variables to represent the variables in p.
5. Constrain the plan variables so that p_j codesignates with c.
6. p_z = choose some term from the goal of prevention p.
7. If $z \neq j$ then s = choose some action in plan such that possibly s achieves p_z and possibly s comes before new-action.
8. If $z = j$ then s = new-action.
9. Constrain the plan variables so that s necessarily achieves p_z.
10. Order new-action at or before S.
11. Add a causal link to protect c between new-action and S. Resolve any conflicts with this new causal link.
12. For each term p_i in p such that $i \neq j$ and $i \neq z$:
   (a) X = choose an action in plan which possibly achieves p_i and which is possibly before s.
   (b) a = choose an effect of X which possibly codesignates with p_i.
   (c) Constrain the plan variables so that a codesignates with p_i.
   (d) Order X at or before s.
   (e) Add a causal link to protect p_i between X and s. Resolve any conflicts with this new causal link.
13. Remove all the constraints and causal links that have been added during DetectViolation(). This will restore the original plan.
   Return that a violation was found.

Fig. 7. The algorithm DetectViolation ( ).

latest point, all terms in the goal of prevention must be true and so this point is a candidate value for s. It is not clear how to predict which achievers will come last and so each achiever must be tried as a possible value of s.

The implementation also required a strategy for handling the variables in the goal of prevention. Any such variables will be implicitly universally quantified (a goal of prevention holds for any binding of its variables). CEP represents the variables in a goal of prevention by creating a set of plan variables to associate with the goal of prevention variables. The simplified planner is allowed to constrain these new variables as normal. In effect, the simplified planner is directed to find the violation of any instance of a goal of prevention.

When a goal of prevention violation is found in a plan, the planner must either resolve the violation or backtrack. The plan cannot ignore the violation and produce an unsafe plan. For completeness, all non-redundant methods of resolving each violation must be considered. A goal of prevention violation occurs if for each term in the goal of prevention there is some action which asserts that term (the *achiever* of the term) and no action which possibly denies (or *clobbers*) the term between the point at which it is achieved and the point s where the goal of prevention is violated. A goal of prevention violation is said to have been resolved if the DetectViolation( ) algorithm will not signal the same violation again. Intuitively, two violations are the same if they involve the same achievers and the same goal of prevention.

There are exactly two ways that a goal of prevention violation may be resolved. First, one of the literals in the goal of prevention can be denied between the point it is achieved and s. This will prevent the success of step 11 or step 12e in the DetectViolation( ) algorithm. Second, variables can be constrained to prevent some achiever from matching the proper term in the goal of prevention, and thus prevent the success of step 3 or step 7 or step 12b in the algorithm. No other steps in the algorithm can be prevented from succeeding without erasing part of the current plan structure.

5. Using pairs to reduce planner search space

Having looked at how flow and valve sequencing is handled in Section 3, and the use of goals of prevention to deal with safety in Section 4, we finally consider how CEP can represent knowledge about where flows originate without tying its domain representation too tightly to the topology of individual plants. First *pairs* are defined in terms of how CEP functions, and then their use as a knowledge representation in process pant is considered.

5.1. What pairs are

Pairs can be seen as a straightforward but useful extension to the expressiveness of the STRIPS formalism still widely used in planning. Recent work (Long and Fox, 1998) distinguishes between three types of extensions to STRIPS: purely syntactic extensions, which make domain description easier but do not change its scope; extensions which allow new domain properties to be represented (Erol, 1995); and extensions which allow extra domain-specific knowledge to be represented so as to reduce the search problem for the planner in some way. Pairs are an example of this third type of extension, which, like the second type, requires associated changes to the planning algorithms used.

The earliest planners, including STRIPS, represent planning variables as a receptacle that can hold a value or another variable or nothing. A useful aspect of STRIPS variables is that they support *co-designation*. A goal `on(?x-5,table)` can be solved by a goal achieving `on(?z-6,table)` without binding `?x-5` or `?z-6` to particular values. This is important if the solving action has preconditions which also involve `?z-6` because the planner is still free to solve the precondition for any value of `?z-6`.

Later planners also allowed *non-codesignation* constraints between variables and values. Examples are Chapman's TWEAK (Chapman, 1987) and Weld's POP (Weld, 1994). Non-codesignation constraints occur when the planner seeks to prevent an action from *clobbering* (that is, negating) the achievement of a goal. For example, to prevent an action with the effect `on(?x-3,blue-block)` from clobbering the achiever of `on(?z-6,table)` the planner must ensure that `?z-6` and `?x-3` are separate objects by applying a non-codesignation constraint. This is the second method referred to above for handling the violation of a goal of prevention.

As an extension of this idea, some planners allow variables to be constrained to a set of possible values. Inconsistencies can then be discovered more quickly because the planner can watch for variables that have exhausted their set of possible values. In a finite domain, associating a variable with a set of possible values is the same as giving that variable a type. For example, associating `?p` with the type "person" is the same as restricting `?p` to the finite set of people that are known to the planner.

However variables may also be constrained by their relationship with other variables. For example, if a person `?p` is to perform a job `?j` then we could require `?p` to be qualified to perform `?j`. Often these relationships are represented using predicates, for example `qualified_for(Mary,Java)`. A relationship between a set of variables may be *static* or *dynamic* in a given domain. Dynamic relationships can change over the length of a plan—it might be possible for John to qualify for software testing—while static relationships are not affected by any of the operators in the domain and so cannot change over the plan. For example, it may be the case that none of the operators in the domain cause a person to become qualified or disqualified. In this case we can describe them as *invariants* (McLuskey and Porteous, 1997).

Planning theory has been developed with dynamic predicates in mind. Most planners use a complex modal truth criterion just so they can reason about the state of dynamic relationships in a changing plan. Such a criterion is not needed to reason about static relationships, which can be handled as a special case, yet most planners treat static relationships in the same way that they treat dynamic relationships in spite of the inefficiency this introduces. An exception is the use of typing just mentioned, as a type can be seen as a static relationship with an arity of one. *Pairs* represent a generalisation of typing so that all static relationships are treated as variable constraints, with a resulting gain in efficiency that turns out to be very important in a heavily interconnected domain such as a process plant.

CEP's implementation is a simple one. It assumes that all the variables in a relationship are unbounded. If Mary is the only Java programmer then it is possible to reason that non-codesignating `?p` with `Mary` also non-codesignates `?j` with `Java`. However it is more common to reason that binding `?p` to `John`, or any other value except `Mary`, non-codesignates `?j` with `Java`.

Thus *pairs* represent the possible values that related variables can take. For example, the pair `<Mary,Java>` represents that Mary is one of the people that can program in Java. One can also represent negative knowledge—for example, in a negative relationship, `<John,Drive>` would represent that John is one of the few people who cannot drive. For simplicity, CEP assumes that most relationships between variables are binary—hence the term *pairs*. So, while CEP can reason about a person being qualified for a job it cannot directly reason about the safety of storing a chemical at a particular temperature in a particular container. However, separate pairs linking the storage of the chemical and the temperature and the temperature and the container can be set up.

In the data structure for a variable, a record is kept of the pairing constraints which apply to that variable. The record includes the list of pairs in the relationship, whether this relationship is positive or negative and which side of the pair should match the variable. Pairing constraints, like typing constraints, are applied when a variable is created. When a variable is bound to a value, the pairs on that variable are simply translated into non-codesignation constraints. In the negative relationship above, if ?*p* binds to John then ?*j* is non-codesignated with `Drive` and all other jobs mentioned against John. In the positive relationship above, if ?*p* binds to `Mary` then ?*j* is non-codesignated with all jobs except `Java` and the other jobs mentioned against `Mary`. When two variables co-designate, the list of pairing constraints is simply unified. Pairing constraints are simply ignored when non-codesignation constraints are applied.

5.2. Using pairs to represent plant knowledge

From a declarative perspective, a chemical plant consists of a (usually very large) set of components and connections. Each component can be represented as a frame in an equipment hierarchy, while the topology is represented by specifying for each component to which other components it is immediately connected. It should be noted that components are instances of generic pieces of equipment such as valves, pumps or vessels, but topology is specific to a particular plant. Knowledge of both components and topology is easy to acquire through the same CAD system which is used to design the plant (Aylett et al., 1997).

It is much more difficult to acquire the procedural knowledge for the domain, the behaviour of the plant, which is embodied in planning operators such as that seen in Fig. 4. Planning operators not only have to accurately model the behaviour of plant components, but also take into account the way in which the planning system itself functions. This dual role can make them hard to define successfully even for experts in a particular planning system. In the INT-OP project the assumption was made that the majority of the planning operators required for a particular plant related to generic components found as instances in the plant. Thus all valves or pumps of a particular type in the equipment hierarchy can be modelled in the same way. This may be summarised as the position 'function is independent of structure'.

It then becomes possible to develop a library of generic planning operators attached to the equipment hierarchy. The appropriate planning operators can simply be loaded for a domain containing the components with which they are associated without the constructor of the domain needing to understand their internal structure, thus making the system accessible to chemical engineers rather than to AI experts. The library currently contains 36 such generic operators (Petley et al., 1998).

However in AI planning, there is always a risk that making planning operators more general will multiply the search effort required to instantiate them for a particular domain. In particular, every use of a planning operator which is concerned with a flow of chemicals, and therefore invokes the valve-sequencing component discussed above, must instantiate variables for the start and end points of the required flow. For example, the vessels known as *formers* in an ammonia plant require a supply of natural gas, which comes into the plant from an external source and flows through a number of pipes and valves to reach these vessels. A planning operator to start up a former in an ammonia plant must establish this flow of natural gas.

Back-tracking across chosen start or end-points for a flow is particularly expensive because each time the valve-sequencer must be called to establish the new route. The interconnectedness of the plant means that there are always, in the absence of knowledge of the specific topology of the plant, many such choices. On the other hand, if knowledge of the specific topology of the plant is incorporated into planning operators, they cease to be generic. Thus a mechanism is required which allows knowledge dependent on topology to be represented in generic planning operators. Pairs offer just such a mechanism.

In order to represent topological knowledge inside planning operators, generic pairs are declared. These are then matched to plant-specific pairs declared in a domain file as a part of the description of a particular plant. Fig. 8 gives an example of the use of generic pairs (in bold) in a CEP planning operator. The first pair in Fig. 8 means that when the operator instantiates the variables ?pilot and ?source the only possibilities will be the values used in the definitions for the plant-specific pair unitSource, an example of which is given in Fig. 9. In the operator of Fig. 8, a flow of fuel is required from a source into the pilot, and without the pair feature CEP would have to try all the possible sources during planning until one was found that was suitable. However, by using pairs, only the sources defined in a unitSource pair would be tried, and for the operator in Fig. 8 only unitSource pairs whose first part was a piece of equipment of type pilot.

In order to demonstrate the efficiency gains of the pairs mechanism, a small experiment was conducted using the DEE domain shown in Fig. 1. In summary, the plant takes an input of brine and extracts the salt by means of evaporation, with steam generated in the first evaporation process used to drive a second. The top-level operator for this domain, a macro operator which generates the procedure to start the whole plant, was implemented in three versions: firstly as an operator including specific plant knowledge (Fig. 10); secondly as a generic operator without the use of pairs (Fig. 11); and finally as a generic operator with pairs (Fig. 12).

Note the differences between these three versions: in the first, specific plant components are referenced—thus HE1 is the particular heat exchanger in this plant. In the second version, specific references to actual plant

```
macro SupplyFueltoPilot
{ inlet ?source;   vent ?vent;   pilot ?pilot; chemical ?fuel;

  pair unitSource, ?pilot : ?source;
  pair chemSupply, ?source : ?fuel;

  solve
    supplyFuel(?fuel, ?pilot) is true;
  nodes
    1 instant flowOfChemical;
    2 instant pressureControl;
    3 instant stopVenting;
  order
    1, 2, 3, $;
  require
    1, 2 flow(?source, out, ?vent, in, ?fuel, fill);
    2, $ > flow(?source, out, ?pilot, in, ?fuel, fill);
    3, $ noFlowUpstream(?fuel, ?vent, in);
  end }
```

Fig. 8. Generic pairs in a planning operator.

```
pair (unitSource, PI1 : Input4).


pair (chemSupply, Input4 : NaturalGas).
```

Fig. 9. Pairs declared for a specific plant.

components have been replaced by generic variables for the classes of components found in a plant of this type. These are now declared using types in the macro header (differences and additions in bold). In the third version, the first two pairs establish, together with the inequalities above them, that two different heaters are required, one for each evaporator. Pair 2 then identifies a catchpot as linked to the first heater and a condenser as linked to the second heater.

In order to even out variations on the Sun Sparc5 being used to run this domain, the problem was run 10 times for each version of the planning operator. The results can be seen in Table 1. It can be seen that a high penalty of an over 35% increase in planning time is paid for the generic operator without pairs. This is striking given that 20 planning operators are needed in all for this problem and the final plan contains 236 steps. However, once the four generic pairs are included in the operator, the increase in planning time is cut to less than 1%. The use of pairs is thus shown to make a substantial contribution to planning efficiency. In other plant domains, the use of pairs has allowed the planning system to find a solution where previously it failed.

```
macro StartDoubleEvaporator
{   chemical ?chem;

    solve
        startDouble(E2) is true;
    nodes
        1 instant createChemical;
        2 instant activateEvaporator1;
        3 instant heatPlant;
        4 instant activateSprayCondensor;
        5 instant activateEvaporator2;
        6 instant activateHeater2;
    require
        1, $ create(MT1, ?chem, ls) is true;
        2, $ > active(E1) is true;
        3, $ > active(HE1) is true;
        4, $ active(SC1) is true;
        5, $ active(E2) is true;
        6, $ active(HE2) is true;
        6, $ active(CP1) is true;
    order
        1,2,3,4,5,6,$;
    achieve
        $, @ startDouble(E2) is true;
    end }
```

Fig. 10. A plant specific planning operator.

6. Results and conclusions

Table 2 shows some comparative times for the generation of operating procedures in some of the case studies carried out. It can be seen that these times are quite acceptable for interactive use. The metals extraction system is the one case study in which planning was relatively slow—this reflected a temporal complexity of flow in which a particular route supported a flow in one direction followed by a later flow in the opposite direction. The flow interface had not been designed with this in mind and it is likely that further work to extend it would reduce the planning effort required.

Appendix A gives an example of the output of the system—the plant operating procedure for starting up

```
macro StartDoubleEvaporator
{   mixingTank ?mixer;          tubularTempChanger ?temp1, ?temp2;
    glassTubularTempChanger ?glassTubularTempChanger;
    evaporator ?evap1, ?evap2;  condensor ?condensor;
    catchpot ?catchpot;         chemical ?chem;

    ?evap1 != ?evap2;?temp1 != ?temp2;
    ?temp2 != ?glassTubularTempChanger;

    solve
         startDouble(?evap2) is true;
    nodes
         1 instant createChemical;
         2 instant activateEvaporator1;
         3 instant heatPlant;
         4 instant activateSprayCondensor;
         5 instant activateEvaporator2;
         6 instant activateHeater2;
    require
         1, $ create(?mixer, ?chem, ls) is true;
         2, $ > active(?evap1) is true;
         3, $ > active(?temp1) is true;
         4, $ active(?condensor) is true;
         5, $ active(?evap2) is true;
         6, $ active(?temp2) is true;
         6, $ active(?catchpot) is true;
    order
         1,2,3,4,5,6,$;
    achieve
         $, @ startDouble(?evap2) is true;
    end }
```

Fig. 11. A generic planning operator without pairsI.

the DEE plant. This procedure was checked by an expert in the operation of the plant and found to be correct, though initially not quite the same as the one a human would have produced (Aylett et al., 1997). The aim of the project was to produce correct procedures in a reasonable time, and this has been met—however issues of clarity, justifying each step and spatial influences on the ordering of steps—all of which are important to end-users—would require further work.

This paper has argued that AI planning technology is an appropriate choice for the combinatorial sequencing problems involved in generating plant operating procedures because of its specialist algorithms and representations. AI planning provides generic mechanisms which allow it to be applied to this problem in a variety of process plant—for example the DEE shown in Fig. 2, as well as the back-end loop of an ammonia plant and the metals extraction unit of an acetic acid plant. By providing the appropriate domain knowledge, CEP could be applied to any other continuous process plant (for example, the making of cottage cheese). With suitable extension it could also be applied to batch plant.

The system does not aim at optimisation of the resulting plant operating procedures, but this could be carried out as a back-end operation of the type discussed in Crooks and Macchietto (1992). It is not designed to perform calculations nor to reason explicitly about time—sequencing handles time implicitly (in terms of *before*, *after* and *unordered*) but does not allocate time to processes. This has not been found to be a problem in the plant considered and may be more properly seen as a process design consideration than one that is central to operability.

While CEP is based on an entirely generic approach to planning (essentially the STRIPS formalism mentioned in a number of places above) it has nevertheless incorporated mechanisms specifically aimed at dealing with process plant, though in each case it has done this in a very general way. Thus the valve sequencing component discussed in Section 3 is specific to the routing of chemical flows, but the macro interface used to implement it could be used to incorporate any other subplanner that returned a fragment of plan. The handling of safety is a major preoccupation of this particular domain, but the goals of prevention discussed

```
macro StartDoubleEvaporator
{   mixingTank ?mixer;          tubularTempChanger ?temp1, ?temp2;
    glassTubularTempChanger ?glassTubularTempChanger;
    evaporator ?evap1, ?evap2;  condensor ?condensor;
    catchpot ?catchpot;         chemical ?chem;

    ?evap1 != ?evap2; ?temp1 != ?temp2;
    ?temp2 != ?glassTubularTempChanger;

    pair evaporatorHeater, ?evap1 : ?temp1;
    pair evaporatorHeater, ?evap2 : ?temp2;
    pair heaterCatchpot, ?temp2 : ?catchpot;
    pair evaporatorCondensor, ?evap2 : ?condensor;

    solve
        startDouble(?evap2) is true;
    nodes
        1 instant createChemical;
        2 instant activateEvaporator1;
        3 instant heatPlant;
        4 instant activateSprayCondensor;
        5 instant activateEvaporator2;
        6 instant activateHeater2;
    require
        1, $ create(?mixer, ?chem, ls) is true;
        2, $ > active(?evap1) is true;
        3, $ > active(?temp1) is true;
        4, $ active(?condensor) is true;
        5, $ active(?evap2) is true;
        6, $ active(?temp2) is true;
        6, $ active(?catchpot) is true;
    order
        1,2,3,4,5,6,$;
    achieve
        $, @ startDouble(?evap2) is true;
    end }
```

Fig. 12. A generic planning operator with pairs.

Table 1
Results of using pairs

Operator	Planning time (s)
Specific	20.2
Generic, no pairs	27.4
Generic plus pairs	20.35

Table 2
Generated procedures

Plant	Proc. steps	Time taken (s)
Cottage cheese	20	<1
Pulsed column rig	17	<1
Double-effect evaporator	238	4
Ammonia back-end loop	168	3
Acetic acid metals extraction	351	348

in Section 4 are also a very general mechanism which can be used in any domain for which there are forbidden intermediate states. Finally, the pairs mechanism of Section 5 is of particular use in allowing generic planning operators to take account of the topology of particular plants, but is a general facility for mutual constraints between variables which could be applied to quite different domains.

In 1996, when the work reported here began, the most current techniques in use in those AI planning systems which were being applied to real-world problems were used as a starting point. In the following years, new algorithms have been developed by AI planning theoreticians looking for ways of producing more efficient planners. The GraphPlan algorithm (Blum and Fust, 1997) is the best known of these developments and has resulted in a great deal of subsequent work, for example Kambhampati et al. (1997), Weld et al. (1998), Cayrol et al. (2000) to name only a few. If one were to start in the year 2000, would this approach be better as a way of generating plant operating procedures than the ones discussed above?

One can only answer such a question provisionally, since future work might meet some of the objections, but as things stand the answer is a definite no. The first reason for saying so is that it is clear that this approach as of now does not scale up to a problem the size of a real chemical plant. GraphPlan in its pure form builds a propositional version of the domain before planning, only feasible if the number of schemas and constants is small. Currently, although some initial work has been carried out, there is no hierarchical version of Graph-Plan, while hierarchy is fundamental to the way plant operating procedures are constructed in the real world as well as to the way designers conceive of process plants.

Indeed, at this point, nobody to our knowledge has applied a GraphPlan planner to a real-world problem, while in 1996 the existence of substantial applications of planners like OPLAN, SIPE and PRODIGY was an existence proof that these technologies could cope with the necessary scale.

A second set of problems is associated with how a user would interact with a GraphPlan planner. The use of planning operators in CEP allows the user to understand what subtask the planner is working on as well as the causal links involved. The propositional decomposition needed for the GraphPlan algorithm is much harder to interact with. Thus the technology lacks the necessary maturity at this point to be a suitable candidate for the domain so successfully tackled by CEP.

In conclusion, it appears that the combination of mature AI planning mechanisms with appropriate extensions has allowed the INT-OP project to solve the OPS problem for real-world plant in a way that was not possible for previous systems. It is hoped that this will pave the way for further applications of the technology—as for example recent work in developing control sequences at a lower level (Castillo et al., 1999). If this paper causes other groups working in engineering applications of AI to consider AI planning as a possible approach, then it will have met its objective.

Appendix A Operating procedure generated by CEP to start-up the DEE (part)

1. Set controller LRC_5 and turn on.(46)
2. Set controller FRC_6 and turn on.(45)
3. Set controller LRC_7 and turn on.(44)
4. Set controller FRC_8 and turn on.(43)
5. Set controller TRC_9 and turn on.(42)
6. Open valve HV7.(185)
7. Open valve HV25.(232)
8. Achieved: Flow route from Input2 to MT1 for processWater.(228)
9. Open valve HV6.(211)
10. Turn on pump P3.(210)
11. Achieved: Flow route from MT1 to MT1 for brine.(206)
12. Mixing in MT1.(194)
13. Open valve HV32.(202)
14. Achieved: Flow route from Input8 to MT1 for salt.(198)
15. Close valve HV32.(203)
16. Achieved: Stopped flow of salt to MT1.(197)
17. Warning: Make sure that all the salt needed has entered system before stoppingflow.(192)
18. Mixing brine in MT1.(190)
19. Open valve HV16.(59)
20. Open valve HV20.(60)
21. Achieved: Flow route from E1 to Output10 for steam.(55)
22. Open valve HV4.(183)
23. Open valve HV5.(182)
24. Turn on pump P3.(180)
25. Turn on pump P1.(181)
26. Achieved: Flow route from MT1 to E1 for brine.(176)
27. Open valve HV11.(187)
28. Open valve HV2.(189)
29. Open valve HV10.(188)
30. Turn on pump P2.(186)
31. Achieved: Flow route from E1 to MT1 for brine.(172)
32. Open valve HV26.(104)
33. Achieved: Flow route from GP1 to Output100 for steam.(100)
34. Open valve HV31.(111)
35. Open valve HV24.(112)
36. Achieved: Flow route from Input3 to Output4 for coolingWater.(107)
37. Open valve HV22.(90)
38. Achieved: Flow route from Input5 to GP1 for steam.(86)
39. Open valve HV28.(97)
40. Achieved: Flow route from Input5 to TD1 for steam.(93)
41. Wait until air flushed out of GP1.(77)
42. Close valve HV26.(146)
43. Achieved: Flow route for steam (heat source) through GP1 established.(76)
44. Achieved: Flow route from TD2 to Output8 for steam.(161)

References

Aarup, M., Arentoft, M.M., Parrod, Y., Stokes, I., Vadon, H., Stader, J., 1992. Optimum-AIV: A knowledge-based planning and scheduling system for spacecraft AI. In: Zweben, M., Fox, M.S. (Eds.), Intelligent Scheduling. Morgan Kaufmann, Altos, CA, pp. 451–470.

Aelion, V., Powers, G.J., 1991. A unified strategy for the retrofit synthesis of flowsheet structures for attaining or improving operating procedures. Computers and Chemical Engineering 15 (5), 349–360.

Aylett, R.S., Jones, S.D., 1996. Planner and domain: domain configuration for a task planner. International Journal of Expert Systems 9 (2), 279–318.

Aylett, R.S., Petley, G.J., Chung, P.W.H., Soutter, J., Rushton, A., 1997. Planning and chemical plant operating procedure synthesis: a case study. In: Steel, S., Alami, R. (Eds.), Recent Advances in AI Planning Planning. Springer, Berlin, pp. 39–51.

Aylett, R.S., Soutter, J., Petley, G.J., Chung, P.W.H., Rushton, A., 1998. AI planning in a chemical plant domain. Proceedings, ECAI '98, pp. 622–626.

Aylett, R.S., Soutter, J., Petley, G.J., Chung, P.W.H., Edwards, D.W., 1999. Increasing planning efficiency and modelling expressiveness through the use of Pairs. Proceedings, 18th Workshop of the UK Planning and Scheduling SIG, University of Salford, Dec. 1999.

Bernard, D., Dorais, G., Fry, C., Gamble, E., Kanefsky, B., Kurien, J., Millar, W., Muscettola, N., Nayak, P., Pell, B., Rajan, K., Rouquette, N., Smith, B., Williams, B., 1998. Design of the remote agent experiment for spacecraft autonomy. Proceedings of the 1998 IEEE Aerospace Conference, Aspen, CO.

Blum, A., Fust, M., 1997. Fast planning through planning graph analysis. Artificial Intelligence 90, 282–300.

Castillo, L., Fdez-Olivares, J., Gonzalez, A., 1999. Automatic generation of control sequences for manufacturing systems based on nonlinear planning techniques. Artificial Intelligence in Engineering, 1999.

Cayrol, M., Regnier, P., Vidal, V., 2000. New results about LCGP a least-commitment GraphPlan. Proceedings, AIPS 2000 (Eds.), Chien, S., Kambhampati, S., pp. 273–282.

Chapman, D., 1987. Planning for conjunctive goals. Artificial Intelligence 29, 1987.

Chien, S.A., Govindjee, A., Estin, T., Wang, X., Griesel, A., Hill Jr., R., 1997. Automated Generation of Tracking Plans for a Network of Communication Antennas. Proceedings 1997 IEEE AeroSpace Conference, Aspen, CO, Feb. 1997.

Chien, S., 1994. Using AI techniques to automatically generate image processing procedures: a preliminary report proceedings, AIPS 94, Chicago IL, pp. 219–224.

Crooks, C.A., Macchietto, S., 1992. A combined MILP and logic-based approach to the synthesis of operating procedures for batch plants. Chemical Engineering Communications 114, 117–144.

Erol, K. 1995. Hierarchical task-network planning systems: formalization, analysis and implementation. Ph.D Thesis, Computer Science, University of Maryland.

Fikes, R.E., Hart, P.E., Nilsson, N., 1972. Learning and executing generalised robot plans. Artificial Intelligence 3, 251–288.

Foulkes, N.R., Walton, M.J., Andow, P.K., Galluzo, M., 1988. Computer aided synthesis of complex pump and valve operations. Computers and Chemical Engineering 12, 1035–1044.

Fusillo, R.H., Powers, G.J., 1987. A synthesis method for chemical plant operating procedures. Computers in Chemical Engineering 11 (4), 369–382.

Georgeff, M.P., 1987. Planning. Annual Review of Computer Science 2, 359–400.

Ivanov, V.A., Kafarov, V.V., Perov, V.L., Reznichenko, A.A., 1980. On algorithmization of the start-up of chemical productions. Engineering Cybernetics 18, 104–110.

Kambhampati, S., Lambrecht, E., Parker, E., 1997. Understanding and extending graphplan. In: Steel, S., Alami, R. (Eds.), Recent Advances in AI Planning, 4th European Conference on AI Planning, ECP97. Springer, Berlin, pp. 260–272.

Kinoshita, A., Umeda, T., O'Shima, E., 1981. An algorithm for synthesis of operational sequences of chemical plants. 14th Symposium on Computerized Control and Operation of Chemical Plants, Vienna, Austria, 1981.

Long, D., Fox, M., 1998. Type analysis of planning domain descriptions. Proceedings, 17th Workshop of the UK Planning and Scheduling SIG, University of Huddersfield, ISSN 1368-5708, pp. 123–132.

McLuskey, L., Porteous, J., 1997. Engineering and compiling domain models to promote validity and efficiency. Artificial Intelligence 95 (1), 1–65.

Penberthy, J.S., Weld, D.S., 1992. UCPOP: a sound, complete, partial order planner for ADL. Proceedings of the Third International Conference on Knowledge Representation and Reasoning.

Petley, G., Aylett, R.S., Chung, P.W.H., Rushton, A., 1998. Development of a reusable operator library for chemical plant domains. Proceedings of the 17th Workshop of the UK Planning and Scheduling Special Interest Group Huddersfield University.

Rivas, J.R., Rudd, D.F., 1974. Synthesis of failure-safe operations. A.I.Ch.E. Journal 20 (2), 320–325.

Sacerdoti, E.D., 1974. Planning in a hierarchy of abstraction spaces. Artificial Intelligence 5, 115–135.

Schank, R.C., Abelson, R.P., 1977. Scripts, plans, goals and understanding—an inquiry into human knowledge structures. The Artificial Intelligence Series. Lawerence Erlbaum Associates, Hillsdale, New Jersey.

Soutter, J., 1997. An integrated architecture for operating procedure synthesis. Ph.D. Thesis, Loughborough University, Loughborough LE11 3TU, UK.

Soutter, J., Chung, P.W.H., 1996. Partial order planning with goals of prevention. Proceedings, 15th Workshop of the UK Planning and Scheduling SIG, Vol. 2. John Moores University, Liverpool, UK, pp. 300–311.

Soutter, J., Chung, P.W.H., 1997. Utilising hybrid problem solving to solve operating procedure synthesis problems. I.Chem.E. Research Event 97, vol.2. Nottingham, UK, pp. 793–796.

Tate, A., Drabble, B., Kirby, R., 1994. O-Plan2: an open architecture for command planning and control. In: Zweben, M., Fox, M.S. (Eds.), Intelligent Scheduling. Morgan Kaufmann, Los Altos, CA.

Veloso, M., Carbonell, J., Perez, M.A., Borrajo, D., Fink, E., Blythe, J., 1995. Integrated planning and learning: The PRODIGY architecture. Journal of Experimental and Theoretical AI 7(1).

Weld, D., 1994. An introduction to least-committment planning. AI Magazine 15, 27-6.

Weld, D., Etzioni, O., 1994. The first law of robotics (a call to arms). In Proceedings 12th National Conference of A.I. AAAI Press, 1994.

Weld, D., Anderson, A., Smith, D., 1998. Extending graphplan to handle uncertainty and sensing actions. 15th National Conference on AI, AAAI98, AAAI Press, pp. 897–904.

Wilkins, D.E., 1988. Practical planning: Extending the Classical AI Planning Paradigm. Morgan Kaufmann, Los Altos, CA.