USING DOMAIN KNOWLEDGE TO RESTRICT SEARCH IN AN AI PLANNER*

K Currie and A Tate

University of Edinburgh, UK

1 Introduction

O-Plan is a project centred around computer based *generative* planning. There are a number of features which make this a problem of interest to AI, not least the aspects of *search*. Early planners investigated the central issues of search space control, though it was not until the mid to late '70s that the Nonlin system [11] also incorporated the ability to fully *enumerate* the space, *if required*, during the search. This may seem an obvious capability but it is essential to have the capability to fully enumerate a search space, if needed, in order to be able to ensure completeness of search; if a solution exists it will be found. Early planners, and some current systems, did not have this fundamental capability. One central issue for the design of a planner which can enumerate its search space is to find means to restrict that space where possible.

The O-Plan project followed on directly from Tate's work in the '70s on Nonlin, also undertaken at Edinburgh University, though it was influenced by many other systems developed in the late '70s and early '80s. In particular it inherits features from:

- NOAH: by being based on a hierarchical representation of plans in which actions may be partially ordered.
- Nonlin: which introduced the notion of *goal structure* as a means of recording the rationale behind actions in the plan, and also the use of *typed preconditions* as an aid to search space control. A declarative Task Formalism (TF) was also used o provide a description of applications to the planner.
- Deviser: [12] itself derived from Nonlin but extended to handle *time* and *events*.
- Molgen: [10] notable for its ability to perform object selection using *least commitment principles*. This is supported by constraint formulation and porpagation techniques.
- OPM: [8] introduced the concept of *cognitive specialists* which can make certain kinds of decisions to alter the plan as it is being built. These include decisions about how to approach the planning problem, what knowledge bears on the problem, what kinds of actions to try to plan, what specific actions to plan, and how to allocate resources during planning.

O-Plan borrows from these systems, but importantly it presents a framework, or architecture, which enables these techniques to be incorporated into a single system in a uniform way. It is a system which was deliberately built upon software engineering principles from the start. The system is fully described in [5].

*This project is supported by the Alvey Directorate and the Science and Engineering Research Council (Alvey IKBS 100, SERC GR/D/58987). The support of SD-Scicon is gratefully acknowledged.

2 Tackling Search

AI planning systems are charged with trying to produce a *plan* which is a possible solution to a specified *problem*, or some task specification. In O-Plan this works as follows. We start by giving O-Plan an input problem, itself a complete plan but represented at a high level or having some unsatisfied conditions, and a set of action descriptions to work with; it returns a plan for the task in hand in terms of the given actions.

The plans produced by O-Plan are networks. The nodes in a network denote actions, and the arcs signify an ordering on action execution. Each node has information associated with it which describes the action's preconditions and effects. Preconditions are the "logical" conditions which must hold before the action can occur. Effects are those logical conditions which will hold after the action is completed. Resource information can also be associated with each plan node. Since actions can require and release resources, each plan node can say something about action-specific resource requirements.

A problem is specified to O-Plan as a high level or flawed plan. The plan will be flawed in the sense that it will have some parts missing (i.e. more detail is required), and other parts that will not work as required (e.g. there is a conflict between competing actions in the plan). O-Plan's task is to correct this input plan so that it *will* work as required. To do this, it needs to modify the input plan until a "flawless" plan is produced. Plan modifications are designed to add a required plan component to achieve an unsatisfied condition, to expand actions to lower levels of detail, or to rearrange the plan so as to remove an interaction flaw, etc. There will often be more than one plan modification possible; that is, there will often be a choice of how to achieve a goal or how to fix a fault. These choices lead to *search*. O-Plan searches through a space of partial plans, modifying one plan to obtain another. It seeks a complete plan that is free of faults.

2.1 Pruning Techniques

O-Plan incorporates a number of mechanisms, some novel and some derived from operational research (OR) or other AI planners, in order to restrict the search space the planner needs to consider. The various mechanisms are described below.

2.1.1 Condition Typing

One main form of search reduction in O-Plan is through the use of condition typing, also used in [11, 12, 13]. This technique however is not strictly independent of the domain of interest as all information about how to use condition typing to prune the search is fed into the system via the domain description language. The responsibility for engaging this pruning therefore falls on the user which may or may not be a bad thing. What sets this technique aside from the other techniques and heuristics outlined below however is that it necessarily leads to loss of completeness in the abstract search space since the domain writer takes the responsibility for a deliberate pruning of the space. Condition typing can be very successful but there is work to be done on how far this technique can be developed. It is precisely the demands of this technique that caused us to adopt the term ***knowledge based planning*** to describe our work. In practice condition typing is essential on realistic problems in order to reduce search spaces to a manageable level, and this can be done effectively by a domain writer providing instructions to the system about how to satisfy and maintain conditions required in the plan.

2.1.2 Time

At any stage of planning, an activity is represented by a node in the plan state. Each activity has distinct start and finish instants stored with the activity node in the form of *[earliest time, latest time]* windows. Thus each activity has an earliest start time, latest start time, earliest finish time, and latest finish time. A time window management algorithm has been successfully developed and is used incrementally (whenever the plan network is altered) to revise information contained in time windows. In this revision process, overall consistency of temporal constraints is checked. If the constraints are mutually inconsistent then the algorithm signals this condition and the corresponding plan state is poisoned (abandoned), resulting in an effective pruning of the search. These algorithms have extended those in Deviser [12] and have provable termination criteria.

The time window approach accepts a lack of precise knowledge about the timing of instants in the plan. No attempt is made explicitly to represent probabilistic uncertainty. However, probabilistic uncertainty might well be one reason causing the planner to have imprecise knowledge on the timing of an instant.

In specifying imprecise knowledge, the user is free to supply any range *[l,u]* for the duration of an activity or the required delay between an activity and one of its immediate successors in the plan. l must be non-negative and u must be greater than or equal to l. Thus, *[0,∞]* represents complete lack of knowledge. In general, specification of narrow ranges imposes tighter temporal constraints and enhances the possibility of pruning the search space due to mutually unsatisfiable temporal constraints. Thus, for example, it may be helpful to impose a realistic deadline on the finish time of the overall plan rather than to specify an upper bound of ∞. Analogous comments apply to specifying finish times of subplans represented by action descriptions.

The temporal representation in [3] assumes that an activity is represented by two time points: its start instant and finish instant. A range on the duration of an activity or on the elapsed time between the finish of one activity and the start of another is represented by a pair of arcs. The plan network explicitly represents certain temporal constraints, each represented by one arc. Using standard network longest path algorithms, one can then discover constraints which are implicitly represented. A longest path of length lij between instant i and instant j is equivalent to a direct arc of length lij between the same two nodes. Thus most constraints are computed on demand. Explicitly represented constraints make up a sparse network. In O-Plan this start and finish time information is attached to a single action node to allow for quicker access to temporal information but this is equivalent to the uniform representation in [3].

Time window information is sufficient to signal mutual unsatisfiability of temporal constraints when this is present. The implementation of this standard network algorithm has highlighted the need for AI systems in general to be able to import and incorporate techniques developed in other disciplines. It can be proved that this algorithm operates in polynomial time and its efficiency is enhanced as it operates incrementally, triggered on any local change to the emerging network.

2.1.3 Resources

Three types of resource have been identified as being important during a plan generation process - consumable, renewable and substitutable resources. These resource types are distinct from time, which we do not treat as a resource in its own right.

Consumable Resources. Consumable resources are those for which an initial stockpile is available which can only be depleted by actions in the plan. Our problem is one of representing and propagating resource consumption constraints in an efficient manner and guaranteeing that total usage does not exceed the initial availability. We will consider a single resource, widgets, with an overall availability of u (a known quantity). The minimum overall usage of widgets must not fall below l (a known quantity, typically 0). Thus the overall usage of widgets must lie in the range *[l,u]*. In any particular plan state we have lower and upper bounds on the number of widgets used by any action. Our plan will be valid with respect to the widget resource if it is possible to select actual usages of widgets for every action from within that action's known bounds such that the total usage in the plan lies in *[l,u]*. Because plan states can only be further constrained as the planning process proceeds, we must insist that the *[lower bound, upper bound]* widget usage range for action a must apply to the overall usage of widgets in a more detailed subplan which replaces action a.

For each action in the plan, a *[lower bound, upper bound]* range on widget usage for that action is maintained. Our algorithm maintains consistency between all such ranges. If one such range should shrink because the planner is somehow made capable of making a more precise statement of resource usage for a particular action, then this constraint is propagated throughout the plan. Such propagation may cause widget usage ranges to shrink for other actions.

Widget usage constraint consistency is maintained as follows: assume that the current plan state is valid with respect to widget usage and that each node has an attached widget usage window. These windows reflect all known widget usage constraints on the plan as it stands. Now assume that an additional constraint is imposed, i.e. the lower or upper bound of widget usage at some node of the plan is altered so that that node's resource window shrinks. There are 4 conditions which must be maintained (or reachieved) through constraint propagation. These conditions relate the resource window at a particular node n with resource windows at node n's parent, children, and siblings. Maintenance of these conditions thus involves propagation of constraints throughout the hierarchy of nodes. The resource propagation algorithm computes and maintains min-max pair information for the use of a single consumable resource. [2] gives precise details of the algorithms used.

Some general points should be made. First, there is value in having bounds as narrow as possible at any node. Narrower bounds represent tighter constraints; if each constraint introduced is as tight as reasonably possible, then the possibility of discovering that all constraints are mutually unsatisfiable is enhanced. This is helpful in pruning the overall search space. Thus there might well be later benefit in immediate investment of computational resources to compute relatively tight bounds rather than to simply use a default value like *[0,∞]*. Liberally wide min-max intervals are of limited value in pruning the search space.

Second, this algorithm considers only simple constraints. With these types of constraints the level of resource usage in one activity can influence the level of resource usage in another activity only through "overall availability" constraints. We have not accommodated the possibility, for example, that the requirement for widgets in activity2 might well depend on the level of widgets (or some other resource) used in activity1 in a more subtle way.

Third, if we arrive at a final plan with resource pairs for all nodes satisfying the governing constraints then we must assume that any value within that pair's range is feasible. At execution time, we could then observe the actual usage of resources as we go along and could use the algorithm to update resource windows and reallocate potential resources to be used. Provided our simple resource constraints are satisfied, and provided actual resource usages are observed one at a time, we are guaranteed that the evolving plan will be feasible with respect to resources. As a simple example consider a plan with only 3 nodes in sequence. The overall resource availability is *20* and the min-max pairs for nodel, node2, and node3 are each *[4,10]*. If we discover that nodel consumes *10* units of the resource then we must modify the min-max pairs for node2 and node3 to be each *[4,6]*. Thus we must assume that a valid execution can result using *[4,6]* for these ranges rather than the original *[4,10]*. Now if node2 is observed to consume 5 units of the resource then the range for node3 must be further modified to *[4,5]* and we must assume that we can execute this action with resources from within this range.

In practice this "assumption" is used to progress the execution of a plan up to a point where a resource usage problem indicates that the remainder of the plan is not feasible. Replanning is then indicated.

Shared Resources with Unit Availability. A shared resource with unit availability is not consumed but can be used by only one agent at a time. Despite an availability of 1, maintaining constraints on usage of this resource is far more difficult than maintaining constraints on a strictly consumable resource. Usage of a shared resource must be scheduled; scheduling problems are potentially computationally explosive because of the number of schedule permutations. We see no distinction between the problem of scheduling a shared resource with unit availability and maintaining any other logical condition (e.g. `(on a b) = true`) in a plan. At any instant the gadget may be either in use or not in use. Thus a precondition to allocating the gadget to an agent is that "in use gadget" is false, an immediate effect of this allocation is that "in use gadget" is made true, and an immediate effect of returning the gadget to the resource pool is that "in use gadget" is made false. Hence, no special purpose representation is proposed for shared resources with unit availability.

Shared Resources with Availability Greater than 1. Given our observation that there is no essential difference between maintaining constraints on a resource with unit availability and maintaining any other logical condition, one might be tempted to decompose a resource with availability $r > 1$ into r distinctly named resources with availability 1. Unfortunately, the resulting r resources will be substitutable and a planner could expend much useless effort trying to satisfy unsatisfiable resource constraints by trying other permutations of the usage of such substitutable resources. For example, assume that $r = 10$, that the planner has allocated units 1 through 10 to actions $a_1, \ldots, a_{10}$ in parallel, and that the planner discovers that action k also in parallel requires one unit of this resource. The planner correctly diagnoses the conflict that 11 units are required when only 10 are available. In attempting to correct the situation, the planner might then try each of 10! possible reallocations of units 1 through 10 to actions $a_1, \ldots, a_{10}$. Enough said!

It then appears essential that we somehow model a controller of the usage of any particular resource. A plan would be valid with respect to that resource if it contained a schedule of activities which met all temporal constraints required in our temporal management algorithm and also obeyed the resource constraints imposed by the controller of that resource. Resource smoothing (the management of shared resources) is an inherently intractable problem. A least-commitment approach to planning would insist that we could reassure ourselves that a plan exists which is valid with respect to every resource while respecting temporal constraints. This is a notorious problem. Although it is possible to construct simple heuristics for resource smoothing, these heuristics are not guaranteed to find a feasible solution. Thus these heuristics cannot be used if we wish to ensure completeness of the search.

Our problem differs from traditional sequencing and scheduling problems which seek to minimize such objective functions as project makespan, weighted tardiness of jobs, etc., subject to constraints on the flow of jobs through a shop, utilization of machines, etc. The primary differences and analogies are:

- our objective is simply to verify the existence of a feasible schedule (and to construct it when planning is completed),
- some of our resource constraints are analogous to those in traditional job shop sequencing and scheduling. A shared resource with availability n is analogous to n identical machines in parallel,
- many temporal constraints that result from one action's effects being a precondition to a later action do not have analogues in traditional scheduling and sequencing.
- strictly consumable resources and shared resources are not often handled within the same scheduling and sequencing model (although both are common in different models).
- sequencing and scheduling models do not provide for the possibility of resources which are renewable in the sense that some activities may be inserted into the plan which have the effect of increasing the resource's availability. Such resources are discussed briefly below.

The first two items should work toward making our task easier than that of traditional scheduling. However the rest make the typical AI planning task considerably more complicated than scheduling problems which can be handled successfully by techniques reviewed in [1] or other more recent OR sources. To speculate a little, it is reasonable to question whether present domain-independent AI planners can cope with realistic problems that have a heavy sequencing component. Such planners do well in environments where their task of plan synthesis involves primarily finding appropriate schemata to satisfy particular goals or expand particular actions, and their scheduling activity is mainly the correction of interactions. In this case scheduling takes the form of adding to an existing set of temporal constraints which can be represented without resorting to disjunction. Sequencing, on the other hand, requires extensive use of disjunction in representing possible orderings of activities thus implying lots of search. A natural area for further research is to investigate the appropriateness of known sequencing and scheduling algorithms or more flexible disjunctive plan representations within a domain-independent AI planning framework. We are considering such approaches.

Renewable Resources. Resource smoothing problems are further complicated by the introduction of renewable resources. This provides a mechanism for the possible substitutability of resources, a concept which is difficult to analyze in a traditional mathematical model. With renewable resources it may be possible to consume resourcel in a task whose effect is to produce additional units of resource2, e.g. money may be converted to fuel. Thus resourcel and resource2 become substitutable in the sense that a valid plan may be constructed out of various combinations of initial availabilities for the two resources. If resourcel and resource2 are both consumable rather than shared then it may be possible to represent constraints on their overall usage. Such constraints would be standard linear programming constraints provided that the process of producing one resource from some combination of other resources had a linear production function. However, if either resource1 or resource2 is a shared resource, the picture is considerably more complicated. Since handling shared renewable resources is inherently more complicated than handling the shared resources above, there is little point in tackling the modelling of renewable resources before a satisfactory methodology has been achieved for these problems.

2.1.4 Temporal Coherence

There may be constraints on legal states of a world model; some combination of facts may not be able to hold simultaneously in a physical state. Sets of such inconsistent facts have been referred to as domain constraints. Temporal coherence is our method of using such constraints to reduce search. The technique is fully described in [7]. This work was motivated by the difficulties of plan repair and dependency recording and by the failure of some of the earlier planners to generate some obvious plans in the blocks world. It was further motivated by a gap in the work of Chapman [4]. Chapman provided the Modal Truth Criterion (MTC) as a statement of the conditions under which an assertion will be true at a point in a partially ordered plan. Essentially, the MTC says that an assertion p is necessarily true at a point in a plan if and only if 1) there is a point necessarily before the required point where p is necessarily asserted; and 2) for every operator that could possibly come between the point of assertion and point of requirement, if the intervening operator possibly retracts an assertion which might turn out to be p, then there must be another (appropriately placed) operator which restores the truth of p whenever the intervening operator deletes it.

One can take a procedural reading of the MTC to produce a non-deterministic goal achievement procedure (e.g. as in Chapman's TWEAK). Such a procedure will form the heart of any correct "Nonlinear" planner. TWEAK searches a space of partial plans, using the goal achievement procedure as a partial plan generator and in TWEAK the space is explored breadth-first. However planning systems such as Nonlin, SIPE, Deviser and O-Plan strive for realism and so cannot afford this luxury of breadth-first search. Heuristics for selecting among the plan modification operations sanctioned by the MTC are required if plans are to be produced in acceptable time.

The MTC says nothing about an order in which to pursue goals. Possible bindings are determined by goal-ordering, so the MTC gives no guidance regarding sensible bindings for un-bound variables. The heuristic of temporal coherence addresses this problem. It suggests avoiding work on plans whose bulk preconditions do not "make sense". The bulk preconditions for a plan are the overall conditions on which the plan depends for its successful execution. We say that these preconditions do not make sense if they do not describe a physically realisable domain state. If a plan's bulk preconditions do not make sense, then the plan has internal inconsistencies, and is best avoided.

The basic principle of temporal coherence is this: *don't work on partial plans which have inconsistent bulk preconditions.* Think of this as follows. At each point in its search a planner will have a partially completed plan. The search begins with a given, or "root" plan, and each partial plan uncovered in the search will differ from the root plan by the addition of some number of operator schemata and binding of variables. Each added operator schema will have preconditions. Unless the plan is complete, flawless down to the truth of each and every precondition, there will be at least one precondition of at least one operator in the plan which is not true by the MTC.

Each assertion (precondition) which *is* true will either be true by some added operator, or true from the initial situation. We are interested in analyzing those assertions which must be true if the partial plan developed so far is to be "executable"; these assertions are the bulk preconditions for the developed plan.

Temporal coherence suggests working on those plans whose "bulk preconditions" describe a physically possible state of the given planning domain. By "possible" here we mean consistent with certain prespecified physical laws. Suppose that a plan's bulk preconditions do not describe a possible domain state. Why would this happen? It would happen only if the operators in the plan were not "causally independent" of each other, and required further sequencing to form a valid plan. Future goal achievements allowed by the MTC might well do this, but when faced with the choice between a plan which *already* requires a valid state of the world for its execution, and one which does not, it makes sense to choose the former. If possible it is best to avoid plans which require impossible initial states and the corrective work they entail. This avoidance of temporary impossibilities appears to be a good search heuristic. In our experience, this can lead to significant time savings in plan construction. Full details of the work is given in [7].

Temporal coherence is currently implemented in O-Plan in the way described above though we believe it to have much more potential [6], and we have used it in other non-implemented ideas. The basic problem of ordering is a crucial one and goes beyond goal ordering. The ordering problems associated with agenda handling in general also need to be tackled. We have made a start.

2.2 Search Order

Temporal coherence only addresses goal ordering, but the fact that there are many different types of choice open to a planning system means that other approaches to search control are necessary. O-Plan records choice left open to the system at any point during the generation of a plan and can exploit this choice with a simple but efficient backtracking mechanism. However even elaborate backtracking schemes are only supporting failure of the ordering techniques used in planning, so should be viewed as a last resort. With this in mind, O-Plan progresses its search in a local best, then global best manner by assigning priorities to the outstanding flaws in the plan in some informed manner. Typed preconditions and typed flaws, and measures of the degree of determinacy of the flaw form part of this informed assignment, but it is difficult to achieve truely opportunistic control. O-Plan's architecture resembles that of a blackboard system [5]; a recognition of the aims of opportunistic control in that design.

3 Summary

The O-Plan project included a study into search control in AI planning. However, search implies decisions at choice points and hence has to include arbitration and decision making into its overall aims. This cannot be achieved without proper dependency and representational schemes capable of allowing resolution of conflicts. The O-Plan study has given us insight into the real requirements of such schemes, though they remain difficult, and it has made a positive contribution to the current arsenal of search space control techniques.

References

[1] Baker, K. An introduction to sequencing and scheduling. *Wiley, 1974.*

[2] Bell, C.E., Currie, K.W. & Tate, A. Managing scheduling and resource usage constraints in O-Plan. *Artificial Intelligence Applications Institute* AIAI-TR-23, *1986.*

[3] Bell, C.E. & Tate, A. Using temporal constraints to restrict search in a planner. *Artificial Intelligence Applications Institute* AIAI-TR-5, *1986.*

[4] Chapman, D. Planning for conjunctive goals. *Masters Thesis, report* AI-TR-802, MIT *Artificial Intelligence Laboratory, 1986.*

[5] Currie,K.W. & Tate, A. O-Plan: the Open Planning Architecture. *Submitted to the AI Journal. Also* AIAI-TR-67. *1989.*

[6] Desimone, R. & Mallen, C. Complete, consistent goal sets: Controlling the search in non-linear plan generation. *In procs. of the First International Conference on Expert Planning Systems, 1990.*

[7] Drummond, M. & Currie, K.W. Goal Ordering in Partially Ordered Plans. *In procs.* IJCAI*-89, Detroit,* USA, *1989.*

[8] Hayes-Roth,B. & Hayes-Roth, F. A Cognitive Model of Planning. *Cognitive Science,pp 275 to 310, 1979.*

[9] Sacerdoti, E. A structure for plans and behaviours. *Artificial Intelligence series, publ. North Holland, 1977.*

[10] Stefik, M. Planning with constraints. *In Artificial Intelligence, Vol. 16, pp. 111-140. 1981.*

[11] Tate, A. Generating project networks. *In procs.* IJCAI-77, *Cambridge,* USA, *1977.*

[12] Vere, S. Planning in time: windows and durations for activities and goals. IEEE *Transactions on Pattern Analysis and Machine Intelligence Vol. 5, 1981.*

[13] Wilkins, D. Domain independent planning: representation and plan generation. AI *Journal Vol. 22, 1984.*