GENERATING PROJECT NETWORKS

Austin Tate
Department of Artificial Intelligence
University of Edinburgh
Edinburgh Scotland

Abstract

Procedures for optimization and resource allocation in Operations Research first require a project network for the task to be specified. The specification of a project network is at present done in an intuitive way. AI work in plan formation has developed formalisms for specifying primitive activities, and recent work by Sacerdoti (1975a) has developed a planner able to generate a plan as a partially ordered network of actions. The "planning: a joint AI/OR approach" project at Edinburgh has extended such work and provided a hierarchic planner which can aid in the generation of project networks. This paper describes the planner (NONLIN) and the Task Formalism (TF) used to hierarchically specify a domain.

1. AI and OR approaches to planning

The general problem of planning a task is one of very broad scope. Current work in Operations Research (OR) and Artificial Intelligence (AI) has concentrated on different aspects of the problem. We have taken an interdisciplinary approach in the hope that this will lead to a development of both these aspects.

In the OR approach, the planning process falls into two stages.

1. The constituent "jobs" of a plan are specified together with their precedence relationships (i.e. requirements of the form that one job precede another). This information defines a graph, termed a project network.
2. Various operations are performed on the project network to establish schedules and allocate resources (e.g. using critical path analysis).

OR work has been concerned with the second stage of computational operations on a given project network. The preliminary stage is performed in an intuitive, not well understood, and probably haphazard way.

It can be argued that the generation of a project network is important because of the structure it imposes on the task in hand. It forces component jobs to be isolated and necessary orderings between them considered. The project network can be used not only for predictions of how the project will be done, but also as a tool to aid in monitoring its progress and allowing bottlenecks to be identified. However, a considerable amount of effort is expended in the manual construction of project networks.

Steps towards automating the process of specifying constituent jobs for some task and for giving the precedence relationships between jobs, have been developed for representing the data to the planning process, i.e. a description of the goals of the plan and the operations (jobs) of which it might consist (notably the representation of operator schemas to STRIPS - Fikes & Nilsson, 1971). The applicability of the AI work on plan formation to the generation of project networks has been restricted because most of the work has concentrated on the production of plans with totally ordered sequences of primitive jobs rather than the networks needed for OR analysis. Avoiding as much unnecessary sequencing of primitive jobs as possible is important to permit effective scheduling by OR techniques. Recently, Sacerdoti (1975a) has explored the use of a planner able to generate a plan as a partially ordered network of actions. This work forms the basis of our approach to aiding a user to generate a project network for some task. Sacerdoti's approach cannot be described here, so a reader unfamiliar with the work should see the reference cited above.

2. An overview of the project

The "planning: a joint AI/OR approach" project has been concerned with aiding a user in the process of constructing a project network. To do this, as in the work of Sacerdoti, we have been investigating the use of a partially ordered network of actions to represent a plan (or project) at any stage of development. Any ordering in the network results from the fact that either

i) an action achieves a condition for a subsequent action, or
ii) an action interferes with an important effect of another action and must be removed outside the effects' "range".

Range here is used to mean the time between when a goal is achieved and the point at which it was required to satisfy a condition on a later node.

2.1 Task Formalism (TF)

A formalism (TF) has been specified to enable actions in a domain to be described in a hierarchic fashion. Sub-task descriptions can be written independently of their use at higher levels. The Task Formalism is intended to encourage the writing of modular job descriptions at various levels of detail. Within the specification of each task will be information about

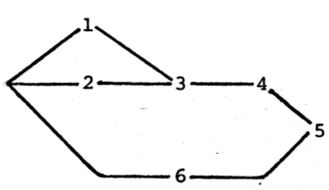
a) When to introduce an action in the plan
b) The effects of an action
c) What conditions must hold before an action can be performed
d) How to expand an action to lower level actions.

The author's present address is: Edinburgh Regional Computing Centre, Alison House, Nicolson Square, Edinburgh, EH8 9BH, U.K.

We illustrate the form of TF by an "ACTSCHEMA" from a simple house building task (the complete listing is given in Tate, 1976).

```
ACTSCHEMA DECOR
  PATTERN <<DECORATE>>
  EXPANSION
  1 ACTION <<FASTEN PLASTER AND PLASTER BOARD>>
  2 ACTION <<POUR BASEMENT FLOOR>>
  3 ACTION <<LAY FINISHED FLOORING>>
  4 ACTION <<FINISH CARPENTRY>>
  5 ACTION <<SAND AND VARNISH FLOORS>>
  6 ACTION <<PAINT>>
  ORDERINGS 1 --->3  6 --->5 SEQUENCE 2 TO 5
  CONDITIONS
  UNSUPERVISED <<ROUGH PLUMBING INSTALLED>> AT 1
  UNSUPERVISED <<ROUGH WIRING INSTALLED>> AT 1
  UNSUPERVISED <<AIR CONDITIONING INSTALLED>> AT 1
  UNSUPERVISED <<DRAINS INSTALLED>> AT 2
  UNSUPERVISED <<PLUMBING FINISHED>> AT 6
  SUPERVISED <<PLASTERING FINISHED>> AT 3 FROM 1
  SUPERVISED <<BASEMENT FLOOR LAYED>> AT 3 FROM 2
  SUPERVISED <<FLOORING FINISHED>> AT 4 FROM 3
  SUPERVISED <<CARPENTRY FINISHED>> AT 5 FROM 4
  SUPERVISED <<PAINTED>> AT 5 FROM 6.
END;
```

This schema says that an ACTION node with pattern <<DECORATE>> can be expanded into 6 lower level actions with the following partial ordering:



Conditions on nodes in the expansion are given types. SUPERVISED conditions are made true within the expansion of the task (e.g. the ACTION <<PAINT>> (6), achieves the SUPERVISED condition <<PAINTED>> on ACTION 5). UNSUPERVISED conditions are made true by other experts (mainly here by an "INSTALL SERVICES" expert). Another condition type, "USEWHEN", would say that an ACTSCHEMA containing it should not be used unless the condition was already true. It is also possible to specify the EFFECTS on a node of the expansion. In the case of the DECOR schema these would be defined by lower level actions.

2.2 The non-linear planner (NONLIN)

A planner, NONLIN, has been implemented which can generate plans from task descriptions given in the Task Formalism. It generates a plan at progressively greater levels of detail and can handle interactions between sub-plans to produce a plan as a partially-ordered network of actions. The algorithms employed in the planner have been designed so that ordering choices are avoided where possible. However, where a choice does become necessary all choice points are kept for later analysis or re-planning. A simple clear representation of the goal structure (GOST) of a plan is kept (the conditions on nodes of the network together with the points where the conditions are achieved). An example of a GOST entry during a house building task might be

```
<<SUPERVISED <<SCAFFOLDING ERECTED>> TRUE 6>>
with value [4].
```

This would mean that <<SCAFFOLDING ERECTED>> had to be true at node 6 and was made true at node 4 (nodes in a network are numbered). Node 4 here will be referred to as a "contributor" to satisfying the condition. It is possible to have several potential contributors. The GOST thus specifies a set of "ranges" for which patterns have a certain value. Goal structure provides information about a plan which would be difficult to extract from the detail of the plan itself. The use of goal structure to direct search in a problem solver was first investigated in Tate (1975). The goal structure of a plan not only provides information to aid the search of the planner, it contains valuable information for monitoring the execution of a plan.

2.3 Comparison with NOAH

NONLIN, as mentioned previously, is based upon the work of Sacerdoti (1975a) on the NOAH planner. We accept the concept of Sacerdoti's work: that ordering constraints should only be imposed between the actions comprising a plan if these are necessary for the achievement of the overall purpose of the plan. However, the NOAH program still had to make choices as to the order that actions were to be placed in a plan to correct for interactions. NOAH made this choice in one particular way. It did not keep any backtrack choice points, so this decision, once made, was irreversible. This leads to an incompleteness of the search space which can render some simple block pushing tasks unachieveable by NOAH (see section 10 of Tate, 1975 for a full account). NONLIN is capable of correcting for an interaction by suggesting two orderings (which are sufficient to ensure the incompleteness of NOAH mentioned above is avoided — see section 4.4). Other operations performed by NOAH deterministically (i.e. without generating alternative courses of action) should also be considered as choice points. Two examples of this are

a) the choice of which method to use to expand a node where alternatives exist, and

b) the decision to merge two nodes in a network.

If such decisions cannot be undone some problems are unsolvable. NONLIN keeps such choice points. We found it impractical to store all alternatives in a single AND/OR network. Instead, we make choices as they become necessary but keep alternatives for later re-use. As in NOAH, we expect that the first choice taken should lead to a solution since many of the choices made by linear planners have been avoided. Indeed, if failure occurs with the first plan being considered, our experience is that backtracking can lead to long searches since many consequent ordering choices may have been made because of an inappropriate choice early in the generation of the plan. We are tackling this problem by the use of a "Decision Graph" (see Daniel, 1977).

NOAH had no way to distinguish between im-

portant effects at nodes which achieved a condition on some later node and unimportant side-effects. If a node was introduced to achieve some goal for a later node, NOAH ensured that any effect at the introduced node was kept true up to the goal node. This can mean that no plan ordering may be found for problems to which solutions exist. NONLIN distinguishes between an effect at a node which satisfies some condition at a later node and other unimportant effects. The goals structure information which NONLIN keeps is used to enable NONLIN to suggest the minimum of two alternative orderings to correct for interactions and to distinguish important effects.

We have given careful attention to the design of a question answering program which behaves correctly for queries in a partially ordered network of nodes. This algorithm is fully described later in this paper. The algorithms used in NOAH failed to take into account interference on the truth of some statements by the effects of actions in parallel with any path from a query node to the initial situation in a plan network.

3. Task Formalism for domain specifications

At the outset of this work the problem of straightforwardly specifying a domain to a problem solver in a hierarchic fashion was recognized as being of primary importance. We wish to allow high level definitions of a task to be given, each part of which can be expanded into lower level descriptions and so on down to some arbitrary level which the user of the program requires as output, or for which "libraries" of lower level plans are available. It should be possible for each component at lower levels to be specified in a modular way — not requiring knowledge of the exact form of other components. Given any particular task, the planner must choose appropriate lower level actions so that each part of a plan can be performed successfully and so that the overall purpose of the plan is achieved.

Sub-task description — the use of condition types

When sub-tasks are being provided, the experts who produce them may know that the constituent jobs ought to be done in a particular order, or know that several jobs can be done together (in parallel). They know that certain conditions ought to hold before some jobs can proceed. For example, a carpet layer knows that before he does his job the floor boards ought to be layed, even though that isn't part of his job. These conditions are not under the supervision of this expert and are the responsibility of others. They will be termed UNSUPERVISED CONDITIONS.

Experts also know that certain conditions must be made to hold under their supervision before their task can be completed. Again, the carpet fitter knows it is his responsibility to get the carpet to the site, but the details of that task may be sub-contracted. Such conditions will be termed SUPERVISED CONDITION. N.B. as we will see these correspond to normal preconditions in means-end analysis driven systems such as STRIPS (Fikes and Nilsson, 1971).

There is a third type of condition which an expert may impose. Conditions may be stated which must hold before this expert can be called into use at all. For example, consider a block stacking expert which knows how to clear blocks by moving a block on top of the block to be cleared to some other place. If a block cannot be found to be on top of the one to be cleared it is no use calling this expert at all. If the conditions were merely stated as a goal to be achieved before the movement of the upper block to somewhere else was done, we could get into a situation where we actually move some block onto the one to be cleared and then move it off again. Such static conditions on the use of a particular expert will be called USEWHEN CONDITIONS. Usewhen conditions can be considered to be an extension to the check of relevancy of some schema which imposes them.

So, we can distinguish three different types of conditions:

(a) Unsupervised conditions
(b) Supervised conditions
(c) Usewhen conditions

Making a distinction between them can be of great benefit in controlling the number of choice points generated during a search and in choosing an alternative after a failure. Only conditions of type (b) are allowed to cause further expansions to be made to the plan being generated, i.e., allow further experts to be called in to plan to achieve the conditions. This is why they correspond to normal preconditions as specified in STRIPS. If expansions were allowed to achieve the unsupervised and usewhen conditions, we could find that the net contained much redundancy which could be difficult to resolve. It seems better to allocate jobs to appropriate experts.

Example of TF

TF is completely declarative and is based upon the operator schemas provided in STRIPS (Fikes and Nilsson, 1971). A full BNF description of the Task Formalism and further examples of its use are given in Tate (1976). However, some idea of its form can be got from the listing of one "ACTSCHEMA" from a small house building domain in section 2. Below we give a complete listing of the block stacking domain translated from Sacerdoti's (1975a) SOUP code for comparison (ℊ* is a variable prefix).

```
ACTSCHEMA PUTON
   PATTERN <<PUT ℊ*X ON TOP OF ℊ*Y>>
   CONDITIONS USEWHEN <<CLEARTOP ℊ*X>> AT SELF
              USEWHEN <<CLEARTOP ℊ*Y>> AT SELF
              USEWHEN <<ON ℊ*X ℊ*Z>> AT SELF
   EFFECTS + <<ON ℊ*X ℊ*Y>>
           - <<CLEARTOP ℊ*Y>>
           - <<ON ℊ*X ℊ*Z>>
           + <<CLEARTOP ℊ*Z>>
   VARS X UNDEF Y UNDEF Z UNDEF;
END;
```

```
OPSCHEMA MAKEON
   PATTERN <<ON $*X $*Y>>
   EXPANSION 1 GOAL <<CLEARTOP $*X>>
             2 GOAL <<CLEARTOP $*Y>>
             3 ACTION <<PUT $*X ON TOP OF $*Y>>
   ORDERINGS 1 --->3  2 --->3
   VARS X UNDEF Y UNDEF;
END;

OPSCHEMA MAKECLEAR
   PATTERN <<CLEARTOP $*X>>
   EXPANSION 1 GOAL <<CLEARTOP $*Y>>
             2 <<PUT $*Y ON TOP OF $*Z>>
   ORDERINGS 1 --->2
   CONDITIONS USEWHEN <<ON $*Y $*X>> AT 2
              USEWHEN <<CLEARTOP $*Z>> AT 2
   VARS X <:NOT TABLE:> Y UNDEF
        Z <:AND <:NOT $*X:> <:NOT $*Y:> :>;
END;

ALWAYS <<CLEARTOP TABLE>>
```

In the block stacking description, SUPERVISED conditions are omitted. TF fills these in automatically for any GOAL nodes to the following node in an expansion. The PUTON schema does not specify an expansion and only gives further effects and conditions on the action. The ALWAYS statement asserts that <<CLEARTOP TABLE>> is true in any situation. Several schemas can be given which have the same PATTERN. These then being alternative methods of expanding a node with the pattern. Choices between alternatives are handled explicitly by the planner.

4. NONLIN: The Planner

NONLIN starts with a single node representing the task to be planned. The simplified control cycle is similar to that used in NOAH.

1. Expand a node in the network using the expansion from an appropriate schema.
2. Correct for any interactions introduced.
3. Repeat from 1 until there are no further nodes to expand.

The system is run mostly "stand-alone" at present. However, it does support a simple interactive 'top-down' planning process, asking a user for information as it is found to be lacking.

4.1 Representation of the Network

The network is represented as a collection of nodes which are referred to via a subscript, e.g. NODE(4). Each node has associated with it various components:

nodenum	its NODE subscript
nodetype	GOAL, ACTION or PHANTOM*
pattern	used to seek an expansion schema
prenodes	a list of nodes linked immediately before this one
succnodes	a list of nodes linked immediately after this one
nodectxt	a context containing the effects of this node. The partially ordered network of contexts is defined by the nodectxts and the prenodes and succnodes links.
parentnode	the node was inserted as a result of the expansion of its parentnode
nodemark	a temporary marker used to record the relation of this node to some other node in the network ("BEFORE", "NODE", "AFTER" or "IN PARALLEL"). This is used during question answering.

*A PHANTOM node is a GOAL node whose pattern was already true in the network at the point where it is placed.

Other components can be ignored for the purposes of this paper. The network thus described gives the ordering constraints between the nodes in the network. 2 other structures are used to represent a plan, a TOME and a GOST.

Table of Multiple Effects (TOME)

As in Sacerdoti (1975a) we keep a record of what values are given to patterns at each node. The TOME is used during question answering and to detect interactions.

Goal Structure (GOST)

A condition of any type on any node in a plan is stored in GOST together with a list of "contributors". Contributors are nodes, any one of which could make this condition hold. See section 4.3 for the method used to find the contributors for any pattern.

The Goal Structure allows the purposes of any particular effect at any node to be determined (if it has any). This allows interactions to be detected and allows corrections (suggested linearizations) to be sensitive to the important effects of nodes (those which satisfy some condition). Unimportant effects are therefore ignored. Once the interacting effects of nodes are determined and the goal structure is available, simple linearizations can be suggested to remove the interactions (as in INTERPLAN — Tate 1975).

4.2 Expanding a Node

A node is expanded to get more detail of how a task can be performed or a goal achieved.

1) GOAL nodes A goal node is present to state that the pattern of the node should be true at the node. There are three ways this could be achieved.
   a) If the pattern was already true at that point.
   b) If we could introduce links into the network to make the pattern true at that point.
   c) If we could make an expansion of the node which would make the pattern be true.

   In cases (a) and (b), the goal node is returned with a new type "PHANTOM". A GOST entry with a special condition type "PHANTOM" is made to show the contributors which make the pattern true at the node. Links will have been put in the network as a result of (b).

   In case (c), it is necessary to find an expansion for the pattern and replace the goal node in the network by the expansion. One member of the list of schemas which can be used to expand any pattern is chosen and alternative ways to expand the pattern are kept as choice points.

2) ACTION nodes An action node is present as a command to do something. No attempt is therefore made to see if its pattern is true or can be made true (by linking) as cases (a) and (b) for goal nodes. However, case (c) is performed exactly as for goal nodes. An expansion of the pattern is sought and used to replace the action node in the network. An action node is allowed to have a null expansion. This indicates to the system that the action can be considered primitive and it should not be replaced in the network or expanded further. A shorthand TF form PRIMITIVE ... ; can be used to declare primitives.

4.3 Question answering in a partially ordered network of contexts

Current data base systems which provide a context mechanism, e.g. CONNIVER (McDermott and Sussman, 1972), provide efficient facilities for storing a changing data base by remembering the alterations made to an initial situation. However, they only provide facilities for the determination of the value of a pattern with respect to a given context in a fully ordered tree of contexts. In a tree of contexts there is a strict time sequence along a single context path so answers are fully determinate. In the partially-ordered network, answers will depend on the nodes in parallel with a particular node as well as the answer got by retracing back through a network. This answer, got by retracing through the network, will itself vary as nodes are linked earlier in the network. A full world model kept at each context would have to be continuously updated. So, as for a tree of contexts, it is best to store only the changes to an initial world model at each node.

We have provided a QA system for such a world model which can respond to two kinds of query:
(a) Does statement P have value V at node N in the current network? It could have value definitely V, definitely not V, or be undecidable.
(b) What links would have to be added to the network to make P have a certain value at N if it did not have this value in the given network?
The system finds lists of "critical" nodes in the network and uses these to give a truth result for requests of type (a). The lists contain the information needed to suggest links if a request of type (b) is given.

defns - P-node is a node which gives statement P a value.
PV-node is a node which gives statement P a value V.
P$\overline{V}$-node is a node which gives statement P a value other than V.

a critical node for (P,N) is a node which, in a possible linearization, gives a value to statement P which could be maintained up to node N.

N.B. The critical nodes for (P,N) are
i) the last P-node on each incoming branch to N (ignore P-nodes which are also predecessors of any other critical nodes since there may be redundant links in the network).
ii) all P-nodes which are in parallel with N.

QA (P,V,N) finds the lists of critical P-nodes by marking the other nodes of the network with their position with respect to N and looking for TOME entries for the statement P.
P definitely has value V at node N if there is at least one critical PV-node before node N, and there are no critical P$\overline{V}$-nodes.* P definitely does not have value V at node N if there is at least one critical P$\overline{V}$-node and there are no critical PV-nodes.

If neither of these 2 definite cases arises then it may be possible to make P have value V at node N by making suitable links in the network if this is required. We must have at least one critical PV-node linked in before node N and link out all critical P$\overline{V}$-nodes (both parallel to and before N), so that at least one path from a critical PV-node to N has no P$\overline{V}$-node in parallel with it. Since both the PV-nodes and P$\overline{V}$-nodes involved may be contributors to conditions on later nodes, the suggestion of links must be sensitive to the goal structure. The process used to suggest compatible links for this scheme is very similar to the process which corrects for interactions in a network. The common procedure used is described next. It is provided with the lists of critical nodes found during question answering.

4.4 Linking process for the network

There are 2 occasions on which it is necessary to suggest links in the network.
a) to detect and remove interactions
b) to make a statement have a particular value at some node.

We use a common procedure for both these tasks. The overall idea is very simple. It relies on having the goal structure of a network available. Goal structure gives a set of "ranges" for which a statement must have a particular value. A statement is given a value at a particular node and must retain this value up to a node which requires the statement as a condition. Our process simply ensures that there is no overlap between any ranges for which a statement P must have value V and any ranges for which a statement P must have a value other than V (i.e. $\overline{V}$ in our previous notation). We take into account 2 facts
i) where there are multiple contributors to a condition on any node, we are only constrained to maintain one of them as contributor.
ii) If the condition is only present to make a GOAL node a PHANTOM node we can remove all its contributors if necessary and this will merely force us to consider ways to achieve the GOAL.

The detail of the operation is described in Tate (1976). It emerges that once a pair of conflicting ranges are identified it is necessary to suggest both putting a link from the end of one range to the beginning of the other and vice versa (if this is compatible with the existing links in the network). This is needed to avoid the incompleteness mentioned in section 2.3. This process is a generalization of the interaction correction procedure first suggested for linear problem solvers in Tate (1975).

*The PV-nodes before node N are the "contributors".

It is vital that comparisons of all ranges specified in the goal structure are not being made continuously to check for interactions. Our method ensures that only those ranges jeopardized by any operation on the network need to be checked. We outline below how this is done for the two different uses to which the linking procedure is put.

a) To detect and remove interactions

As effects are added to nodes in the network, they are also recorded in the Table of Multiple Effects (TOME). As they are added we can find if an interaction resulted by performing two checks.

i) see if any parallel node has an opposite value for the statement (a check on the TOME). The network will already have been marked with respect to the node at which the effect was added, as a result of question answering.

ii) see if the node given the effect is in parallel with any range for which the statement must have a different value (a check on the GOST).

The linking procedure is only entered with any conflicting nodes or ranges, thus limiting the computation needed.

b) To make a statement P have a particular value V at some node N

We mentioned in section 4.3 that the QA routine can provide lists of "critical" nodes which can be used to suggest links to make a statement have a particular value at some node. Given these lists, the operation can be performed by ensuring that there is at least one critical PV-node "linked-in" before node N. This may already be the case, but if not, a choice point is made and one of the critical PV-nodes is linked before node N. The linking procedure is then used to "link-out" all critical PV-nodes from the PV-range which establishes the condition on node N. So here again we drastically reduce the potential range conflicts which need to be compared by using the lists of "critical" nodes provided by the question answering routine.

5. Summary

We have used recent work in AI aimed at generating plans as partially ordered networks of actions to aid in project network construction. Such networks are in a suitable form for the use of Operations Research optimization techniques. The present NONLIN system is a development of NOAH (Sacerdoti, 1975a). However, we have sought to improve over NOAH in several important ways.

a) Interactions are corrected for in all legal ways to avoid an incompleteness present in NOAH. In fact only 2 alternative orderings need be considered in order that this is achieved.

b) Interactions are corrected for only on the "important" effects of nodes (those which are required as the contributor to a condition on a later node).

c) We use a question answering procedure which behaves correctly for queries in a partially ordered network of nodes.

d) All alternatives generated at choice points in the planner's search space are kept for backtracking. NOAH did not keep alternatives where it made arbitrary choices.

The provision of an explicit record of the conditions on any node together with the nodes which achieve those conditions (the goal structure of the network) has provided a simplified representation of the plan which is of benefit in directing the planner's search (e.g. for (a) or (b) above). More detail of the NONLIN program and the procedures used can be found in Tate (1976). This paper also gives examples of the use of the program.

We have developed a Task Formalism (TF) to enable a group of people to co-operatively describe a task to the system with the planner's aid. TF is completely declarative and this has aided us in providing the Table of Multiple Effects (TOME) and Goal Structure (GOST). The declarative form of TF descriptions is also proving of use in the design of a "Decision Graph" to localize the alterations which need to be made to a network to recover from a search failure (see Daniel, 1977).

We are currently engaged on an investigation of project planning in the scheduling of generator maintenance in power stations. We hope to gain a better understanding of the formal channels of communication used between the planner in an organization, management who gives directives to the planner and people from whom the planner gets information to enable him to plan a project. We hope to test our present ideas of how this process is performed (as modelled in NONLIN and TF) on a realistic application in this domain.

Acknowledgements

This research work was performed on a Science Research Council Grant held by Professor B. Meltzer (Grant No. B/RG/94455). The work has benefited from discussions with Gottfried Eder and my co-worker on the project, Lesley Daniel.

References

Daniel, L. (1977). Project planning: modifying non-linear plans. Forthcoming DAI Memo.

Fikes, R.E., and Nilsson, N.J. (1971). STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2, pp. 189-208.

McDermott, D.V. & Sussman, G.J. (1972). The CONNIVER Reference Manual, MIT AI Lab., Memo No. 259.

Sacerdoti, E.D., (1975a). The non-linear nature of plans. Advance papers of 4th International Joint Conference on Artificial Intelligence (IJCAI4), Tbilisi, USSR, pp. 206-214.

Sacerdoti, E.D. (1975b). A structure for plans and behaviour, SRI AI Center, Technical Note 109.

Tate, A. (1975). Using goal structure to direct search in a problem solver. Ph.D. thesis, Machine Intelligence Research Unit, University of Edinburgh.

Tate, A. (1976). Project planning using a hierarchic non-linear planner. Research Report No. 25, Department of Artificial Intelligence, University of Edinburgh.