Repairing Plans On-the-fly

Brian Drabble, Jeff Dalton & Austin Tate
Artificial Intelligence Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom
Tel: (+44) 131 650 2732 Fax: (+44) 131 650 6513
E-mail: drabble@cirl.uoregon.edu, j.dalton@ed.ac.uk & a.tate@ed.ac.uk

Abstract

Even with the most careful advance preparation, and even with inbuilt allowance for some degree of contingency, plans need to be altered to take into account execution circumstances and changes of requirements. We have developed methods for repairing plans to account for execution failures and changes in the execution situation. We first developed these methods for the Optimum-AIV planner designed to support spacecraft assembly, integration and verification at ESA, and later deployed for Ariane IV payload bay AIV. This system was itself based on our Nonlin and O-Plan planning algorithms and plan representation. We subsequently refined the methods for the O-Plan planner and incorporated plan repair methods into the system. This paper describes the algorithms used for plan repair in O-Plan and gives an example of their use.[1]

Introduction

Even with the most careful advance preparation and even allowing for some degree of contingency pre-built into the plans, any plan being executed in the real world will have to be adapted to take into account execution circumstances and changes of requirements. For example, a deep space probe may require to adapt to new science experiments as new information leads to further experiments. Alternatively, cases such as Galileo have shown that failures in the spacecraft's hardware may need to be overcome by altering the current set of tasks and plans.

One of the aims of the O-Plan project (Currie and Tate, 1991; Tate et. al. 1994b; Tate et. al. 1996) during Phase II of the DARPA/Rome Laboratory Planning Initiative (Tate, 1996a) was to develop techniques to allow plans to be changed to take into account modifications in the task requirements and in the execution environment. The techniques allowed a failure to be identified and repaired with minimum impact on the rest of the plan.

The basis for the techniques was first developed for the Optimum-AIV planner (Aarup et. al. 1995; Tate, 1996b) designed for spacecraft assembly, integration and verification support at ESA and later deployed for Ariane IV payload bay AIV.

This paper will briefly describe some of the background work on O-Plan and Optimum-AIV, and then describe the algorithms used for plan repair in O-Plan. The paper describes a demonstration which was conducted in a command, planning, and control environment of the US air force. The task was to evacuate a number of foreign nationals from the fictional island of Pacifica (Reece et. al. 1993) and to transport them to safety. While the example is not directly related to the space domain, the demonstration does show how new requirements and changes in the environment can be integrated into an ongoing and executing plan and would be of use in the solving problems such as AIV, control of autonomous spacecraft, and lander missions.

[1]Brian Drabble is now a member of the Computational Intelligence Research Laboratory, University of Oregon.

Optimum-AIV – Assembly, Integration and Verification Planning

Planning is a key issue in the management of the assembly, integration and verification (AIV) activities of a space project. Not only must technological requirements be met, but cost and time are critical. There are costly testing facilities which must be shared with other projects, and there is a need to plan the coordination between a number of participants (agencies, contractors, launcher authorities, users). A delay caused by one participant normally leads to serious problems for others. Managers at all levels of a space project are concerned with planning, and they control closely the progress of the work. However, it has been difficult to find computer-based planning aids which meet the needs of this application. General purpose project management software cannot represent the wide range of factors to be taken into account, and is too complex to be used to interactively modify plans during project execution (Parrod et. al. 1993). For this reason, the European Space Agency commissioned the Optimum-AIV system which utilizes AI planning representations and techniques (Aarup et. al. 1995; Tate, 1996b).

The system which was developed was based on the earlier Nonlin (Tate, 1977) and O-Plan (Currie and Tate, 1991; Tate et. al., 1994b) planning algorithms and plan representation. The following techniques are used in Optimum-AIV:

- Optimum-AIV adopts a partially-ordered plan representation, which supports causally independent activities that can be executed concurrently.
- It searches through a space of partial plans, modifying them until a valid plan/schedule is found.
- The system employs hierarchical planning. The term hierarchical refers to both the representation of the plan at different levels, and also the control of the planning process at progressively more detailed levels.
- During plan specification and generation, the system operates on explicit preconditions and effects of activities that specify the applicability and purpose of the activity within the plans. With this knowledge, it is possible to check whether the current structure of the plan introduces any conflicts between actual spacecraft system states, computed by the system, and activity preconditions, which have been specified by the user. Such conflicts would arise if one activity deletes the effect of another, thus removing its contribution to the success of a further activity. The facility for checking the consistency of the plan logic, by dependency recording, is not possible within existing project management tools, which assume that the user must get this right.
- Detailed constraints are associated with the plan. These represent resource and temporal constraints on the activities in the plan as well as a more general class of global activity constraints. The scheduling task in Optimum-AIV is considered as a constraint satisfaction problem solved by constraint-based reasoning. The constraints are propagated throughout the plan, gradually transforming it into a realizable schedule. Invariably not all of the constraints can be met, such that some have to be relaxed via user intervention.
- During planning, the system records the rationale behind the plan structure; that is, user decisions on alternatives are registered. This is used to assist during plan repair where the user tries to restore consistency. Information can then be derived about alternative activities, soft constraints that may be relaxed, and potential activities that may be performed in advance.
- Test Failure Recovery Plans are available as plan fixes to enable the plan to be brought back on track after the failure of a test during the assembly and integration process. The same AI planning methods used to generate a plan are also used to assist in fixing such problems. Optimum-AIV assists the user in plan repair in an interactive way rather than performing the

repair itself.

Following an evaluation of Optimum-AIV at ESA, it has been reported (Parrod et. al, 1993) that the system is in use for planning the production of the vehicle equipment bays of the European Ariane IV launcher. It was reported that the system was chosen by the Araine IV project team due to the following:

- the wealth of information which can be provided to and used by the tool to describe the constraints inherent in the AIV activity.
- the quality of support provided by the tool to allow resource conflicts to be resolved.
- the clear representation of information and the interactive capabilities which enables engineering management to access several planning scenarios on-line.
- the fact that Optimum-AIV provides a single solution to problems of managing the plan, schedule, and allocation of resources amongst competing vehicle equipment bays which are concurrently being assembled.

Optimum-AIV provides a rich plan representation and aids to allow for the editing of AIV planning information and a wide range of constraints on the process. This information forms a basis for plan generation, checking of plan logic, and analysis of plans. Facilities are available to allow for the interactive repair of executing plans when tests indicate failures of components under assembly and integration. Optimum-AIV is an example of a deployed application of a number of AI planning techniques.

O-Plan Demonstration and Scenario Description

A demonstration experiment was performed which showed O-Plan (Tate et. al., 1996) solving a number of tasks from an integrated command, planning and control scenario related to the performance of Non-combatant Evacuation Operations (NEOs) on the fictional island of Pacifica (Reece et. al. 1993). The aims of the demonstration were to show:

- O-Plan reacting to changes in the environment and identifying those parts of the plan which were now threatened by these changes.
- O-Plan reacting to changes in the overall task by integrating new plan requirements into the plan.

The types of plan repairs explored in this demonstration include responses to failures of trucks due to blown engines and tyres and the inclusion of new task objectives, such as to pick up an extra group of evacuees. The Pacifica scenario used for the demonstration is a simplification of a real logistics problem of interest to the DARPA/Rome Laboratory Planning Initiative (Tate, 1996a). The plan schema library for this domain contained 12 schemas which defined alternative evacuation methods: trucks or helicopters, fuel supplies, transport aircraft, etc. The plans generated contained an average of 20 actions and were developed in approximately 40-60 seconds. Four different repair plans were used in the demonstration:

- Three cases repairing a broken engine on a ground transport:
  - The engine can only be fixed by a repair crew which is dispatched from the Pacifica airport at Delta with a tow truck. The ground transport is then towed to Delta for repairs. The evacuees remain with the ground transport while it is being towed.
  - The failure of the transport occurs in a time critical situation and there is insufficient time to tow the broken transport to Delta. The evacuees are moved from the broken ground transport by helicopter to Delta and the transport is abandoned.
  - The failure of the transport occurs in a time critical situation, and the evacuees are moved by another ground transport instead of by helicopter.

- One case reparing a blown tyre on a ground transport:
  - The driver of the ground transport can fix the tyre by the side of the road. The effect of the repair action is to delay the ground transport by a fixed amount of time.

In addition, a closely allied Ph.D student project by Glen Reece developed a more comprehensive reactive execution agent called "REA" (Reece, 1994; Reece and Tate, 1994) based on the O-Plan architecture. It has been used to reactively modify plans in response to operational demands in a simulation of the Pacifica island in the context of a NEO.

O-Plan Plan Repair Algorithms

The plan-repair mechanisms allow O-Plan to integrate a number of pre-assembled repair plans—e.g., to repair a blown engine, or to repair a flat tyre—into an ongoing and executing plan. Although the integration was performed by the planning agent, the techniques and methods could easily have been added to the capabilities of a separate execution agent – as in Reece's REA.

O-Plan's internal plan representation contains two tables used by the plan repair algorithms to determine the consequences of failures: the **Table of Multiple Effects (TOME)** and the **Goal Structure Table (GOST)**. Plans contain actions (nodes), and actions can have effects. Effects can take place at either end of an action: (`begin_of`) or (`end_of`). Each effect is recorded in the TOME by an entry of the form ⟨pattern⟩ `=` ⟨value⟩ `at` ⟨node-end⟩. For example, `(colour_of ball) = green at end_of node-1`.

When an action depends on an effect asserted earlier, that is recorded in the GOST by an entry of the form ⟨condition-type⟩ ⟨pattern⟩ `=` ⟨value⟩ `at` ⟨condition-node-end⟩ `from` ⟨contributor-node-ends⟩. This specifies a protected range: ⟨pattern⟩ = ⟨value⟩ is asserted at one of the contributor-node-ends and is required at the condition-node-end. For example, `unsupervised (colour_of ball) = green at begin_of node-2 from (end_of node-1)`.

These tables are maintained by the O-Plan TOME and GOST Manager (TGM) - a plug in constraint manager in the O-Plan Architecture (Tate et. al. 1996). A plan repair is required when one or more of the GOST entries are broken—i.e. a contributor of a GOST entry is not asserted as expected, or an external world event occurs and has extra effects that break a protected range by undoing a required effect.

Plan repairs are dealt with by a number of knowledge sources—pieces of code which deal with a specific aspect of the planning problem. The knowledge sources are responsible for determining the consequences of unexpected events, or of actions that do not execute as intended, for deciding what action to take when a problem is detected, and for making repairs to the effected plan.

O-Plan maintains an agenda of "issues" that need to be resolved in the plan. For each type of issue, there is a corresponding issue handler (called a knowledge source in O-Plan). The top-level control structure in O-Plan is a loop that repeatedly selects an issue from the agenda and calls the appropriate knowledge source. When describing algorithms below, we will therefore sometimes speak of "posting" an agenda entry, where the issue type is represented by the knowledge source name (KS-CONTINUE-EXECUTION, KS-FIX, etc.)

The two types of problems that are dealt with by the repair mechanisms can now be described in more detail:

- **Execution Failure**:
  An execution failure occurs when one or more of the expected effects at a node-end fail to be asserted. For example, the node-end corresponding to the end of the action `Check_out_ground_transport` should assert that the status of the engine and tyres is fine: `(engine_status gt1) = working` and `(tyre_status gt1) = working`. This may not in fact be so if the action has not executed correctly. This type of failure may cause problems if the expected effects of the action are needed

to satisfy the preconditions of a later action. For example, the evacuation of people from an outlying city can only precede if the tyres and engine of the ground transport continue to function correctly.

- **Unexpected World Event**:
  Unexpected events cause effects in the world which can make planned actions fail. For example, a landslide event may have the effect `(road_status Abyss_to_Barnacle) = closed` and this would interfere with any action requiring the road to be open.

The description of the algorithms of the execution and plan repair system is divided into three main sections. The first describes how the system maintains an execution fringe of the node-ends awaiting execution; the second describes how the system deals with plan failures; and the third describes how it handles unexpected world events.

Further details of the algorithms and the demonstration experiments is given in Drabble et. al. (1995).

Maintaining the Execution Fringe and "Necking" the Plan

An activity is represented in an O-Plan plan as a node with two ends (time points). Conditions and effects can be attached to either end of a node and are monitored by the execution system. The system reasons purely in terms of conditions and effects at node-ends and not in terms of their associated activities or events[2].

The "execution fringe" is the list of node-ends currently ready for execution. A node-end is ready when all node-ends that must execute before it in the partially ordered plan have completed execution.[3] When ready, it can be dispatched for execution. That involves sending a message to an execution agent, which in turn sends messages to a world simulator. The simulator maintains a picture of the world in which execution is taking place, for demonstration purposes. As actions begin and end in the world, the demonstration simulator reports back to the execution agent, resulting in success and failure messages about the corresponding node-ends being sent from the execution system to the planner. When the planner receives a success or failure message about a node-end, it marks the end as having completed execution; and that may lead to further node-ends being considered ready.[4]

By keeping track of which node-ends have finished execution, the system maintains a context within which replanning for plan repair can take place and can establish a focus point when considering where to insert repair actions—after all node-ends which have executed and before any node-ends waiting to execute. This point is known as the plan's *neck point* and a single dummy node can be added to the plan by the repair algorithm to *neck* the plan at that point, when necessary.

Note that the "ready to execute" check for a node-end E considers only whether all the node-ends that must execute before E have been executed, regardless of whether the execution was successful. It assumes that any problems due to execution failures or world events have been fixed, and it is the responsibility of other parts of the system to ensure that this is so.

A node-end that is ready can have its status set back to not-ready after a plan repair, because the repair may introduce new actions that must execute first.

Dealing with Execution Failures

When an execution failure occurs at a particular node-end, some of the expected effects may not occur. They are returned from the execution monitoring system to the planning agent as a list of failed-effects. The task of the planning system is

[2]This allows plug in temporal constraint managers to be employed such as Tachyon (Stillman et. al., 1996) or TMM (Boddy, 1996).

[3]This check considers both links explicitly in the plan and temporal constraints maintained by a Time-Point Network Manager (TPNM) and other plug in constraint managers in O-Plan.

[4]It is assumed that execution is not so rapid relative the the planner's ability to respond that the planner's model becomes significantly out of date.

to fix the plan so that any condition that needed one of the failed effects as a contributor is satisfied in some other way. The fix can be relatively simple if there is already another contributor in the GOST entry or if there is a suitable alternative contributor already present in the plan. If these simple fixes cannot be applied, then the system will attempt to add a new action to the plan. However, if nothing requires the failed effects, then the execution "failure" can be ignored.

The main algorithm used by the system to track execution and initiate repairs is as follows:

- Mark the node-end as having been executed.
- If there are no failed effects, then a repair is not needed.
- If there are failed effects then remove the TOME entries that correspond to them.
- Determine which GOST entries are affected by the failed (removed) effects. If there are none, then a repair is not needed.
- At this point there is a failure that must be repaired.
  - Search through the affected GOST entries in turn. If a GOST entry has more than one contributor, check if any are still valid. If so, reduce the contributor list; otherwise record the GOST entry as truly broken.
  - If no GOST entries are truly broken, then the repair is complete.
- At this point, some GOST entries are truly broken and result in "issues" that must be resolved. For each of the broken GOST entries, post a KS-FIX agenda entry. When that agenda entry is processed, the KS-FIX knowledge source will be invoked, and it will consider two repair methods for satisfying the condition in the broken GOST entry:[5]
  - Find an existing alternative contributor in the plan.
  - Bring in additional actions (a repair plan) which assert the appropriate effect. Any new nodes will be linked after the *neck point* described above.
- Post a KS-CONTINUE-EXECUTION to continue execution after the fixes have been made.

[5]The "fix" issue introduces a condition of type `achieve` as described in (Tate et. al., 1994a).

Certain details of the repair depend on the type of the condition recorded in the broken GOST entry. In particular, a `supervised` condition (Tate et. al. 1994a) is unlike all other types because it requires that a ⟨pattern⟩ `=` ⟨value⟩ assignment be true *across a range*, rather than only at a single point.

Suppose a broken GOST entry g has the form `supervised` $p = v$ `at` e `from` (c). Then c is a node end that asserts $p = v$, and $p = v$ must be so not only at node-end e (which is all that other condition types would require) but also at node ends between c and e that are *spanned by the condition*. These are the siblings of c and e that are explicitly linked between c and e, or the descendants of such siblings, where two node-ends are siblings if they were introduced as sub-actions of the same action.

Broken `supervised` conditions are handled as follows:

- Create a new dummy node d to act as the "delivery point".
- Link d after the neck point, before e, and before all node-ends that are spanned by the condition and have not yet been executed.
- Change the GOST entry to list d as the contributing node-end, and give $p = v$ at d as an effect in the TOME.
- Post a KS-FIX to re-establish $p = v$ at d.

The system must be consistent in its use of the "ends" (`begin_of` and `end_of`) of d to avoid "gaps" in the goal structure which would effect the meaning of the plan.

Dealing with Unexpected World Events

When a significant event that is not in the plan occurs in the world, it is reported to the planner as a time, an event pattern, and a list of effects (of form ⟨pattern⟩ = ⟨value⟩). For instance, the occurrence of a landslide might be reported as:

```
event {landslide} with effects
   {status road-a} = blocked,
   {status road-b} = blocked;
```

Events are treated the same way as plan activities except they are not placed in the plan until they have occurred. The effects may break GOST ranges in the plan and if so, the planner must try to satisfy those conditions some other way. However, even if no GOST entries are broken, the planner needs to add a node to represent the world event. This is because, even if the event's effects don't make any difference now, they may matter later on.

The new event node represents something that has definitely and already happened. So it must be linked after all node-ends that have already been executed and before all node-ends that have not yet been executed.

The algorithm for dealing with unexpected world events is as follows:

- Add an event node, E, to represent the world event. Link it as described above. Mark E as having already been executed.
- Edit the GOST to remove any contributors that can no longer contribute, and get a list of the truly broken GOST entries. A contributor is removed when:
  - the condition is at a node-end that has not been executed,
  - the contributor is a node-end that has been executed, and
  - the unexpected world-event has a conflicting effect.
- For each truly broken GOST entry g, post a KS-FIX agenda entry as in the case of an execution failure, using `end_of` E as a neck point.
- Add the world event's effects at `end_of` E.
- If there were no truly broken GOST entries, then we are finished. Otherwise, Post a KS-CONTINUE-EXECUTION to continue execution after the fixes have been made. (The fixes will be made by processing the KS-FIX agenda entries.)

Conclusions

This paper has shown that current AI planning and scheduling techniques have reached the point where they can be deployed in real-world applications. This means real plan execution in the face of uncertainty and changing circumstances must be dealt with. Systems such as Optimum-AIV and O-Plan have shown that they provide valuable support to human users in identifying the point of failure in a plan and suggesting appropriate repairs. The techniques described in this paper to support plan repair are general enough to be applied in a wide variety of planning and scheduling applications.

Acknowledgements

The O-Plan project is sponsored by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under grant number F30602-95-1-0022. The O-Plan project is monitored by Dr. Northrup Fowler III at the USAF Rome Laboratory. The U.S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either express or implied, of DARPA, Rome Laboratory or the U.S. Government.

References

Aarup, M., Arentoft, M.M., Parrod, Y., Stokes, I., Vadon, H. and Stader, J. (1994) Optimum-AIV: A Knowledge-Based Planning and Scheduling System for Spacecraft AIV, in Intelligent Scheduling (eds. Zweben, M. and Fox, M.S.), pp. 451-469, Morgan Kaufmann.

Boddy, M.S. (1996) Temporal Reasoning for Planning and Scheduling in Complex Domains: Lessons Learned, in Advanced Planning Technology, pp. 77-83, (Tate, A., ed.), AAAI Press.

Currie, K. and Tate, A. (1991) O-Plan: the Open Planning Architecture, Artificial Intelligence Vol. 52, pp. 49-86, Elsevier.

Drabble, B. (1995) Applying O-Plan to the NEO Scenarios, Appendix O in Tate, A., Drabble, B. and Dalton, J. (1995), An Engineer's Approach to the Application ofKnowledge-Based Planning and Scheduling Techniques to Logistics, Final Technical Report RL-TR-95-235, Rome Laboratory, Air Force Materiel Command, Rome, New York. Also available as DARPA-RL/O-Plan/TR/23 dated July 1995.

Parrod, Y., Valera, S. (1993) Optimum-AIV, A Planning Tool for Spacecraft AIV, in Preparing for the Future, Vol. 3, No. 3, pp. 7-9, European Space Agency.

Reece, G.A., (1994) Characterization and Design of Competent Rational Execution Agents for Use in Dynamic Environments, Ph.D Thesis, Department of Artificial Intelligence, University of Edinburgh, November 1994.

Reece, G.A. and Tate, A. (1994) Synthesizing Protection Monitors from Causal Structure, Proceedings of the Second International Conference on AI Planning Systems (AIPS-94), AAAI Press, Chicago, USA, June 1994.

Reece, G.A., Tate, A., Brown, D. and Hoffman, (1993) M., The PRECiS Environment, Paper presented at the ARPA-RL Planning Initiative Workshop at AAAI-93, Washington D.C., July 1993. Also available as University of Edinburgh, Artificial Intelligence Applications Institute Technical Report AIAI-TR-140.

Stillman, J., Arthur, R. and Farley, J. (1996) Temporal Reasoning for Mixed Initiative Planning, in Advanced Planning Technology, pp. 242-249, (Tate, A., ed.), AAAI Press.

Tate, A. (1977) Generating Project Networks, Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-77), pp. 888-893, Cambridge, MA, USA, Morgan Kaufmann.

Tate, A. (1996a) Advanced Planning Technology, AAAI Press.

Tate, A. (1996b) Responsive Planning and Scheduling Using AI Planning Techniques, Trends and Controversies, IEEE Expert - Intelligent Systems and Their Applications, Winter 1996.

Tate, A., Drabble, B. and Dalton, J. (1994a) The Use of Condition Types to Restrict Search in an AI Planner, Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), pp. 1129-1134, Seattle, USA, August 1994.

Tate, A., Drabble, B. and Kirby, R. (1994b), O-Plan2: an Open Architecture for Command, Planning and Control, in Intelligent Scheduling, (eds, M.Zweben and M.S.Fox), Morgan Kaufmann.

Tate, A., Drabble, B. and Dalton, J. (1996), A Knowledge-Based Planner and its Application to Logistics, in Advanced Planning Technology, pp. 259-266, (Tate, A., ed.), AAAI Press.