Decision-Theoretic Control of Failure Recovery

Lisa J. Burnell[1]
Department of Computer Science Engineering
The University of Texas at Arlington
Box 19015
Arlington, Texas 76019
(817) 273-3785
burnell@cse.uta.edu

Abstract

A number of strategies exist for the recovery from execution-time plan failures. One dimension in which these strategies differ is the degree of dependence on the reliability and availability of the planner's knowledge. The best strategy, however, may be dependent on a number of considerations, including the type of plan failure, the critically of the failure, the availability of resources, and the reliability and availability of the specific knowledge involved in a given plan failure instance. In our work on multi-agent, multi-modal (centralized, distributed and local) planning, we are interested in intelligently selecting and applying failure recovery strategies that are appropriate to the situation and that can cope with uncertainty. We are examining a decision-theoretic approach to diagnose plan failures and to dynamically select from multiple failure recovery strategies when an execution-time plan failure occurs.

Introduction

For failure recovery actions, complete information may not be available to a planning system. Even when it is possible, the costs of collecting and reasoning with complete information may be intractable, especially given time pressures and other resource constraints. We are examining decision-theoretic control of failure recovery and learning strategies to cope with the uncertainties and resource constraints of the multi-agent, multi-modal autonomous planning systems we are developing. By multi-modal, we mean planners that have the ability to dynamically switch among centralized, distributed and local planning modes.

Failure recovery strategies generally classify, with assumed or proven certainty, the type of error that occurred during plan execution and then select a fixed strategy to recover from that error. The error classification and recovery strategy may be either domain-independent or domain-dependent. Assumptions regarding the accuracy and completeness of the planner's domain model vary. Based on these assumptions, approaches may be placed on a "spectrum of plan debug and repair strategies" (Figure 1). On one end are approaches that use deterministic heuristics or purely syntactic analyses to debug and repair. These approaches often are efficient and require limited knowledge, but generally are limited in the level of diagnosis and repair they can perform. On the other end are approaches that are robust and more autonomous, but are knowledge and resource intensive and generally require that the planner's knowledge be accurate.

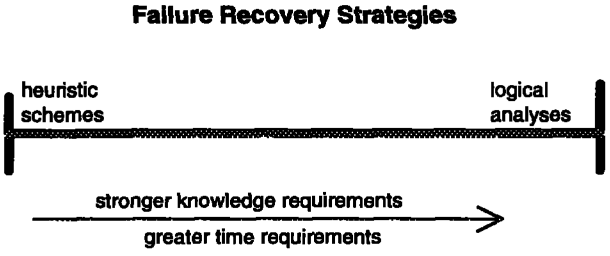
Failure Recovery Strategies
heuristic schemes
logical analyses
stronger knowledge requirements
greater time requirements

Figure 1.

In some cases, error classification is based on determining the type of error that occurs at the local point of failure. However, for learning and more in-depth error classification, the source of the error needs to be identified. This often requires tracing the previous actions of the planner to identify the root cause of the failure. From this analysis, recovery strategies that include planner modification can be identified.

But tracing previous actions of the planner is a resource-intensive task. It becomes even more difficult if the synergistic effects of multiple actions caused the failure or

[1] This work is supported in part by ARPA and the State of Texas Advanced Technology Projects under project numbers 003656-057 and 003656-008 and the National Science Foundation project MSS-9114009.

if the planner has incomplete information regarding previous planning actions, as may be the case, for example, in multi-agent planners. In order to deal with these difficulties, we are applying decision theory to guide the control of failure recovery activities and probabilistic reasoning to cope with the incompleteness and uncertainties that exist when resource constraints prohibit gathering complete, accurate information.

We are building a failure recovery approach, based on integrating and extending existing schemes, that allows for variations in strategy dependent on the mode of planning, knowledge availability and reliability, failure criticality, etc., for our planners. The selection of the appropriate strategy for a given planning failure instance is governed by decision-theoretic control. We posit that by combining the best of existing approaches, and applying them at the appropriate time, we will have more adaptable, robust planners.

The remainder of the paper is organized as follows. First, some existing failure recovery schemes are reviewed. In the next section, the basic approach is presented. Following is a simple example to illustrate some of the basic concepts of the approach. A summary concludes the discussion.

Failure Recovery Strategies

Howe and Cohen (1991) discuss a failure recovery scheme that maps classes of symptoms to domain-independent recovery strategies for the plans (e.g. replan, change some variable values). The recovery schemes chosen, and the order in which they are tried, are based on a cost-benefit analysis, using the probability of success of given strategies and a cost measure for applying the strategy. Error classification is local, i.e. no trace of previous planner actions is analyzed.

Simmons (1988) presents a debugging theory for planning and interpretation that can compensate for inadequacies in the planners domain model. Causal explanations about bugs are constructed and analyzed to find the underlying assumption on which they depend. The method can not debug if the bug is caused by the synergistic effect of multiple assumptions. Repair is achieved using domain-independent heuristics that select strategies based on classes of assumptions. The approach is computationally very expensive, and thus is recommended only when necessary as part of the generate-test-debug paradigm.

Kambhampati (1990) presents a theory for plan modifications that relies on a semantic analysis of plan failures. His strategy is applicable for (1) modifying old plans for slightly different goals, (2) modifying a plan if its initial or goal states change and (3) checking the validity of the plan during plan generation. *Validation structures* are built to capture internal dependencies and to allow reasoning about the effects of changes to the plan. Repairs are recommended based on classes of inconsistencies identified. This approach is used to correct plans, based on the assumption of a correct domain theory, for the purpose of improving planner efficiency. Hammond's (1986) CHEF addresses both plan recovery and learning that also assumes a correct domain theory. CHEF does not need to deal with execution-time plan failures or plan under uncertainty.

Approach

There are 3 primary components of our research: (1) diagnosis of plan failures, (2) plan repair and (3) planner modification. When a failure is detected, we use a probabilistic method we have developed for determining the error class and the source(s) of the error. This method has already proven effective in debugging programs (Burnell & Horvitz 1993; Burnell & Talbot 1993) , and is being adapted for use in debugging generated plans. Plan repair strategies are selected using a decision theoretic approach in a fashion similar to (Howe & Cohen 1991), with the addition of dealing with potential uncertainties in the error classification and resource constraints. Finally, machine learning methods are employed, when warranted, to correct and refine the planner itself. The primary goals of the research are to develop an approach that incorporates these 3 components, to apply the approach to multi-agent, multi-modal planners we are developing in other research and to characterize the impact of various assumptions on the type and quality of knowledge used.

There exists some parallels between plan failure diagnosis and that of autonomous debugging of algorithmic programs. Our approach combines logical and probabilistic reasoning for diagnosing errors in assembler programs (Burnell & Horvitz 1993; Burnell & Talbot 1993). While much information can be determined via a logical analysis of the program structure and related data, incompleteness and complexity often inhibit a complete diagnosis. Our approach uses probabilistic models, represented as belief networks, to guide a search for useful, relevant information that can be obtained from logical analyses and to construct an ordering over classes of errors. The belief networks model the uncertain relationships about the nature and structure of program execution paths and the likelihood that an error resides on that path. Given that an error is likely to have occurred on a given path, the evidence nodes in the belief network suggest which logical analyses to conduct in order to collect evidence for determining the most likely class of error.

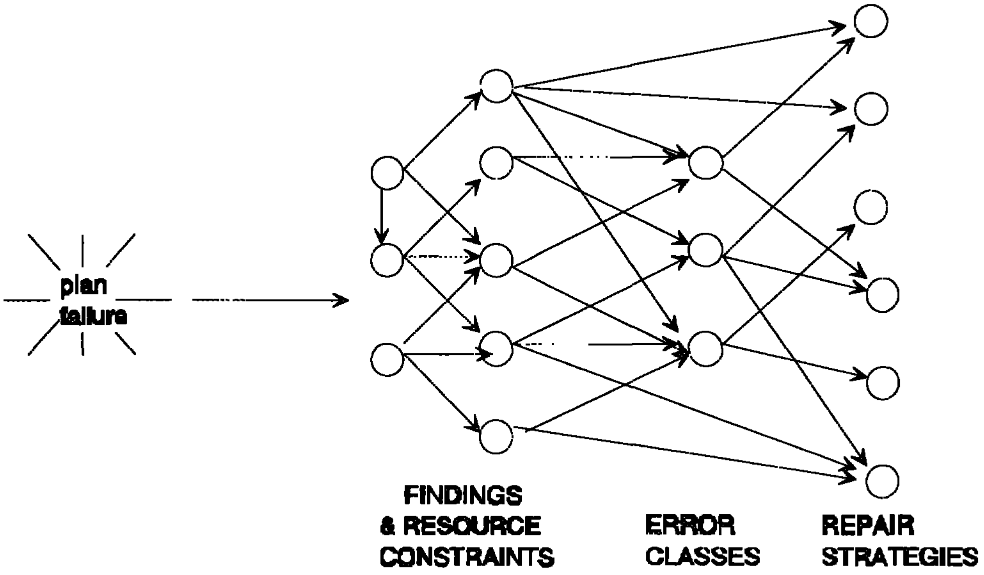

Figure 2. Upon notification of a plan failure, a belief network is activated to gather evidence from an execution trace to diagnose the most likely error class. Value of information calculations help determine what additional information is needed from a logical analysis of the trace and the planner domain theory. Error class beliefs and current resource constraints are used to recommend a good repair strategy.

Failure recovery analysis is guided by the probabilistic model, which can recommend which computationally complex actions are worth undertaking to determine the error class and an appropriate repair strategy (Figure 2). These repair strategies may be domain-independent, e.g. (Howe & Cohen 1991), or based on domain-specific heuristics, or some combination of the these. For example, in distributed, hierarchical planning for an automated manufacturing center, repair strategies for a given failure may include local selection of a reactive failure-recovery action or sending a request for replanning to a more sophisticated planner. Even when the cause of the failure can be determined with certainty, variable time or other resource constraints influence the preferred failure-recovery strategy.

Making the choice of what to learn from plan failures involves a number of issues, including the certainty with which we can identify what changes to make to operators and the effects of the changes on the correctness, efficiency and stability of the planner and the plans it generates. We adopt ideas from (Dean & Wellman 1991; Hammond 1986; Horvitz 1988).

EXAMPLE

We show a very simple blocks world (with weighted blocks) example. Two operators and a sample plan are shown in Figure 3. When a failure occurs, the following is available:

- The generated plan graph for the agent, e.g. PICKUP(A) -> ...->STACK(A,B)
- The operators (but not the trace of instantiated operators used during plan generation). Multiple operators may exist for a single plan step, e.g. two operators for stack(X Y). From this, we build a dependency graph of the operators used by each relevant plan step (multiple operators for a single action may introduce uncertainties regarding which operator was used in plan generation). The possible reasons for the PICKUP operator to fail in this example are: (1) C does not exist (2) C is not clear (3) C is not on-table (4) C is not in the expected location *(5) C does not weigh less than 200* (6) The arm is not really empty.
- Depending on the context, we may know or be able to estimate resource constraints, such as time availability or a measure of failure criticality.
- Depending on the planner domain, we may have a domain-specific belief network to use for diagnosis and recovery. If not, a domain-independent belief network is used. For the example, we'll take the case in which no domain-specific belief network exists. In this case, we don't "know" about the domain;

weight(X) and location(X) are simply two "variables" that may be initialized, modified, or used.

The domain-independent belief network that is loaded includes hypotheses and "triggers" - quickly collected pieces of evidence that support (but do not prove) the belief in a hypothesis. These include:

Hypothesis: Bad-Set class of errors:

Trigger example:

- If a suspect "variable" had been used successfully, was later reinitialized and the failed operator is the first use since the reinitialization, e.g. if the block had been picked up before and its weight was subsequently reinitialized.

Hypothesis: Resource-Not-Available class of errors

Trigger example:

- If suspect "variable" has not been used in any previous plan step.

```
(PICK-UP
    (params (<ob> <pos>))
    (preconds (and (object <ob>)
        ...
        (weight <ob> <w>) weight is TESTED
        (less-than <w> 200)
        (arm-empty)))
    (effects ((del (on-table <ob>))
        (del (location <ob> <pos>))
        (add (vacant-loc <pos>))
        (del (clear <ob>))
        (add (holding <ob>)))))
(GLUE
    (params (<ob1> <ob2> <pos1> <w1> <w2>))
    (preconds (and (object <ob1>)
        ...
        (weight <ob1> <w1>) weight is USED
        (weight <ob2> <w2>)
        (add-numb <w1> <w2> <w3>))) a function
    (effects ((del (holding <ob2>))
            (del (weight <ob1> <w1>))
            (del (weight <ob2> <w2>))
            (add (weight <ob1> <w2>)) wrong!
            (del (object <ob2>)))))
```

The correct postcondition is (add (weight <ob1> <w3>))

The planner generates the plan:

```
... (some plan steps)
GLUE(C B 100 100)
... (more plan steps)
PICKUP (C) Execution Failure: Can't Pickup!
```

Figure 3.

The belief in each failure class is calculated from the belief network, using the triggers and the results of the logical analyses selected from the value of information calculations (that calculate which pieces of evidence, if they were known, contribute most to differentiating the belief in hypotheses). For example:

```
Bad-Set:                                   0.7
        Evidence:
                weight is used, then reset
                in GLUE operator
                ...
Resource-Not-Available:                    0.1
        Evidence:
                <supporting evidence>
<Other Hypotheses>
    ...
```

Based on the most likely class of failure (if one is found), and other criteria that are known (with some degree of belief, e.g. time criticality), a recovery strategy is selected that maximizes expected utility. If a domain-specific belief network had been available, a possible recovery strategy would be "if PICKUP fails, try PUSH". In our example, we choose to reset the weight of block C to a value greater than 200 and try to replan. Additionally, since we believe a bad weight was calculated in the GLUE operator, we notify the planner modification module to consider modifying that operator. Other repair operations, for different conditions, include adding operators or adding/changing preconditions, to avoid this failure in the future.

Summary

We have briefly described a decision theoretic approach to plan failure debugging and recovery that integrates existing methods and adapts them to the multi-agent, multi-modal planners we are developing. The proposed approach is intended to fall between purely heuristic approaches to failure recovery, which are efficient but may lack robustness, and purely logical approaches, which are robust but may require more time or knowledge than is available to a planner. This is early work and there is much that remains to be done to fully develop the ideas and to test their value. We desire to measure the efficiency of the approach, the accuracy of the diagnoses and recommended repairs, and the robustness of the approach given varying levels of incompleteness and uncertainty of the knowledge. Even at this early stage, we believe this approach, which integrates and extends the best of existing approaches, has promise to increase the adaptability and robustness of autonomous planning systems that must operate in complex environments.

References

Burnell, L. J., and Horvitz, E. J. 1993. A Synthesis of Logical and Probabilistic Reasoning for Program Understanding and Debugging. In Proceedings of the Ninth Conference on Uncertainty in AI, 285-291. San Mateo, Calif.: Morgan Kaufmann Publishers.

Burnell, L. J., and Talbot, S.E. 1993. Incorporating Probabilistic Reasoning in a Reactive Program Debugging System. In Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications, 321-327. Los Alamitos, Calif.: IEEE Computer Society Press.

Dean, T., and Wellman, M. 1991. *Planning and Control.* San Mateo, Calif.: Morgan Kaufmann Publishers.

Hammond, K. 1986. CHEF: A Model of Case-Based Planning. In Proceedings of the Fifth National Conference on Artificial Intelligence, 267-271. San Mateo, Calif.:Morgan Kaufmann.

Horvitz, E.J. 1988. Reasoning under varying and uncertain resource constraints. In Proceedings of the Seventh National Conference on Artificial Intelligence, 111-116. San Mateo, Calif.:Morgan Kaufmann.

Howe, A. and Cohen, P. 1991. Failure Recovery: A model and Experiments. In Proceedings of the Ninth National Conference on Artificial Intelligence. 801-808. Menlo Park, Calif.: AAAI Press.

Kambhampati, S. 1990, A Theory of Plan Modification. In Proceedings of the Eighth National Conference on Artificial Intelligence, 176-182. Menlo Park, Calif.: AAAI Press.

Simmons, R. 1988. A Theory of Debugging Plans and Interpretations. In Proceedings of the Seventh National Conference on Artificial Intelligence, 94-99. San Mateo, Calif.:Morgan Kaufmann.