Improving Big Plans

Neal Lesh, Nathaniel Martin, James Allen*
Computer Science Department
University of Rochester
Rochester NY 14627
{lesh,martin,james}@cs.rochester.edu

Abstract

Past research on assessing and improving plans in domains that contain uncertainty has focused on analytic techniques that are exponential in the length of the plan. Little work has been done on choosing from among the many ways in which a plan can be improved. We present the IMPROVE algorithm which simulates the execution of large, probabilistic plans. IMPROVE runs a data mining algorithm on the execution traces to pinpoint defects in the plan that most often lead to plan failure. Finally, IMPROVE applies qualitative reasoning and plan adaptation algorithms to modify the plan to correct these defects. We have tested IMPROVE on plans containing over 250 steps in an evacuation domain, produced by a domain-specific scheduling routine. In these experiments, the modified plans have over a 15% higher probability of achieving their goal than the original plan.

Introduction

Large, complex domains call for large, robust plans. However, today's state-of-the-art planning algorithms cannot efficiently generate large or robust plans from first principles. We have combined work on data mining (Agrawal & Srikant 1995), qualitative reasoning (Wellman 1990), planning (Nebel & Koehler 1995), and simulation to construct the IMPROVE algorithm which modifies a given plan so that it has a higher probability of achieving its goal.

An algorithm for improving big plans is useful only if big, but not especially robust, plans can be generated to begin with. We believe domain specific and collaborative planning algorithms (e.g. (Ferguson, Allen, & Miller 1996)) will produce such plans. In our experiments, we use a greedy, domain specific algorithm to produce plans with 250 to 300 steps in an evacuation domain. This algorithm, however, ignores projected weather patterns and undesirable outcomes of actions such as buses overheating or helicopters crashing.

The IMPROVE algorithm repeatedly simulates the input plan, analyzes the execution traces to identify what went wrong when the plan failed, and then generates a set of modifications that might avoid these problems. Finally, the algorithm re-simulates to determine which, if any, of the modifications best improves the robustness of the plan. This whole process is repeated until the modifications cease to improve the plan.

In the first step, we use a discrete event simulator to generate execution traces based on a probabilistic model of the domain. We show that simulation is an attractive alternative to analytic techniques for determining the probability of a plan achieving its goal. Simulation is linear in the length of the plan and fast in practice. Further, the estimate produced by simulation converges, as more simulations are performed, to the true probability that a plan will achieve its goal.

In the second step, we use SPADE (Zaki 1997), a sequential discovery data mining algorithm to extract patterns of events that are common in traces of plan failures but uncommon in traces of plan successes. The patterns are used to determine what to fix in the plan. For example, if the plan often fails when Bus1 gets a flat tire *and* overheats, then IMPROVE attempts to prevent at least one of those problems from occurring.

In the third step, we apply qualitative reasoning techniques (Wellman 1990) to modify the plan to avoid the problems that arose in simulation. For example, if the problem is that a bus is getting a flat tire then this step might suggest changing its tires.

Finally, plan modifications can necessitate further planning. For example, adding an action to a plan requires that the preconditions of the action be satisfied, perhaps by adding more actions. We show how this problem is closely related to the problem of plan re-use or adaptation (Nebel & Koehler 1995).

Below, we first formulate the plan improvement problem. We then describe our four components: sim-

*Many thanks to George Ferguson, Mitsu Ogihara, and Mohammed Zaki as well as the AAAI reviewers for comments and discussion. This material is based upon work supported by ARPA under Grant number F30602-95-1-0025.

ulation, data mining, qualitative reasoning, planning. We then present our improvement algorithm, describe our experiments, and then conclude.

Formulation

In this section, we define our terms and specify the input and output of the plan improvement problem.

The problem of planning under uncertainty has recently been addressed by a variety of researchers. See (Goldman & Boddy 1996) and (Boutilier, Dean, & Hanks 1995) for comparisons of various approaches. We use the representation of actions used by the Buridan planner (Kushmerick, Hanks, & Weld 1995), which allows for probabilistic and conditional effects, but not for exogenous events or for actions to be executed in parallel or have varying durations. In our actual system, we allow for limited forms of all of these factors.

An *action* is a set of *consequences* $\{ \langle t_1, e_1, p_1 \rangle, ..., \langle t_n, e_n, p_n \rangle \}$, where for every i, t_i is an expression called the consequence's *trigger*, e_i is a set of literals called the consequence's *effects*, and $0 \leq p_i \leq 1$ indicates the probability that the action will have effect e_i if executed in a state in which t_i holds. The triggers must be mutually exclusive and exhaustive. For our purposes, each action a_i is associated with a set of *outcomes*, such that there is a unique outcome label $o_{i,k}$ for every possible effect e_k of action a_i.

Let $P[G \mid \mathcal{I}, \mathcal{A}]$ be the probability of goal G holding in the state resulting from executing action sequence $\mathcal{A}$ from state $\mathcal{I}$. See (Kushmerick, Hanks, & Weld 1995) for an exact description of how to compute the probability of a plan achieving a goal, as well as the probability of a state resulting from executing a given action from a given state.

A *plan improvement* problem is a tuple $\langle \mathcal{A}, G, \mathcal{I}, \mathcal{O} \rangle$ where $\mathcal{A}$ is a sequence of actions, G is a goal, $\mathcal{I}$ is the initial state, and $\mathcal{O}$ is a set of possible actions. A solution to a plan improvement problem is a sequence of actions $\mathcal{A}'$ such that every member of A' is also a member of $\mathcal{O}$ and $P[G \mid \mathcal{I}, \mathcal{A}'] > P[G \mid \mathcal{I}, \mathcal{A}]$.

The intent is that $\mathcal{A}'$ will resemble $\mathcal{A}$ and could be produced by applying a small number of simple operations (deleting, adding, or re-ordering actions) to $\mathcal{A}$.

Components

We now describe the four components of our system.

Simulation

In this section, we briefly describe how we simulate probabilistic plans. Our simulator maps from an initial state and an action sequence to an execution trace randomly chosen from the distribution of possible executions of the action sequence. To simulate a plan, we start with the initial state and for each action we "roll the dice" to determine the action's effects and, consequently, the next state. We repeat the process for every action in the plan, to produce an execution trace. For each simulation, we record which of the possible outcomes of each action occurred, as well as the sequence of states that arose.

Since simulation explores only one possible execution path at a time, the complexity of simulation is linear in the length of the plan. Previous analytic techniques for assessing plans are exponential in the length of the plan (e.g. (Kushmerick, Hanks, & Weld 1995)).

We now briefly discuss the advantages of using simulation in probabilistic planning. Standard statistical arguments (McClave & II 1982) show that simulation converges in the sense that one can guarantee there is a high probability that the estimate returned by simulation is within some ϵ of the true probability. What's more, the probability that the true probability, p, is more than ϵ away from our estimate is approximately $z_{\alpha/2}\sqrt{(p(1-p))}$.[1] For example, even making the most pessimistic assumptions about p, 10,000 simulations yields about a 95% confidence that the estimate is within 0.01 of the correct answer. This is a lot of simulations. But note that it is independent of plan length! In contrast, adding a single action can double the work required by an analytic technique.

Furthermore, a small number of simulations can be used to reject a bad plan. Suppose a plan succeeds in 20 of 100 simulations and we need a plan that succeeds with at least .6 probability. Reasoning about sample size indicates that there is well over 99% confidence that a plan that fails in 100 randomly chosen simulation has lower than a .6 probability of succeeding.

Data mining for significant flaws

In this section, we describe how we use data mining to determine what went wrong in the plans that failed during simulation. The objective is a function which outputs expressions of the form Prevent$(a_i, o_{i,k})$ which translates to "Try to prevent action a_i from having outcome $o_{i,k}$." This process is more completely described in (Zaki, Lesh, & Ogihara 1997).

Sequential discovery (Agrawal & Srikant 1995) is the problem of mining patterns from sequences of unordered sets of items, such as

$$ABC \mapsto DE \mapsto EF \mapsto GHI$$
$$CD \mapsto AB \mapsto CIJ \mapsto BG$$
$$AB \mapsto E \mapsto IJ$$

[1] Values for z can be found in any statistics textbook. For example, $z_{.05} = 1.645$, $z_{.025} = 1.96$ and $z_{.005} = 2.575$.

where $A, B, C..J$ are items. The algorithms find patterns that occur with high frequency. For example, the sequence $A \mapsto I$ occurs in all three of the above sequences, and $AB \mapsto E \mapsto I$ occurs in two of them.

We have applied the SPADE algorithm (Zaki 1997) to the execution traces produced by the simulator. We convert the ith trace into a sequence of actions $(A_1^i, ..., A_{m_i}^i)$, where each action is represented as a set composed of the action's id, the action's name, the parameters of the action, and the outcome of the action. This is a simple transformation, in that we know that action a_i was executed at time i in every trace, and so what varies from trace to trace is what outcome the action had. For example, suppose that action a_8 is a Move action on Bus1 from Delta to Abyss, and that in the 12th simulation the outcome of a_8 was Flat, then A_8^{12} would be:

(id8 Move Veh=Bus1 Frm=Delta To=Abyss Flat)

We then apply the SPADE algorithm to extract patterns that occur with high frequency in the failed plans but low frequency in the successful plans. An example pattern the data mining algorithm might return is:

(Move Bus1 Flat) $\rightarrow$ (Move Bus1 Overheat)

which indicates that Bus1 gets a flat tire in one Move action and overheats in a subsequent Move action.

We can be reasonably confident that significant trends will emerge from our data mining, if we have sufficient numbers of both successful and failed plan traces. Let *seq* be some sequential pattern of events. In order to correctly assess whether *seq* predicts failure, we need an accurate estimate of its frequency in *both* the successful and failed plans. Let C_s be the probability that the frequency of *seq* in the successful traces will be within some ϵ of the true probability of *seq* occurring in a successful plan. Similarly, let C_f be the probability that the frequency of *seq* in the failed plan traces will be within some ϵ of the true probability of *seq* occurring in a failed plan. The probability that *seq* will be represented accurately in both successful and failed plans is $C_s \times C_f$, which will be high iff both C_s and C_f are high. Thus, a significant trend is likely to emerge if our simulations include a sufficient number of both successful and failed plans.

If a pattern is common in failed plan traces but uncommon in successful traces then we speculate that preventing this sequence from occurring might increase the plan's success rate. The sequence represents a chain of events that causes failure and thus can be broken by preventing any part of the chain from occurring. For example, if data mining extracts the pattern (id8 Flat) $\mapsto$ (id17 Overheat) then we assert Prevent(a_8,Flat) and Prevent(a_{17}, Overheat). But the patterns might not include specific actions. Given the pattern Flat $\mapsto$ Overheat, we re-examine the simulation traces to find actions that often resulted in a Flat or Overheat.

More formally, we generate Prevent statements as follows. Let $\mathcal{P} = (p_{1,1} \mapsto p_{1,2}... \mapsto p_{1,m_1}), ..., (p_{n,1} \mapsto p_{n,2}... \mapsto p_{n,m_n})$ be the patterns returned by the data mining routine. For every action a_i and outcome $o_{i,k}$, let $\mathcal{A}$ be all occurrences of action a_i in the simulations where the outcome was $o_{i,k}$. For every $p_{o,l} \in \mathcal{P}$, we count the number of actions in $\mathcal{A}$ that $p_{o,l}$ matches (i.e. $p_{o,l}$ is a subset of the action in the trace). If this number is above a user-defined threshold, normally about .1 of the total number of traces, then we assert Prevent$(a_i, o_{i,k})$. The Prevent statements can be calculated in a single pass through the execution traces by keeping a separate counter for each action-outcome pair for every $p_{o,l} \in \mathcal{P}$.

The highly structured nature of the database of plan traces makes it difficult to mine rules that predict failure. A compuational problem is that there are a staggering number of highly frequent, but unpredictive, rules such as (Move) $\mapsto$ (Load) $\mapsto$ (Move) which appears in every plan trace. The typical strategy of mining all highly frequent rules and then removing all unpredictive ones is not efficient in this case. A second problem is that the existence of one rule that predicts failure, say (Bus1 Flat) $\mapsto$ (Bus1 Overheat), implies the existence of many related, equally predictive rules, such as (Move Bus1) $\mapsto$ (Move Bus1) $\mapsto$ (Bus1 Flat) $\mapsto$ (Bus1 Overheat), since the Move action is so common. (Zaki, Lesh, & Ogihara 1997) discusses these problems and describes pruning stratagies for addressing them.

Qualitative reasoner

In this section, we describe a function for mapping from a statement of the form Prevent$(a_i, o_{i,k})$ to a set of suggestions for changing the plan to decrease the likelihood that action a_i will have outcome $o_{i,k}$.

The function SUGGESTCHANGES takes in the Prevent statement, the original plan, the set of possible actions $\mathcal{O}$, and also a set of *qualitative rules* $\mathcal{QR}$. The qualitative rules indicate positive and negative influences among state variables, actions, and action-outcome pairs. Informally,

- An action a_i positively influences a boolean state variable v_j iff $P[v_j|a_i] > P[v_j]$, i.e. if v_j is more likely to hold in the current state if action a_i was just executed than otherwise.[2]

[2]Currently, we assert a positive influence only if there does not exist any state in which executing a_i decreases the probability that v_j will be true.

- A state variable v_j negatively influences an action outcome pair $\langle a_i, o_{i,k} \rangle$ iff $P[\langle a_i, o_{i,k} \rangle | v_j] > P[\langle a_i, o_{i,k} \rangle]$, i.e. if action a_i is more likely to have outcome $o_{i,k}$ if v_j is true in the state in which a_i is executed than if v_j is false.

For example, if a_i is the Change-Tire action then a_i might positively influence v_j = TireConditionGood which would negatively influence $\langle a_k, \text{Flat} \rangle$.

Positive and negative influences can be combined and chained in the obvious way: if a positively influences b and b positively influences c then a positively influences c. If a negatively influences b and b positively influences c then a negatively influences c. And, finally, if a negatively influences b and b negatively influences c then a positively influences c.

Currently, we employ the following simple strategy for inserting actions:

Func:SUGGESTCHANGES ($\langle a_i, o_{i,k} \rangle$,$\{a_1, .., a_n\}$,$\mathcal{QR}$,$\mathcal{O}$)

- If action $a' \in \mathcal{O}$ is not in $a_1, ..., a_n$ and action a' negatively influences $\langle a_i, o_k \rangle$ given the rules in $\mathcal{QR}$ then assert Insert(a', a_i) which indicates that a' should be inserted into $a_1, ..., a_n$ prior to a_i.

Currently, we hand code a set of qualitative rules $\mathcal{QR}$ for the domain.

Our approach can be described as a simple case of a qualitative probabilistic network (QPN) (Wellman 1990). QPNs are used to perform inference over qualitative influences. The complexity of determining if one node positively or negatively influences another is $O(|\mathcal{V}|^2)$ where $\mathcal{V}$ is the number of nodes. These techniques are thus suitable for large domains. We intend to extend our use of qualitative networks to take more advantage of their power including the ability to add more than one action to the plan.

Recall that data mining focused our improvement efforts on preventing some actions from having certain outcomes. There are, however, many ways of making it less likely that a given action will have a given outcome. For example, one might add any action which could make any aspect of the trigger of that effect false. We believe that qualitative reasoning is a promising technique for further focusing an improvement algorithm on a subset of the possible ways of preventing the sequences detected by data mining.

Plan adaptation

We have shown how analysis of simulated execution traces can suggest modifications to improve the plan, such as inserting an action. However, these modifications might result in an unexecutable plan. For example, the new action may have unsatisfied preconditions. We now address the problem of further modifying the plan to accommodate the new changes.

Variations of this problem arise in case-based planning (Hammond 1989; 1990), transformational planning (Simmons 1988), or plan re-use or plan adaptation (Nebel & Koehler 1995). In each case, a plan that almost solves a given goal is modified, or repaired, to solve the goal. In particular our problem, can be mapped into a plan-reuse or plan adaptation problem (Nebel & Koehler 1995), for which there are several domain-independent algorithms including SPA (Hanks & Weld 1995), PRIAR (Kambhampati & Hendler 1992), and NOLIMIT (Veloso 1994).

We focus on SPA, which is based on the SNLP partial-order planner (McAllester & Rosenblitt 1991). Planning by adaptation is similar to planning from first principles. The primary difference is that search begins from the plan to be adapted rather than from a null plan. As a consequence, plans can be refined by retracting elements from, as well as adding elements to, the plan.

Plan adaptation is suitable for large plans in that the complexity of plan adaptation is *not* exponential in the length of the given plan. The complexity is, however, exponential in the number of adaptations needed to repair the plan. In the worst case, of course, the entire given plan will be dismantled and a new plan built up. We do not expect our system to find such repairs. Rather, we are encouraged by the fact that small repairs to large plans are feasible even though large repairs to large plans are not. Our hope is that as new techniques are developed for generating plans more efficiently, these same techniques will extend the extent to which large plans can be repaired.

In the previous section, we showed how to generate suggestions of the form Insert(a', a_i). We now describe the INSERTACTION function which inserts action a' before action a_i in the given plan $a_1, ..., a_n$. We can map a call to INSERTACTION directly into a call to the SPA plan adaptation algorithm. The basic idea is to generate two unique predicates q_1 and q_2, and then add q_1 as an effect of a' and as a precondition of a_i. This will force SPA to add a' in order to keep a_i in the plan. Additionally, we add q_2 as an effect of a_i and as a precondition of the goal so that SPA cannot remove a_i from the plan, since no other action can achieve q_2. We set q_1 and q_2 to be false in the initial state.

However, the SPA function is only defined for deterministic STRIPS operators, as opposed to the probabilistic actions used in our formulation. To bridge this gap, formally, we would have to extend SPA or instead rely on abstract models of our probabilistic actions.[3]

[3]We are not claiming that either option is

For our system, we have implemented a domain-specific version of INSERTACTION. The input and output of our domain-specific program satisfy the SPA input/output specification, but the internals of the program resemble a scheduling routine more than a planning routine. It is very fast, but does not perform any general purpose reasoning. We describe the domain-specific INSERTACTION below, in the section describing the domain.

Plan improvement algorithm

In this section, we put together the pieces described above to form the IMPROVE algorithm.

As shown in figure 1, the first step of IMPROVE is to simulate the input plan many (typically 1000) times. We then feed the traces into the data mining algorithm, which produces a set of statements of the form Prevent$(a_i, o_{i,k})$ which translates to "Try to prevent action a_i from having outcome $o_{i,k}$." We then use the qualitative reasoning function, SUGGESTCHANGES to convert each Prevent statement into a set of suggested changes, of the form Insert(a', a_i) which translates to "Insert action a' before action a_i in the plan.

We then call INSERTACTION on each Insert statement which returns a new plan or nil, if it couldn't produce a plan with a' before a_i. If INSERTACTION does return a plan, for the statement Insert(a', a_i), then the plan will contain a' but other actions may have been added, removed, or re-arranged as well.

At this point we have a new set of plans. IMPROVE calls the simulator on each new plan to estimate the probability the plan has of achieving the goal. If any new plan has a better chance that the original plan, then the plan with the highest probability is chosen and the process is repeated on it. Otherwise, the original plan is returned.

Func. IMPROVE($\langle\{a_1, ..., a_n\}, G, \mathcal{I}, \mathcal{O}\rangle$, $\mathcal{QR}, k$)

- Set $\mathcal{A}' = \{a_1, ..., a_n\}$
- Repeat
  1. Simulate plan $\mathcal{A}'$ k times from state $\mathcal{I}$.
  2. Set *base* to be the number of simulations in which G holds in the final state in the trace.
  3. Call the data mining routines on the simulated traces to produce a set of statements of the form Prevent(a_i, o_i, k).
  4. For each statement Prevent(a_i, o_i, k) call SUGGESTCHANGES($\langle a_i, o_{i,k}\rangle$,$\mathcal{A}'$,$\mathcal{QR}$,$\mathcal{O}$), and collect all the results into a set of statements of the form Insert(a', a_j).
  5. Set $\mathcal{TEST} = \emptyset$.
     For each statement Insert(a', a_j) call INSERTACTION(a', a_j,$\{a_1, ..., a_n\}$,$\mathcal{I}$,G,$\mathcal{O}$). If INSERTACTION returns a plan,then add it to $\mathcal{TEST}$.
  6. For each plan $\mathcal{A}_i$ in $\mathcal{TEST}$, simulate the plan k times from state $\mathcal{I}$ and count the number of times that G is true in the final state.
  7. IF no plan $\mathcal{A}_i$ scored better than *base* in step 6, then RETURN($\mathcal{A}'$),
     ELSE set $\mathcal{A}'$ to the plan that scored the highest in step 6.

Figure 1: The IMPROVE algorithm

Experimental validation

We now describe experiments that measure IMPROVE's ability to increase the probability of goal satisfaction.

Domain

We tested our system in an evacuation domain consisting of 35 cities, 45 roads, 100 people, 2 buses, a helicopter and a variety of bus maintenance equipment. Buses and helicopter can move between cities (only by road for buses), pick up and drop off people and equipment and also apply a variety of bus maintenance actions. Buses can carry 25 people and helicopters can carry 1 person at a time. The goal is for all people to be in one specified city by a specified time.

The plan fails if a bus gets stuck or breaks down or a helicopter crashes. Buses can also malfunction in a variety of other ways, such as getting flat tires or overheating, that do not cause the plan to fail but can cause delays and make other malfunctions more likely. Each road has properties, such as being steep or bumpy, which make certain malfunctions more likely. For every malfunction, there is a maintenance action, such as adding coolant to prevent overheating, that will drastically reduce the probability that the malfunction will happen for a fixed amount of future driving. The maintenance actions, however, have preconditions of having one or two tools. The tools are at various cities on the map, and so adding a maintenance action can require a side trip to pick up the tool.

Finally, there is also a storm which makes certain malfunctions more likely. In each experiment, we randomly choose when and where the storm will hit and how fast it will move. Additionally, we vary the number and location of the people and tools, as well as the probability with which various combinations of bus malfunctions will cause the bus to break down.

We wrote a greedy scheduling algorithm that produces a plan to get all the people to the evacuation point. This algorithm ignores all the stochastic factors of the domain, such as weather and road conditions.

We also wrote a domain specific version of INSERTACTION. The primary use of it is to add bus mainte-

straightforward.

nance actions, which require various tools. First, the algorithm adds the maintenance action to the plan at the specified point. The algorithm then finds the city with a required tool that is closest to the bus's route prior to the maintenance action. The algorithm then adds a side trip to pick up the tool from that city. The algorithm repeats the process until the bus has all the tools required for the maintenance action.

Results

In each experiment, we generate a new random problem, which defines an initial state $\mathcal{I}$ and then call our scheduling routine to produce a sequence of move, load, and unload actions $a_1, ..., a_n$ to solve this problem. In all our trials, the sequence contained at least 250 actions. We then call IMPROVE with $a_1, ..., a_n$, which returns a new sequence of actions, typically with several additional move, pickup, and maintenance operations.

To evaluate IMPROVE, we use the simulator to compare the performance of original plan against the plan that IMPROVE returns. We use the simulator because the plans were too big to manage analytically.

As shown in table 1, our algorithm significantly improved the given plan. The initial plans achieved their goal about 82% of the time, and the improved plans achieved their goal about 98% of the time. On average, 11.7 alternative plans were simulated per invocation of IMPROVE.

Note that without the possibility of malfunctions, the plans produced by the greedy algorithm would succeed 100% of the time. The IMPROVE algorithm does not result in a more efficient schedule for evacuating people, which we consider the "easy" part of the domain in these experiments. The "hard" part of the domain is to deciding which maintenance actions to perform, and when to perform them. While the greedy algorithm can quickly generate a reasonable plan, it does not have any mechanism for reasoning about which maintenance actions to add.

To show that is not trivial to add maintenance actions to make the plan more robust against failure, we compared IMPROVE against two "straw men" algorithms. First, we tested the RANDOM algorithm which repeatedly chooses five random maintenance actions, uses the INSERTACTION algorithm to add them to the plan, simulates the five new plans, and selects the one with the best performance. RANDOM repeats the process until no change improves the performance of the plan. As shown, RANDOM only improved the plan slightly. Adding a maintenance action can make the plan worse because the side trips required to get the necessarily tools can lead to other malfunctions and also delay the plan so that the weather is worse.

	initial plan length	final plan length	initial success rate	final success rate	number plans tested
IMPROVE	272.3	278.9	0.82	0.98	11.7
Random	272.3	287.4	0.82	0.85	23.4
High	272.6	287.0	0.82	0.83	23.0

Table 1: Performance of IMPROVE, compared against two simple. Results averaged over 70 trials.

iteration	number of plans tested	improvement per plan	improvement of best plan
1	4.2	.073	.123
2	7.9	.016	.042
3	9.3	.013	.028
4	16	.012	.024

Table 2: Details of IMPROVE's performance.

We also evaluated an algorithm we called HIGH which works just like IMPROVE except that instead of data mining, it simply tries to prevent the five malfunctions that occurred most often, in each iteration. As shown, this algorithm also performed very badly demonstrating the need to focus repair efforts on the defects in the plan that are most responsible for failure.

Many of the actions IMPROVE added were in service of actions directly suggested by the data mining and qualitative reasoning components. There were at most four iterations of the simulation-mining-reasoning-adapting cycle. In each iteration, one maintenance operation is inserted into the plan, and then more actions are inserted to obtain the necessary tools for that action. Table 2 shows how many plans were considered per iteration of IMPROVE and the average and best improvement, on average, of those plans.

The data mining routine was written in C++ and run on an SGI with a 100MHz processor. The other functions were written in LISP and run on a variety of SPARC stations. Each test of IMPROVE took on average, based on the ten runs we measured, 75 minutes. About 34 minutes was spent in data mining, at a rate of 0.9 plans mined per CPU second. About 25 minutes was spent, on average, in simulation, at a rate of 3.2 plans simulated per CPU second. Much of the remaining time was spent generating the plan, doing bookkeeping, and file-based I/O between the processes.

Discussion

We now discuss some limitations of our current approach. Although the repairs considered by our algorithm can include many steps, they are all in service of a single action added to prevent some outcome from occurring. Many improvements, however, might require

several actions such as adding but later removing snow chains from a bus, or adding coolant several times to top off the radiator. Furthermore, we only mine the execution traces for reasons that cause failure, instead of also looking for patterns that predict success.

Furthermore, there may be a variety of opportunities for improving the plan that never arise in the executions of that plan, such as substituting one action for another, or using a helicopter rather than a bus to evacuate some city. Again, our approach will not currently improve the plan in this way because we only focus on preventing sequences that were common in the failed simulated traces and uncommon in the successful ones.

Finally, there are ways of improving a plan besides adding actions to it. For example, although we have not described it, we have explored techniques for improving a plan by re-ordering its actions. For example, in our domain, we might first evacuate the cities that the storm is going to hit first.

Related work

We now discuses several areas of related work.

Analyzing execution traces

Much work has been done on analyzing planning episodes, i.e. invocations of the planner, to improve future planning performance in terms of efficiency or quality. (e.g. (Minton 1990)). In contrast, we analyze execution traces to improve the quality of the plan at hand, not the planning process. Both types of techniques can be used in conjunction. But note that our approach is relevant only in domains with uncertainty, in which multiple executions of the same plan might differ from each other.

(McDermott 1994) describes a system in which a robot repeatedly simulates the execution of a plan while executing the plan, with hopes of finding a more robust alternative. Each error in the simulations is considered a bug and categorized into an extensive taxonomy of plan failure modes. This contrasts to our work, in which each simulation contains a great number of undesirable effects, such as buses getting flat tires or overheating, and our analysis attempts to discover important trends that distinguish successful from unsuccessful executions.

Plan repair

CHEF (Hammond 1990) is a case-based planning system addresses many issues. We focus on those most related to our work. CHEF composes a plan from plans in its memory, simulates it, and then analyzes the execution trace to improve the plan. CHEF employs a variety of repair strategies, including modifying the plan to avoid a side-effect of an action. Much of the similarities between CHEF and our IMPROVE algorithm are superficial, however, in that CHEF was motivated primarily by the idea of using episodic memory to perform planning. CHEF simulates a plan once and performs an analysis of the result using a deep causal model, and then applies repair-strategies that have worked previously. In contrast we simulate our plans hundreds of times and apply shallow, statistical methods to pinpoint defects and apply general plan adaptation techniques to repair plans.

The GORDIUS system (Simmons 1992) also applies repair strategies to a plan, by replacing an incorrect assumption which gave rise to the flaw. While our notion of plan repair is certainly no more rich than GORDIUS's, we focus on flaws that arise not from a defective assumption about the world but instead on probabilistic trends that arise frequently in the execution of the plan. One limitation of GORDIUS is that it can only resolve flaws that arise from a single mistaken assumption. We find problems that arise from sequences of defects that cause the problem.

The overall spirit of our approach resembles *robustification* in (Drummond & Bresina 1990). Here, a reactive plan is incrementally refined by detecting the state in which it is most likely to fail and then producing new instructions for that state. We use very different techniques than theirs. We use data mining to detect defects in the plan, rather than a set of heuristics to identify bad states. Furthermore, we focus on repairing a large, non-reactive plan to *avoid* the trouble, whereas Drummond and Bresnia enhance a reactive plan to get itself out of trouble once it has occurred.

(Alterman 1988) describes *run-time* repair of plans by, for example, replacing a step that just failed with a similar action. This is rather different that our approach of repairing plans in advance. We believe both techniques are necessary. In our domain, for example, there are maintenance actions which must be performed *before* the error occurs and require advanced planning to obtain the necessary tools.

Probabilistic planning

Classical planners have been extended to probabilistic domains (e.g. (Kushmerick, Hanks, & Weld 1995; Draper, Hanks, & Weld 1994; Goldman & Boddy 1994)). If this work scaled to the point where it could produce as large plans as anyone needed, then there would not be much need for our research on improving plans. However, both the complexity and the branching factor of probabilistic planning is much worse than classical planning, which itself has not produced very large plans. We believe domain-specific and collaborative techniques will yield large plans, which can be

used as the starting point for our IMPROVE algorithm.

Furthermore, this work addresses a crucial problem that probabilistic planners also face: deciding how to improve a plan that is not sufficiently reliable. In traditional planning, a plan has easily detectable flaws (i.e. unsatisfied or threatened preconditions). A planner needs to address each flaw and if there are no flaws, then the plan is sufficiently good. And it is these conditions that enable backchaining to work so effectively. In probabilistic planning, a plan might have no critical flaws, but still not be sufficiently good. In this case, there are many ways of improving the plan: for example, the planner can add an action that will increase the support of *any* of the preconditions in the plan. To our knowledge, no system uses analysis of the plan to guide the choice of what to work on next. In this paper, we have explored the use of data mining and qualitative reasoning techniques to focus attention of a planner on how to improve a plan.

Conclusions

This work was motivated by the desire to work with large plans. Our long term objective is to combine work on simulation, data mining, qualitative reasoning, and planning to produce tools for developing large, robust plans for complex domains. This paper has, hopefully, shown that such a system is possible and sketched out some of the key ideas.

We have shown that simulation, qualitative reasoning, and data mining are all well suited for working with large plans. Plan adaptation is not as well suited, but the complexity of improving plans is, at least, not exponential in the length of the plan to be improved.

We have described and implemented the IMPROVE algorithm which takes in a large plan, containing over 250 steps, and modifies it to improve the probability of goal satisfaction by over 15% in our experiments.

References

Agrawal, R., and Srikant, R. 1995. Mining sequential patterns. In *Intl. Conf. on Data Engg.*

Alterman, R. 1988. Adaptive planning. *Cognitive Science* 12:393–421.

Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under uncertainty: Structural assumptions and computational leverage. In *Proc. 2nd European Planning Workshop.*

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proc. 2nd Intl. Conf. AI Planning Systems.*

Drummond, M., and Bresina, J. 1990. Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction. In *Proc. 8th Nat. Conf. AI*, 138–144.

Ferguson, G.; Allen, J. F.; and Miller, B. 1996. TRAINS-95: Towards a mixed-initiative planning assistant. In *Proceedings of the Third International Conference on AI Planning Systems (AIPS-96).*

Goldman, R. P., and Boddy, M. S. 1994. Epsilon-safe planning. In *Proc. 10th Conf. Uncertainty in Artifical Intelligence.*

Goldman, R. P., and Boddy, M. S. 1996. Expressive Planning And Explicit Knowledge. In *Proc. 3rd Intl. Conf. AI Planning Systems.*

Hammond, K. 1989. *Case-Based Planning: Viewing Planning as a Memory Task.* Academic Press.

Hammond, K. 1990. Explaining and repairing plans that fail. *J. Artificial Intelligence* 45:173–228.

Hanks, S., and Weld, D. S. 1995. A domain-independent algorithm for plan adaptation. *J. Artificial Intelligence Research* 319–360.

Kambhampati, S., and Hendler, J. 1992. A validation structure based theory of plan modification and reuse. *J. Artificial Intelligence* 55:193–258.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An Algorithm for Probabilistic Planning. *J. Artificial Intelligence* 76:239–286.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proc. 9th Nat. Conf. AI*, 634–639.

McClave, J. T., and II, F. H. D. 1982. *Statistics.* San Francisco: Dellen Publishing Company.

McDermott, D. 1994. Improving robot plans during execution. In *Proc. 2nd Intl. Conf. AI Planning Systems*, 7–12.

Minton, S. 1990. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence* 42(2–3).

Nebel, B., and Koehler, J. 1995. Plan reuse versus plan generation: a theoretical and emperical analysis. *J. Artificial Intelligence* 76:427–454.

Simmons, R. 1988. A theory of debugging plans and interpretations. In *Proc. 7th Nat. Conf. AI*, 94–99.

Simmons, R. 1992. The roles of associational and causal reasoning in problem solving. *J. Artificial Intelligence* 159–208.

Veloso, M. 1994. Flexible strategy learning: Analogical replay of problem solving episodes. In *Proc. 12th Nat. Conf. AI*, 595–600.

Wellman, M. P. 1990. Fundamental concepts of qualitative probabilistic networks. *AI Magazine* 44:257–303.

Zaki, M. J.; Lesh, N.; and Ogihara, M. 1997. Sequence mining for plan failuresk. Technical Report URCS TR 671, University of Rochester.

Zaki, M. J. 1997. Fast mining of sequential patterns in very large databases. Technical Report URCS TR 668, University of Rochester.