Monitoring the Execution of Optimal Plans

Christian Fritz
Department of Computer Science,
University of Toronto,
Toronto, Ontario. CANADA.
fritz@cs.toronto.edu

Introduction

When executing plans, the world may evolve differently than predicted resulting in discrepancies between predicted and observed states of the world. These discrepancies can be caused by noisy sensors, unanticipated exogenous actions, or by inaccuracies in the predictive model used to generate the plan in the first place. Regardless of the cause, when a discrepancy is detected, it brings into question whether the plan being executed remains *valid* (i.e., projected to reach the goal) and where relevant, *optimal* with respect to some prescribed metric.

Effective execution monitoring requires a system to quickly discern between cases where a detected discrepancy is relevant to the successful execution of a plan and those cases where it is not. Algorithms dating as far back as PLANEX (1972, (Fikes, Hart, & Nilsson 1972)), Shakey the Robot's execution strategy, have exploited the idea of annotating plans with conditions that can be checked at execution time to confirm the continued *validity* of a sequential plan.

We are interested in the more difficult and unsolved problem of monitoring plan *optimality*. Our work is motivated in part by our practical experience with the fast-paced RoboCup domain where teams of robots play soccer against each other. In RoboCup, the state of the world is typically observed 10 times per second, each time raising the question of whether to continue with the current plan or to replan. Verifying plan validity and optimality must be done quickly because of the rapidly changing environment. Currently, there are no techniques to distinguish between relevant and irrelevant discrepancies (w.r.t. optimality), and so replanning is frequently done unnecessarily or discrepancies are ignored altogether, ultimately resulting in plan failure or sub-optimal performance.

We study the problem of monitoring the continued optimality of a plan. We build on the ideas exploited in algorithms for monitoring plan validity. To this end, we begin by formally characterizing common approaches to monitoring plan validity as found in the literature. We then outline how to generalize this characterization for monitoring plan optimality of deterministic plans, and later also of conditional plans in the context of decision-theoretic planning. We have implemented our algorithm for both, deterministic and conditional plans and show empirical results showing the potential computational savings resulting from our approach, compared to the only available alternative of replanning from scratch in the case of a discrepancy.

This extended abstract covers the intuitions behind (Fritz & McIlraith 2007a) and (Fritz & McIlraith 2007b), to be presented at ICAPS07 and one of its workshops, respectively.

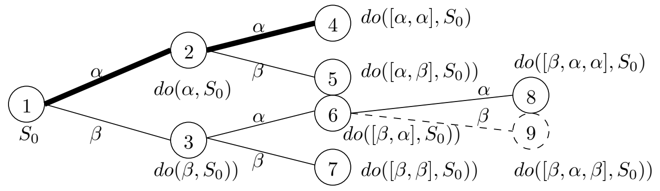
Monitoring Plan Validity

The available literature for execution monitoring is almost exclusively concerned with the problem of monitoring plan validity. This is the problem of deciding whether an executing plan will still reach the goal after unexpected events have happened, like actions failing to achieve some of their alleged effects, exogenous events that have changed the state of the world, etc.. Apart from this common limitation to plan validity, many if not all presented approaches, e.g. (Fikes, Hart, & Nilsson 1972; Wilkins 1985; Ambros-Ingerson & Steel 1988; Kambhampati 1990), also share the same underlying technique, and this while being seemingly unaware of that similarity and in particular without formalizing it.

We here characterize the approach formally as the regression of the goal and remaining preconditions over the remainder of the plan. The result of the regression provides the sufficient and necessary condition for plan validity with which the plan can be annotated for later verification during on-line execution.

The *regression* of a formula φ over an action α is another formula φ' that has to hold in order for φ to hold after executing α. We write $\varphi' = \mathcal{R}[\varphi, \alpha]$. Regression can be defined in several action languages. In our work we use the situation calculus (Reiter 2001), but since the availability of regression is the only requirement for the applicability of our approach, it can equally well be used with any other language for which regression can be defined, for instance STRIPS or ADL ((Pednault 1989)).

In the situation calculus the state of the world is represented by the sequence of actions that occurred since a distinguished and (partially) known initial situation S_0. Axiomatic knowledge about the effects of actions together with axioms describing what is true in the initial situation S_0 describe the (truth-)values of fluents in any particular situation. The function $do(a, s)$ denotes the situation reached after executing action a in situation s. We use $do([\alpha_1, \ldots, \alpha_n], s)$ to abbreviate $do(\alpha_n, do(\alpha_{n-1}, \ldots, do(\alpha_1, s) \ldots))$.



1
S_0
2
$do(\alpha, S_0)$
3
$do(\beta, S_0))$
4
$do([\alpha,\alpha], S_0)$
5
$do([\alpha,\beta], S_0))$
6
$do([\beta,\alpha], S_0))$
7
$do([\beta,\beta], S_0))$
8
$do([\beta,\alpha,\alpha], S_0)$
9
$do([\beta,\alpha,\beta], S_0))$
α
β

Figure 1: An example search tree. Dashed lines denote impossible actions, and $[\alpha, \alpha]$ is the optimal plan.

We denote the preconditions of an action α in a particular situation s by *Poss*(α, s) and use *Poss*$([\alpha_1, \ldots, \alpha_n], s)$ to abbreviate *Poss*$(\alpha_1, s) \wedge$ *Poss*$(\alpha_2, do(\alpha_1, s)) \ldots$ *Poss*$(\alpha_n, do([\alpha_1, \ldots, \alpha_{n-1}], s))$. [1]

Then, the above mentioned plan annotation can be formally defined as follows:

Definition 1 (Annotated Plan)**.** Given initial situation S_0, a sequential plan $\vec{\alpha} = [\alpha_1, \ldots, \alpha_m]$, and a goal formula $G(s)$, the corresponding annotated plan for $\vec{\alpha}$ is a sequence of tuples $\pi(\vec{\alpha}) = (G_1(s), \alpha_1), (G_2(s), \alpha_2), \ldots (G_m(s), \alpha_m)$ where

$$G_i(s) = \mathcal{R}_s\left[G(do([\alpha_i, \ldots, \alpha_m], s) \wedge Poss([\alpha_i, \ldots, \alpha_m], s)\right]$$

That is, in each step of the plan we annotate the regression of the goal and the remaining preconditions, where the regression is done over all remaining actions. $\mathcal{R}_s[\varphi(s')]$ denotes for $s' = do([a_1, \ldots, a_k], s)$ the repeated regression of φ over the actions a_i.

The annotation can be used during execution as follows, where we denote the actual situation encountered during execution, possibly deviating from the expected situation, by S^*.

Definition 2 (Algorithm for Monitoring Plan Validity)**.**

```
obtain $S^*$
while $(\neg G(S^*))$ {
    $\mathbf{i} = \mathbf{m}$
    obtain $S^*$
    while $(\neg G_{\mathbf{i}}(S^*))$  { $\mathbf{i} = \mathbf{i} - 1$}
    if $(\mathbf{i} > 0)$ then $execute\ \alpha_i$ else $replan$ }
```

This closely resembles the PLANEX execution strategy.

Monitoring Plan Optimality

With this formalization at hand we can now go on and address the more complex problem of monitoring optimality by generalizing above method. Given an optimal plan, where optimality is with respect to some user-defined metric, for instance minimizing costs, we are interested in asserting that an executing plan remains not only valid but also optimal. Space precludes presenting the technical details here (the interested reader is referred to (Fritz & McIlraith 2007a)). Instead we present the intuition, focusing on the difference to the case of validity monitoring. Consider the example search tree of Figure 1. Let $[\alpha, \alpha]$ be the plan found by an optimal, forward-search planner, and let optimality be expressed in terms of minimizing action costs. Then, the plan is optimal if all feasible alternatives reach the goal only with higher accumulated costs. This is typically asserted through the use of an admissible heuristic which provides a lower bound on the costs of reaching the goal from any given state, allowing to soundly prune branches from the search tree. In our example, the heuristic would have provided such bounds for the nodes 5, 7, and 8, and the these bounds together with the already accumulated costs of actions leading to these nodes, have to have exceeded those for node 4. Since $[\alpha, \alpha]$ is a plan, Node 4 also satisfies the goal.

Thus, intuitively in order to verify optimality on-line, we have to verify that the above circumstances still hold if, at the time the plan is to be executed, we find that the world is not in situation S_0 as we thought, but in an arbitrary, different situation S^*. To accomplish this, checking whether the plan is still optimal among all its alternatives, we follow the same idea as earlier for plan validity. We have to answer two questions: (a) what do we need to annotate the plan with? and (b) how can we use this annotation on-line to quickly distinguish between discrepancies that affect the optimality of the plan and those that don't?

(a) *Annotation:* Instead of annotating the plan with just the regression of the goal and the remaining preconditions, we now need to also include the regression of action costs and heuristic function. This is the first difference to the validity case. The second is that we have to do this not only for the plan itself, but also for all viable alternatives, since optimality is relative rather than absolute. It may be the case that an alternative has improved as a result of a discrepancy and also then we would like to reconsider which plan to take.

(b) *Execution Algorithm:* Once the plan has been annotated with the regression of all relevant entities (goal, preconditions, action costs, heuristic function in leaf nodes) we can again use this extra information on-line to quickly decide whether the current plan remains optimal. Here the main benefit of regressing the conditions to the current situation comes into play. Since, after regression, the values of all relevant entities are expressed in terms of the current situation (state), we can directly exploit knowledge about the actual discrepancy. If, say, situation S_i was expected but S^* observed, we can determine efficiently, which fluents (properties of the world) are different in these situations. Generally, discrepancies only affect a very small subset of fluents and this is what we can exploit. *Only conditions that mention any of the affected fluents need to be reevaluated, all others maintain their previous (truth-)value.* This allows for drastic computational savings, as we will see next.

Empirical Results

We were interested whether the approach was time-effective – whether the discrepancy-based incremental reevaluation could indeed be done more quickly than simply replanning when a discrepancy was detected.

To this end, we compared a preliminary implementation of our algorithm, which we call `monoplex`, to replanning from scratch on 9 different problems in the metric `TPP` domain of the 5th International Planning Competition. In each case, we solved the original planning problem, perturbed the

[1]For the situation calculus savvy: In this extended abstract we will abuse syntax by removing all reference to a basic action theory and in particular will write $\varphi(s)$ to express that φ holds in s, rather than the, correct, $\mathcal{D} \models \varphi(s)$, given an action theory $\mathcal{D}$.

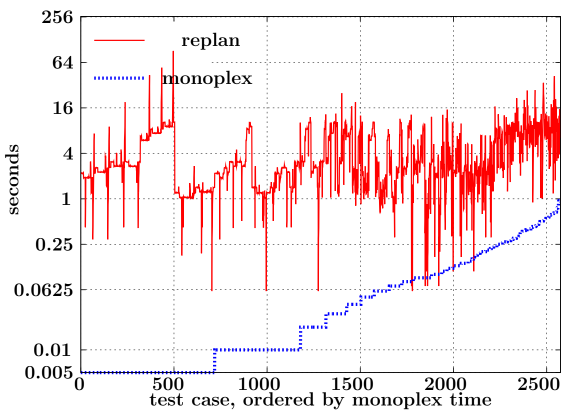


Figure 2: running time comparison (note the logarithmic scale)

state of the world by changing some fluents, and then ran both `monoplex` and replanning from scratch. To maximize objectivity, the perturbations were done systematically by multiplying the value of one of the numeric fluents by a factor between 0.5 and 1.5 (step-size 0.1). This resulted in a total of 2574 unique test cases. Figure 2 shows the performance of both approaches on a logarithmic scale (all experiments were run on an Intel Xeon, 3.6GHz, 1GB RAM). To enhance readability, we ordered the test cases by the running time of `monoplex`. The results show that although it is possible for `monoplex` to be slower than replanning (in 0.3% of the cases), it generally performs much better, resulting in a pleasing average speed-up of 209.12. In 1785 cases the current plan was asserted still optimal and therefore replanning unnecessary, in 105 it had become invalid. In 340 of the remaining 684 cases, replanning found the current plan to still be optimal. Sometimes the reevaluation of states and/or preconditions can even be entirely avoided, namely when the perturbation does not affect any relevant fluents. This happened in 545 cases and constitutes the greatest time savings potential, a result of our formal characterization of the situation-dependent relevance of fluents to the optimality of the plan.

Monitoring Policy Execution

The same technique can be used to monitor the execution of policies, represented as conditional plans from a known initial state, in domains with uncertain action outcomes (stochastic actions).

Markov Decision Processes (MDPs) are the de-facto standard for decision making under uncertainty. When MDPs have big or even infinite state spaces, standard solution techniques like value- or policy iteration become inefficient as their complexity depends on the size of the state space. Instead it is often beneficial to explore the state space from the initial state, if it is known, in a forward-search manner. This limits the search effort to the reachable subset of state space and can be done in an any-time fashion, increasing the search horizon as time permits. Since under these circumstances the robustness of a full policy, mapping each state of the world to a particular action, is lost, the need for execution monitoring again arises. During execution of the conditional

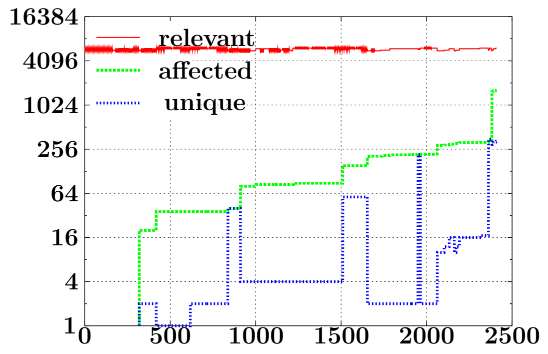


Figure 3: relevant, affected, and unique affected conditions

plan the agent may find itself in a state not considered during planning. This may be either because the search was incomplete, ignoring less likely action outcomes in the interest of time, or, again, because of exogenous events. Under these circumstances our method can again be useful.

Once more we formalized and implemented the annotation idea, that is we decided what the plan needed to be annotated with and how to use this on-line, and we tested it on a variation of the `TPP` domain with stochastic actions. Figure 3 shows three numbers of conditions (from top to bottom): the number of conditions relevant to the quality of the policy, the number of such conditions actually affected by the induced discrepancy, and the number of unique such conditions. The latter is the number of conditions that actually need to be reevaluated, using a cache to quickly obtain the evaluation result for copies of this condition in other parts of the annotation. The number of unique affected conditions is extremely low. On average this number was 2577.71 times lower than the number of relevant conditions. This motivates a patching approach over replanning.

References

Ambros-Ingerson, J., and Steel, S. 1988. Integrating planning, execution and monitoring. In *Proc. AAAI'88*, 83–88.

Fikes, R.; Hart, P.; and Nilsson, N. 1972. Learning and executing generalized robot plans. In *Artificial Intelligence*, volume 3, 251–288.

Fritz, C., and McIlraith, S. 2007a. Monitoring plan optimality during execution. In *Proc. ICAPS-07*. (to appear).

Fritz, C., and McIlraith, S. 2007b. Monitoring policy execution. In *Proc. of the 3rd Workshop on Planning and Plan Execution for Real-World Systems*. (to appear).

Kambhampati, S. 1990. A theory of plan modification. In *Proc. AAAI'90*, 176–182.

Pednault, E. 1989. ADL: exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR'89*, 324–332.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge, MA: MIT Press.

Wilkins, D. 1985. Recovering from execution errors in SIPE. In *Computational Intelligence*, volume 1, 33–45.