Monitoring Plan Optimality During Execution

Christian Fritz and **Sheila A. McIlraith**
Department of Computer Science, University of Toronto, Toronto, Ontario. Canada.
{fritz,sheila}@cs.toronto.edu

Abstract

A great deal of research has addressed the problem of generating optimal plans, but these plans are of limited use in circumstances where noisy sensors, unanticipated exogenous actions, or imperfect models result in discrepancies between predicted and observed states of the world during plan execution. Such discrepancies bring into question the continued optimality of the plan being executed and, according to current-day practice, are resolved by aborting the plan and replanning, often unnecessarily. In this paper we address the problem of monitoring the continued optimality of a given plan at execution time, in the face of such discrepancies. While replanning cannot be avoided when critical aspects of the environment change, our objective is to avoid replanning unnecessarily. We address the problem by building on practical approaches to monitoring plan *validity*. We begin by formalizing plan validity in the situation calculus and characterizing common approaches to monitoring plan validity. We then generalize this characterization to the notion of plan optimality and propose an algorithm that verifies continued plan optimality. We have implemented our algorithm and tested it on simulated execution failures in well-known planning domains. Experimental results yield a significant speed-up in performance over the alternative of replanning, clearly demonstrating the merit of our approach.

1 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. The task of execution monitoring is to monitor the execution of a plan, identify relevant discrepancies, and to take ameliorative action. In many cases the ameliorative action is to replan starting in the current state.

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 back as far as 1972 (e.g., PLANEX (Fikes, Hart, & Nilsson 1972)) 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.

Here, 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.

In this paper, we study the problem of monitoring the continued optimality of a plan. Our approach builds on ideas exploited in algorithms for monitoring plan validity. To this end, we begin by formalizing plan validity in the situation calculus, characterizing common approaches to monitoring plan validity found in the literature. We then generalize this characterization to the notion of plan optimality and propose an algorithm to monitor plan optimality at execution. Prior to execution time we annotate each step of our optimal plan by sufficient conditions for the optimality of the plan. These conditions correspond to the regression (Reiter 2001) of the evaluation function (cost + heuristic) used in planning over each alternative to the currently optimal plan. At execution time, when a discrepancy occurs, these conditions can be reevaluated much faster than replanning from scratch by exploiting knowledge about the specific discrepancy. We have implemented our algorithm and tested it on simulated execution failures in well-known planning domains. Experimental results yield an average speed-up in performance of two orders of magnitude over the alternative of replanning, clearly demonstrating the feasibility and benefit of the approach. Further, while our approach is described in the situation calculus, it is amenable to use with any action description language for which regression can be defined (e.g., STRIPS and ADL).

2 Preliminaries

The situation calculus is a logical language for specifying and reasoning about dynamical systems (Reiter 2001). In the situation calculus, the *state* of the world is expressed in terms of functions and relations (fluents, $\mathcal{F}$) relativized to a particular *situation* s, e.g., $F(\vec{x}, s)$. A situation s is a *history* of the primitive actions a, $a \in \mathcal{A}$ the set of all actions, performed from a distinguished initial situation S_0. The function $do(a, s)$ maps an action and a situation into a new situation thus inducing a tree of situations rooted in S_0. For readability, action and fluent arguments are generally suppressed. Also, $do(a_n, do(a_{n-1}, \dots do(a_1, s)))$ is abbreviated to $do([a_1, \dots, a_n], s)$ or $do(\vec{a}, s)$. In this paper we only consider finite sets of actions, $\mathcal{A}$.

A basic action theory in the situation calculus, $\mathcal{D}$, comprises four *domain-independent foundational axioms*, and a set of *domain-dependent axioms*. Details of the form of these axioms can be found in (Reiter 2001). We write $s \sqsubset s'$ to say that situation s precedes s' in the tree of situations. This is axiomatized in the foundational axioms. Included in the domain-dependent axioms are the following sets:

Initial State, S_0: a set of first-order sentences relativized to situation S_0, specifying what is true in the initial state.

Successor state axioms: provide a parsimonious representation of frame and effect axioms under an assumption of the completeness of the axiomatization. There is one successor state axiom for each fluent, F, of the form $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$, where $\Phi_F(\vec{x}, a, s)$ is a formula with free variables among $a, s, \vec{x}$. $\Phi_F(\vec{x}, a, s)$ characterizes the truth value of the fluent F in the situation $do(a, s)$ in terms of what is true in the current situation s.

Action precondition axioms: specify the conditions under which an action is possible to execute. There is one axiom for each action A, of the form $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$ where $\Pi_A(\vec{x}, s)$ is a formula with free variables among $\vec{x}, s$. We use the abbreviation $Poss([a_1, a_2, \dots, a_m], s) \stackrel{\text{def}}{=} Poss(a_1, s) \wedge Poss(a_2, do(a_1, s)) \wedge \dots \wedge Poss(a_m, do([a_1, \dots, a_{m-1}], s))$.

Regression

The *regression* of a formula ψ through an action a is a formula ψ' that holds prior to a being performed if and only if ψ holds after a is performed. In the situation calculus, regression is defined inductively using the successor state axiom for F as above:

$$
\begin{aligned}
&\mathcal{R}[F(\vec{x}, do(a, s))] = \Phi_F(\vec{x}, a, s)\\
&\mathcal{R}[\neg\psi] = \neg\mathcal{R}[\psi]\\
&\mathcal{R}[\psi_1 \wedge \psi_2] = \mathcal{R}[\psi_1] \wedge \mathcal{R}[\psi_2]\\
&\mathcal{R}[(\exists x)\psi] = (\exists x)\mathcal{R}[\psi]
\end{aligned}
$$

We denote the repeated regression of a formula $\psi(do(\vec{a}, s))$ back to a particular situation s by $\mathcal{R}_s$, e.g. $\mathcal{R}_s[\psi(do([a_1, a_2], s))] = \mathcal{R}[\mathcal{R}[\psi(do([a_1, a_2], s))]]$. Intuitively, the regression of a formula ψ over an action sequence $\vec{a}$ is the condition that has to hold for ψ to hold after executing $\vec{a}$. It is predominantly comprised of the fluents that play a role in the conditional effects of the actions in the sequence.

Regression is a purely syntactic operation. Nevertheless, it is often beneficial to simplify the resulting formula for later evaluation. Regression can be defined in many action specification languages. In STRIPS, regression of a literal l over an action a is defined based on the add and delete lists of a: $\mathcal{R}^{\text{STRIPS}}[l] = \text{FALSE}$ if $l \in \textsc{Del}(a)$ and $\{l\} \setminus \textsc{Add}(a)$ otherwise. Regression in ADL was defined in (Pednault 1989).

Notation: Lower case letters denote variables in the theory of the situation calculus, upper case letters denote constants. We use α and β to denote arbitrary but explicit actions. However, we use capital S to denote arbitrary but explicit situation terms, that is $S = do(\vec{\alpha}, S_0)$ for some explicit action sequence $\vec{\alpha}$. For instance, S_i for $i > 0$ denotes the situation expected during planning, and S^* the actual situation that arises during execution. Variables that appear free are implicitly universally quantified unless stated otherwise and $\varphi[x/y]$ denotes the substitution of all occurrences of x in formula φ with y.

3 Monitoring Plan Validity

In this section we formalize the notion of plan validity and provide an algorithm for monitoring plan validity. This provides the formal foundation for our approach to monitoring plan optimality described in Section 4.

Recall that a situation is simply a history of actions executed starting in S_0, e.g., $do([\alpha_1, \dots, \alpha_m], S_0)$.

Definition 1 (Plan Validity)**.** Given a basic action theory $\mathcal{D}$ and a goal formula $G(s)$, a plan $\vec{\alpha} = [\alpha_1, \dots, \alpha_m]$ is *valid* in situation S if $\mathcal{D} \models G(do(\vec{\alpha}, S)) \wedge Poss(\vec{\alpha}, S)$.

As such, a plan continues to be valid if, according to the action theory and the current situation, the precondition of every action in the plan will be satisfied, and at the end of plan execution, the goal is achieved.

A number of systems have been developed for monitoring plan validity (cf. Section 6) which all implicitly take the following similar approach. The planner annotates each step of the plan with a sufficient and necessary condition that confirms the validity of the plan. During plan execution these conditions are checked to determine whether plan execution should continue. We formally characterize the annotation and its semantics as goal regression. The provision of such a characterization enables its exploitation with other planners, such as very effective heuristic forward search planners.

Definition 2 (Annotated Plan)**.** Given initial situation S_0, a sequential plan $\vec{\alpha} = [\alpha_1, \dots, \alpha_m]$, and a goal formula $G(s)$, the corresponding annotated plan for $\vec{\alpha}$ is a sequence of tuples $\pi(\vec{\alpha}) = (G_1(s), \alpha_1), (G_2(s), \alpha_2), \dots (G_m(s), \alpha_m)$ with

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

I.e., each step is annotated with the regression of the goal and the preconditions over the remainder of the plan.

Proposition 1. A sequence of actions $\vec{\alpha}$ is a valid plan in situation S iff $\mathcal{D} \models \mathcal{R}_S[G(do(\vec{\alpha}, S)) \wedge Poss(\vec{\alpha}, S)]$.

Proof: The proof is by induction using the Regression Theorem (Reiter 2001, pp.65–66).

We can now provide an algorithm that characterizes the approach to monitoring plan validity described above. It is a generalization of the algorithm defined in (Fikes, Hart,

& Nilsson 1972). For the purposes of this paper, we assume that the "actual" situation of the world, S^*, is provided, having perhaps been generated using state estimation techniques or the like. The action theory $\mathcal{D}$ remains unchanged. For instance, S^* may differ from the expected situation $S_i = do([\alpha_1, \ldots, \alpha_{i-1}], S_0)$ by containing unanticipated exogenous actions, or variations of actions executed by the agent. It need not provide a *complete* description of the state of the world. The approach is applicable and even particularly interesting in cases of incomplete knowledge.[1]

Definition 3 (Algorithm for Monitoring Plan Validity)**.** With action theory $\mathcal{D}$ and annotated plan $\pi(\vec{\alpha})$ of length $\mathbf{m}$

obtain S^*
while $(\mathcal{D} \not\models G(S^*))$ {
  $\mathbf{i} = \mathbf{m}$; *obtain* S^*
  while $(\mathcal{D} \not\models G_{\mathbf{i}}(S^*))$ { $\mathbf{i} = \mathbf{i} - 1$}
  if $(\mathbf{i} > 0)$ **then** $execute\ \alpha_i$ **else** $replan$ }

The realization of the entailment of conditions ($\mathcal{D} \models \varphi(s)$) depends on the implemented action language. E.g., in STRIPS this is simple set inclusion of literals in the set describing the current state (roughly, $\varphi \in s$). For the situation calculus efficient Prolog implementations exist.

The correctness of this approach – only valid plans are continued and whenever a plan is still valid it is continued – is provided by the following theorem.

Theorem 1. The algorithm executes action α_i in situation S^* iff the remaining plan $[\alpha_i, \ldots, \alpha_m]$ is valid in situation S^* and i is the greatest index in $[1, m]$ with that property.

Proof: If α_i is executed, $\mathcal{D} \models G_i(S^*)$ holds and plan validity follows by Propositions 1. If there is a maximal index i such that $\mathcal{D} \models G_i(S^*)$ (plan valid by Prop. 1), α_i is executed.

4 Monitoring Plan Optimality

Now that we have a formal understanding of what is required to monitor plan validity, we exploit this to address the challenging problem of monitoring the continued optimality of a plan. Optimality appears in cases where the user not only specifies a goal to define valid plans, but also wishes to discriminate between all possible valid plans, by designating a measure of utility or preference over plans.

Given an optimal plan, our objective is to monitor its continued optimality, electing to replan only in those cases where continued execution of the plan will either not achieve the goal, or will do so sub-optimally. To this end, we extend the plan-annotation approach of Section 3 to monitor plan optimality. This is a critical problem for many real-world planning systems, and one that has been largely ignored.

We begin by defining plan optimality within our framework. Recall that $do(\vec{\alpha}, S)$ denotes the situation reached after performing the action sequence $\vec{\alpha}$ in situation S. The relation $\mathit{Pref}(s, s')$ is an abbreviation for a sentence in the situation calculus defining criteria under which s is more or equally preferred to s'.

[1]To better distinguish variables of the theory and meta-variables, used e.g. in pseudo code, we print the latter using bold face.

Definition 4 (Plan Optimality)**.** Given a basic action theory $\mathcal{D}$, a goal formula $G(s)$, and extra-logical relation $\mathit{Pref}(s, s')$, a plan $\vec{\alpha}$ is *optimal* in situation S if $\mathcal{D} \models G(do(\vec{\alpha}, S)) \wedge \mathit{Poss}(\vec{\alpha}, S)$ and there is no action sequence $\vec{\beta}$ such that $\mathcal{D} \models \mathit{Pref}(do(\vec{\beta}, S), do(\vec{\alpha}, S)) \wedge G(do(\vec{\beta}, S)) \wedge \mathit{Poss}(\vec{\beta}, S)$.

As such, a plan remains optimal in a new situation S^* when it remains valid and there exists no other valid plan that is preferred. Hence, to monitor plan optimality, we require two changes to our plan annotations: i) in addition to regressing the goal, we must regress the preference criteria to identify conditions that are necessary to enforce optimality and ii) since optimality is relative rather than absolute, we must annotate each plan step with the regression of the preferences over alternative plans as well.

This approach can be applied to a wide range of preference representation languages and planners. In this paper, we restrict our attention to preferences described by positive numeric action costs, and an A^* search based forward planner with an admissible evaluation function. To provide a formal characterization, we assume that the planning domain is encoded in a basic action theory $\mathcal{D}$. To keep the presentation simple, we also assume that the goal is a fluent $G(s)$ and can only be established by a particular action finish, contained in the action theory. Any planning problem can naturally be translated to conform to this by defining the preconditions of the finish action corresponding to the goal of the original planning problem.

Recall that in A^* search, the evaluation function is the sum of the heuristic function and the accumulated action costs. We denote functions in relational form. In situation s the accumulated cost c of actions performed since S_0 is denoted by the relational fluent $\mathit{Cost}(c, s)$. It is specified incrementally in the successor state axioms of the fluent $\mathit{Cost}(c, s)$. For instance, $\mathit{Cost}(c, do(a, s)) \equiv \mathit{Cost}(c', s) \wedge (a = driveTo(paris) \wedge c = c' + driveCostToParis) \vee ((\nexists x).a = driveTo(x) \wedge c = c')$. The search heuristic yielding value h in s is denoted by $\mathit{Heu}(h, s)$. We understand this to denote a formula provided by the user, for instance of the form $\mathit{Heu}(h, s) \stackrel{\text{def}}{=} (\phi_1(s) \wedge h = h_1) \vee (\phi_2(s) \wedge h = h_2) \vee (\phi_n(s) \wedge h = h_n)$, where the ϕ_i partition state space. Correspondingly our A^* evaluation relation is specified as follows:

$$\mathit{Value}(v, s) \stackrel{\text{def}}{=} (\exists h, c).\mathit{Heu}(h, s) \wedge \mathit{Cost}(c, s) \wedge v = h + c.$$

The preference relation for our A^* search is defined as:

$$\mathit{Pref}_{A^*}(s_1, s_2) \stackrel{\text{def}}{=} (\exists v_1, v_2).\, \mathit{Value}(v_1, s_1) \wedge \mathit{Value}(v_2, s_2) \wedge v_1 < v_2.$$

By the definition of admissibility we know that it is non-decreasing, that is if s_1 is preferred to s_2, then no successor of s_2 is preferred to s_1:

$$\mathcal{D} \models \mathit{Pref}_{A^*}(s_1, s_2) \supset (\nexists s_2').s_2 \sqsubset s_2' \wedge \mathit{Pref}_{A^*}(s_2', s_1). \qquad (1)$$

In A^* search, nodes that have been seen but not explored are kept in the so-called *open list*. In planning, the open list is initialized to the initial situation. Planning proceeds by repeatedly removing the most preferred situation in the open list and inserting its feasible successor situations. Search terminates successfully when this first element, $do(\vec{\alpha}, S_0)$, satisfies the goal, $\mathcal{D} \models G(do(\vec{\alpha}, S_0))$. The plan described by this, $\vec{\alpha}$, is always optimal because any alternative plan would be a continuation of one of the partial plans in the open list, but by Equation (1) and the fact that $do(\vec{\alpha}, S_0)$ is preferred to any other element in the open list, no such plan could be preferred to $\vec{\alpha}$.

It follows that to determine the continued optimality of plan $\vec{\alpha}$ in a new situation S^* (replacing S_0), it is *sufficient* to check that $do(\vec{\alpha}, S^*)$ is still most preferred with respect to the open list when search terminated. We may, however, need to extend this list first because action sequences previously predicted to be impossible (in S_i) may now be possible (in S^*) and the current plan, $\vec{\alpha}$, must also be preferred to these plans. But even then this is not a *necessary* condition. It may be the case that another element $do(\vec{\beta}, S^*)$ in the (revised) open list is preferred to $do(\vec{\alpha}, S^*)$, but if $do(\vec{\beta}, S^*)$ doesn't satisfy the goal, $do(\vec{\alpha}, S^*)$ may still turn out to be optimal. This could be the case if (i) no successor of $do(\vec{\beta}, S^*)$ satisfies the goal, or (ii) there are successors that satisfy the goal but they are less preferred than $do(\vec{\alpha}, S^*)$. This can occur because the heuristic can, and generally does, increase. Formally, $\mathcal{D} \models \mathit{Pref}_{A^*}(s_1, s_2) \not\supset (\not\exists s_1').s_1 \sqsubset s_1' \wedge \mathit{Pref}_{A^*}(s_2, s_1')$. Neither of these issues, can be resolved without further, time consuming, planning. For this reason, we limit ourselves to a tight sufficient condition, defined in terms of the fringe.

Definition 5 (Fringe)**.** Given an action theory $\mathcal{D}$ and a goal formula $G(s)$, a *fringe* of situation S is a set of situations $L = \{S_1, \ldots, S_n\}$, such that for each $S_j \in L$: (i) S_j is a descendant of S, (i.e., $S \sqsubset S_j$), (ii) there is no $S' \in L$ s.t. $S_j \sqsubset S'$, (iii) for any S'' if $do(\alpha, S'') \in L$ for some $\alpha \in \mathcal{A}$ then also $do(\alpha', S'') \in L$ for every other $\alpha' \in \mathcal{A}$, where $\mathcal{A}$ is the set of actions in the theory, and (iv) there is no S''', $S''' \sqsubset S_j$ such that $\mathcal{D} \models G(S''')$.

Note the similarity between fringe and open list. The main difference is that a fringe can include infeasible situations. An important property of fringes is that any plan has exactly one prefix in any given fringe. This exhaustive character of fringes allows us to make optimality guarantees in conjunction with a heuristic function.

Theorem 2 (Sufficiency)**.** Let $\mathcal{D}$ be an action theory, $G(s)$ a goal, S a situation, and $\vec{\alpha}$ a valid plan for $G(s)$ in S. If there is a fringe L of S such that for every $do(\vec{\beta}, S) \in L$, $\mathcal{D} \models (\exists v_a, v_b).\mathcal{R}_S[\mathit{Poss}(\vec{\beta}, S) \wedge \mathit{Value}(v_a, do(\vec{\alpha}, S)) \wedge \mathit{Value}(v_b, do(\vec{\beta}, S))] \supset v_b \geq v_a$, then $\vec{\alpha}$ is optimal in S.[2]

Proof: Assume to the contrary that $\vec{\alpha}$ is not optimal in S, then there is a $s_b = do(\vec{\beta}, S)$ s.t. $\mathcal{D} \models (\exists v_a, v_b).\mathit{Value}(v_a, do(\vec{\alpha}, S)) \wedge \mathit{Value}(v_b, s_b) \wedge v_b < v_a$ and $\mathcal{D} \models G(s_b) \wedge \mathit{Poss}(\vec{\beta}, S)$. By definition of a fringe, s_b is either in L or is a predecessor or a descendant of some element in L. Since no element in L is preferred to $\vec{\alpha}$ and no predecessor of any element in L satisfies the goal, s_b has to be a descendant of some element in L. But by definition of admissibility (Equation (1)), no such descendant can be preferred to $\vec{\alpha}$. $\square$

This theorem establishes sufficient conditions for determining continued plan optimality. We must now translate these conditions into plan annotations that can be quickly checked during plan execution.

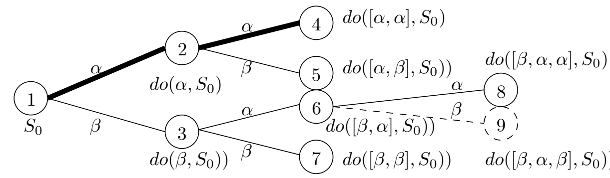


Figure 1: An example search tree. Dashed lines denote impossible actions, and $[\alpha, \alpha]$ is the optimal plan.

4.1 Annotation

Plan annotations can be computed at planning time by our A^* search forward planner. We assume our planner will output an optimal plan $\vec{\alpha}$, the open list O that remained when planning terminated (e.g. nodes 5, 7, and 8 in Figure 1), and a list O^- containing those situation terms found to be infeasible during planning. This is a list of situations $do(\alpha^-, S)$ such that α^- is not possible in situation S, i.e. $\mathcal{D} \models \neg \mathit{Poss}(\alpha^-, S)$ (cf. node 9 in the figure). The union $\{do(\vec{\alpha}, S_0)\} \cup O \cup O^-$ is a fringe of S_0. Since monitoring optimality must be done relative to alternatives, each step of the optimal plan is annotated with the conditions that confirm its continued validity, as well as a list of alternative plans and their corresponding predicted evaluation function values relativized to that plan step. Note that in the text that follows it is the description of the annotation, and not the annotation itself, that is defined in the situation calculus.

Definition 6 (Annotated Plan (Optimality))**.** Given the initial situation S_0, the goal formula $G(s)$, the evaluation relation $\mathit{Value}(v, s)$, an optimal sequential plan $\vec{\alpha} = [\alpha_1, \ldots, \alpha_m]$, the open list O, and the list of infeasible situations O^-, the corresponding annotated plan for $\vec{\alpha}$ is a sequence of tuples

$$\pi(\vec{\alpha}) = (G_1(s), V_1, \mathit{Alt}_1, \alpha_1), \ldots, (G_m(s), V_m, \mathit{Alt}_m, \alpha_m)$$

where $G_i(s)$ is as defined in Definition 2 and V_i and Alt_i are defined as follows, with $\vec{\alpha}_i = [\alpha_i, \ldots, \alpha_m]$ the remaining plan, and $S_i = do([\alpha_1, \ldots, \alpha_{i-1}], S_0)$:

$V_i = (\mathcal{R}_s[\mathit{Value}(v, do(\vec{\alpha}_i, s))], \mathbf{v_i})$,
with $\mathbf{v_i} \in I\!R$ such that $\mathcal{D} \models \mathit{Value}(\mathbf{v_i}, do(\vec{\alpha}_i, S_i))$

$\mathit{Alt}_i = \{\mathit{Alt}_{i_1}, \mathit{Alt}_{i_2}, \ldots, \mathit{Alt}_{i_n}\}$ containing all tuples

$\mathit{Alt}_{i_j} = (\vec{\beta}_{i_j}, \phi_{i_j}^{\mathit{Poss}}, \mathbf{p_{i_j}}, \phi_{i_j}^{\mathit{Value}}, \mathbf{v_{i_j}})$

such that $do(\vec{\beta}_{i_j}, S_i) \in O \cup O^-$ and where:

$\phi_{i_j}^{\mathit{Poss}} = \mathcal{R}_s[\mathit{Poss}(\vec{\beta}_{i_j}, s)]$, variable in s,

$\mathbf{p_{i_j}} = 1$ if $do(\vec{\beta}_{i_j}, S_i) \in O$ and $\mathbf{p_{i_j}} = 0$ if $do(\vec{\beta}_{i_j}, S_i) \in O^-$,

$\phi_{i_j}^{\mathit{Value}} = \mathcal{R}_s[\mathit{Value}(v, do(\vec{\beta}_{i_j}, s))]$, variable in v, s, and

$\mathbf{v_{i_j}} \in I\!R$ such that $\mathcal{D} \models \mathit{Value}(\mathbf{v_{i_j}}, do(\vec{\beta}_{i_j}, S_i))$.

For plan step i, V_i contains the regression of the evaluation relation over the remaining plan $\vec{\alpha}_i$ and its value w.r.t. the expected situation S_i. Alt_i represents the list of alternative action sequences $\vec{\beta}_{i_j}$ to $\vec{\alpha}_i$, together with their respective regression of the evaluation relation and particular value in S_i ($\mathbf{v_{i_j}}$), and regressed preconditions and their truth value ($\mathbf{p_{i_j}}$, represented as 0 or 1) in S_i. For example, in

[2] Since $\vec{\beta}$ is a particular, known action sequence, regressing over it is not a problem and our abbreviation $\mathit{Poss}(\vec{\beta}, S)$ is well defined.

the search tree of Figure 1, $\vec{\alpha} = [\alpha, \alpha]$ is the optimal plan, Alt_1 (cf. node 1) contains tuples for the action sequences $[\alpha, \beta], [\beta, \alpha, \alpha], [\beta, \alpha, \beta], [\beta, \beta]$, and Alt_2 (cf. node 2) contains only one tuple for $[\beta]$. Intuitively, the regression of the evaluation relation over a sequence of actions $\vec{\alpha}$ describes in terms of the current situation, the value the evaluation relation will take after performing $\vec{\alpha}$. As an example, consider the task of delivering a package to a location using a truck. Assume the heuristic yields a value $v = 0$ when the truck has the package loaded and is at the right location, and $v = 1$ otherwise. Then, regressing the heuristic through the action of driving the truck to the right location would yield a formula stating "$v = 0$ *if the package is on the truck, and* $v = 1$ *otherwise*" (ignoring action costs for now).

The *key* benefit of our approach comes from regressing conditions to the situations where they are relevant. Consequently when a discrepancy is detected during plan execution and the world is in situation S^* rather than predicted situation S_i, the monitor can determine the difference between these situations and limit computation to reevaluating those conditions that are affected by the discrepancy. This is only possible because regression has enabled definition of relevant conditions with respect to the situation before executing the remainder of the plan or any alternative.

4.2 Execution Monitoring

Assume we have an optimal plan $\vec{\alpha} = [\alpha_1, \ldots, \alpha_m]$, and that we have executed $[\alpha_1, \ldots, \alpha_{i-1}]$ for some $i \leq m$ and thus expect to be in situation $S_i = do([\alpha_1, \ldots, \alpha_{i-1}], S_0)$. Given the situation estimate S^* and the annotated plan as described in Definition 6, our task is to decide whether execution of the remainder of the plan, $\vec{\alpha}_i = [\alpha_i, \ldots, \alpha_m]$, is still optimal. We will do this by reevaluating all relevant conditions in S^* to verify that the current plan is still valid and achieves maximal value among all alternatives.

Recall the representation of alternative plans $\vec{\beta}$ in Alt_i, containing the regressed preconditions, evaluation relation, and their respective 'values' during planning (i.e. w.r.t. S_i). Also recall that $V_i = (\phi^{Value}, \mathbf{v_i})$ where ϕ^{Value} is a formula over variables v and s, denoting the regression of the evaluation relation over $\vec{\alpha}_i$. A naive algorithm for monitoring plan optimality at execution would be as follows:

Definition 7 (Naive Optimality Monitoring Algorithm)**.** Given the annotated plan $\pi(\vec{\alpha}) = (G_1(s), V_1, Alt_1, \alpha_1), \ldots, (G_m(s), V_m, Alt_m, \alpha_m)$:

$\mathbf{i} = 1$
while $(\mathbf{i} \leq \mathbf{m})$ {
  obtain S^*
  $(\phi^{Value}, \mathbf{v_i}) = V_i$
  if $(\mathcal{D} \models G_\mathbf{i}(S^*)$ and $\exists \mathbf{v_a} \in I\!R$ s.t. $\mathcal{D} \models \phi^{Value}[s/S^*, v/\mathbf{v_a}]$
    and $\forall (\beta, \phi_\beta^{Poss}, \mathbf{p}_\beta, \phi_\beta^{Value}, \mathbf{v}_\beta) \in Alt_i, \exists \mathbf{v_b} \in I\!R$ s.t.
    $\mathcal{D} \models \phi_\beta^{Poss}[s/S^*] \wedge \phi_\beta^{Value}[s/S^*, v/\mathbf{v_b}] \wedge \mathbf{v_b} \geq \mathbf{v_a})$
  then { *execute* α_i; $\mathbf{i} = \mathbf{i} + 1$; }
  else *replan* }

This prescribes to continue execution as long as no feasible element from the list of alternatives achieves a better value in S^* than the current plan. The time cost of this algorithm is greatly determined by the computation of the condition as it reevaluates all annotated formulae anew in S^*. We can significantly reduce this time by only reevaluating those conditions that may have been affected by the discrepancy between the predicted situation S_i and actual situation S^*.

Let $\Delta_F(S_i, S^*)$ be the set of fluents whose truth values differ between S_i and S^*, i.e. $\Delta_F(S_i, S^*) = \{F(\vec{X}) \mid F \in \mathcal{F}$ and $\mathcal{D} \models F(\vec{X}, S_i) \not\equiv F(\vec{X}, S^*)\}$, with $\mathcal{F}$ the set of fluents. Only conditions mentioning any of these fluents need to be reevaluated, all others remain unaffected by the discrepancy. Let *fluents*(φ) denote the set of all fluents occurring in φ. An improved algorithm for monitoring plan optimality during execution is as follows:

Definition 8 (Monoplex)**.** Given the annotated plan $\pi(\vec{\alpha}) = (G_1(s), V_1, Alt_1, \alpha_1), \ldots, (G_m(s), V_m, Alt_m, \alpha_m)$ $monoplex(\mathcal{D}, \pi(\vec{\alpha}))$:

1 $\mathbf{i} = 1$
2 **while** $(\mathbf{i} \leq \mathbf{m})$ {
3   *obtain* S^*; *generate* $\Delta_F(S_i, S^*)$;
4   **if** $(\mathcal{D} \models G_\mathbf{i}(S^*))$ **then** {   // plan remains valid
5     $(\phi^{Value}, \mathbf{v_i}) = V_i$
6   **if** (*fluents*$(\phi^{Value}) \cap \Delta_F(S_i, S^*) \neq \emptyset$) **then**
7   { $\mathbf{v_i} = \mathbf{v_i^{new}}$ *s.t.* $\mathcal{D} \models \phi^{Value}[s/S^*, v/\mathbf{v_i^{new}}]$ }
8   **foreach** $((\beta, \phi_\beta^{Poss}, \mathbf{p}_\beta, \phi_\beta^{Value}, \mathbf{v}_\beta) \in Alt_i)$ {
9   **if** (*fluents*$(\phi_\beta^{Poss}) \cap \Delta_F(S_i, S^*) \neq \emptyset$) **then**
10   { **if** $(\mathcal{D} \models \phi_\beta^{Poss}[s/S^*])$ **then** $\mathbf{p}_\beta = 1$ **else** $\mathbf{p}_\beta = 0$ }
11   **if** ($\mathbf{p}_\beta == 1 \wedge$ *fluents*$(\phi_\beta^{Value}) \cap \Delta_F(S_i, S^*) \neq \emptyset$) **then**
12   { $\mathbf{v}_\beta = \mathbf{v}_\beta^{\mathbf{new}}$ *s.t.* $\mathcal{D} \models \phi_\beta^{Value}[s/S^*, v/\mathbf{v}_\beta^{\mathbf{new}}]$ }
13   **if** $(\mathbf{p}_\beta == 1 \wedge \mathbf{v}_\beta < \mathbf{v_i})$ **then**
14   { *replan* } }   // plan may be sub-optimal, replan
15   *execute* α_i; $\mathbf{i} = \mathbf{i}+1$ // plan remains optimal, continue
17 **else** *replan* } }   // plan is invalid

While the plan has not been executed to completion (line 2), the algorithm does the following:

line 4: it checks validity;

lines 5–7: if the regression of the evaluation function over the plan (ϕ^{Value}) mentions any affected fluent, it is reevaluated, obtaining new value $\mathbf{v_i^{new}}$;

lines 8–14: the algorithm then checks for each alternative $\vec{\beta}$ at this point of plan execution: (in lines 9,10) whether its preconditions are affected and need to be reevaluated, (in lines 11,12) whether its value is affected and needs to be reevaluated, and (in line 13) whether this alternative is now possible and better than the current plan. If an alternative has become better, the algorithm calls for replanning. Otherwise the next action of the plan is executed.

Intuitively, the **foreach** loop revises relevant values – the truth of preconditions and the value of the evaluation function – generated for S_i with respect to the actual situation S^*, aborting execution when a viable and superior alternative is found. Line 12 is most crucial: Here the regression of the evaluation relation over alternative plan $\vec{\beta}$ is reevaluated with respect to the actual current situation S^*, yielding a new value $\mathbf{v}_\beta^{\mathbf{new}}$ for the evaluation relation (cf. Alt_{i_j} in Definition 6). This reevaluation only occurs if the regression result (formula) contains any of the affected fluents. Otherwise the value hasn't changed as a result of the discrepancy. Again, the realization of the entailment ($\mathcal{D} \models \varphi(s)$) depends on the implemented action language. The method is in particular not reliant on the situation calculus and can be used with any action language for which regression can be defined.

Theorem 3 (Correctness)**.** Whenever $monoplex$ executes the next step of the plan (*execute* α_i), the remaining plan $\alpha_i, \ldots, \alpha_m$ is optimal in the actual current situation S^*.

Proof Sketch: The plan $\vec{\alpha}$ and the alternatives $\vec{\beta}_{i_j}$'s in Alt_i describe a fringe of S_i. The **foreach** loop provides for all of the latter that v_{i_j} is such that $\mathcal{D} \models \mathit{Value}(v_{i_j}, do(\vec{\beta}_{i_j}, S^*))$ and that $p_{i_j} = 1$ if $\mathcal{D} \models \mathit{Poss}(\vec{\beta}_{i_j}, S^*)$ and $p_{i_j} = 0$ otherwise (Regression Theorem), and similarly the code of lines 5–7 updates v_i as necessary, i.e. s.t. $\mathcal{D} \models \mathit{Value}(v_i, do(\vec{\alpha}, S^*))$. The theorem then follows from Theorem 2. □

Exploiting the Search Tree Working with the fringe directly causes a lot of redundant work. This is because many alternative action sequences share the same prefix and so the costs and preconditions of these prefixes are annotated and potentially reevaluated multiple times. We can avoid this by exploiting the search tree structure in both the annotation and the algorithm. The details of this method are much more complicated than the above description based on the list of alternatives, which is why we chose to omit these details in this paper. Our implementation, however, uses the improved search tree method. Formally the search tree method is equivalent to the described method with respect to making the decision between continuing and aborting/replanning.

4.3 An Illustrative Example

Consider the following simplified example from the TPP domain, where an agent drives to various markets to purchase goods which she then brings to the depot. For simplicity, assume there is only one kind of good, two markets, and the following fluents: in situation s, $At(l, s)$ denotes the current location l, $Totalcost(t, s)$ denotes the accumulated costs t of all actions since S_0, $Request(r, s)$ represents the number r requested of the good, $Price(p, m, s)$ denotes the price p of the good on market m, and $DriveCost(c, src, dest, s)$ the cost c of driving from src to $dest$. Let there be two actions: $drive(dest)$ moves the agent from the current location to $dest$, and $buyAllNeeded$ purchases the requested number of goods. Assume, the planner has determined the plan $\vec{\alpha} = [drive(Market1), buyAllNeeded, drive(Depot)]$ to be optimal, but has as well considered $\vec{\beta} = [drive(Market2), buyAllNeeded, drive(Depot)]$ as one alternative among others. To shorten presentation, we ignore the heuristic here, i.e. assume uniform cost search ($h = 0$). Then

$$\begin{aligned} V_1 = (\ &(\exists t, l, c_1, p, r, c_2).Totalcost(t, s) \wedge At(l, s) \wedge \\ &DriveCost(c_1, l, Market1, s) \wedge Price(p, Market1, s) \wedge \\ &Requested(r, s) \wedge DriveCost(c_2, Market1, depot, s) \wedge \\ &v = t + c_1 + (r \cdot p) + c_2,\ \mathbf{v_1}\ ) \end{aligned}$$

and similarly $Alt_{1_1} = (\vec{\beta}, \phi_{1_1}^{\mathit{Poss}}, \mathbf{p_{1_1}}, \phi_{1_1}^{\mathit{Value}}, \mathbf{v_{1_1}})$ where, very similar to above, $\phi_{1_1}^{\mathit{Value}} = (\exists t, l, c_1, p, r, c_2).Totalcost(t, s) \wedge At(l, s) \wedge DriveCost(c_1, l, Market2, s) \wedge Price(p, Market2, s) \wedge Requested(r, s) \wedge DriveCost(c_2, Market2, depot, s) \wedge v = t + c_1 + (r \cdot p) + c_2$ where we ignore preconditions for simplicity and $\mathbf{v_1}$ and $\mathbf{v_{1_1}}$ are the respective values of the cost function for the plan and the alternative with respect to the situation where we expect to execute this plan, S_0.

Let's assume that even before the execution of the plan begins, a discrepancy in the form of an exogenous action e happens, putting us in situation $S^* = do(e, S_0)$ instead of S_0. The question is, whether $\vec{\alpha}$ is still optimal and in particular still better than $\vec{\beta}$. This clearly depends on the effects of e. If e does not affect any of the fluents occurring in above annotated formulae, it can be ignored, the plan is guaranteed to remain optimal. This would, for instance, be the case when e represents the event of a price change on a market not considered, as that price would not find its way into the regressed formulae which only mention relevant fluents.

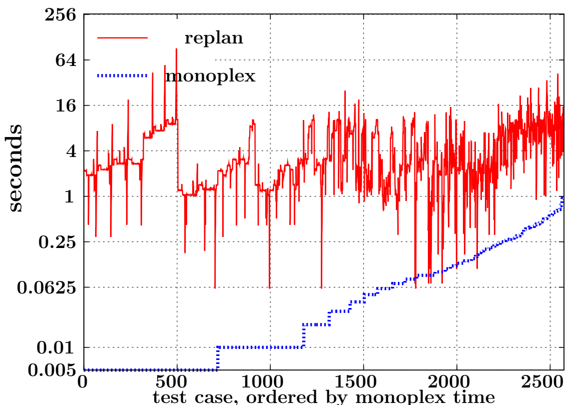
But even when e affects a relevant fluent, replanning may not be necessary. Assume, for instance, that e represents the event of an increased demand, that is, increasing the value r of $Request(r)$, formally $\mathcal{D} \models Request(r, S^*) > Request(r, S_0)$. Then $\Delta_F(S_0, S^*) = \{Request(r)\}$ and we need to reevaluate the annotated conditions, as $\vec{\beta}$ may have become superior. This could be the case, for instance, if the drive cost to $Market1$ is lower than to $Market2$, but the price at this market is higher. Then, a higher demand may make $Market2$ favorable, as the drive cost is compensated more than before by the lower price. This can quickly be determined by reevaluating the annotated conditions, obtaining new values $\mathbf{v_1}$ for $\vec{\alpha}$ and $\mathbf{v_{1_1}}$ for $\vec{\beta}$. If $\mathbf{v_{1_1}} < \mathbf{v_1}$ we have to replan, otherwise the plan is guaranteed to remain optimal.

5 Empirical Results

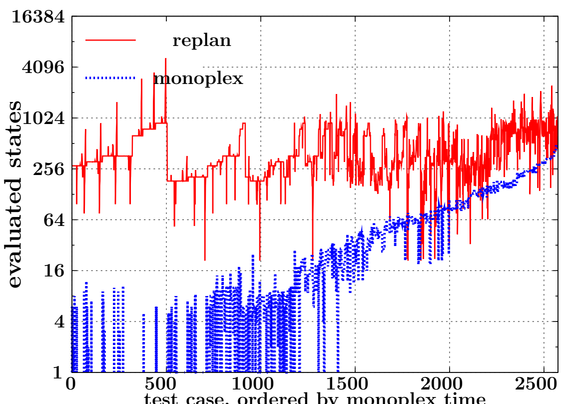
We have proven that our approach establishes conditions under which plan optimality persists in a situation. We were interested in determining whether the approach was time-effective – whether the discrepancy-based incremental reevaluation could indeed be done more quickly than simply replanning when a discrepancy was detected. As noted in the introduction, continuous replanning has already been determined to be ineffective in highly-dynamic domains.

To this end, we compared a preliminary implementation of our `monoplex` algorithm to replanning from scratch on 9 different problems in the metric `TPP` domain and two different problems of the `open stacks` domain with time (durative actions) of the 5^{th} International Planning Competition. In each case, we solved the original planning problem, perturbed the 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), or by changing the truth value of a Boolean fluent.

TPP: In the `TPP` domain 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 determining factors for the running time (cf. Figure 2a) are predominantly the number of states for which the evaluation function had to be reevaluated (2b),



(a) running time



(b) number of evaluated states

Figure 2: `TPP domain` (note the logarithmic scale)

and the number of reevaluated action preconditions. The discrepancy guidance of `monoplex` found that on average only 1.18% of all preconditions evaluated during replanning were affected by the discrepancy and had to actually be reevaluated.

The results show that although it is possible for `monoplex` to be slower than replanning (in 8 out of 2574 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. Notice in (b) and (c) that sometimes the reevaluation of states and/or preconditions can be entirely avoided, namely when the perturbation does not affect any relevant fluents. This happened 545 times 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.

Open stacks: Less drastic, but similar in nature are the result for the `open stacks` domain, with an average speed-up of 139.84. Figure 3 shows the fraction of running time required by `monoplex` compared to replanning for two different planning heuristics. In order to investigate the influence of the applied heuristic function we ran the same set of experiments with two different heuristics, 'A' and 'B',

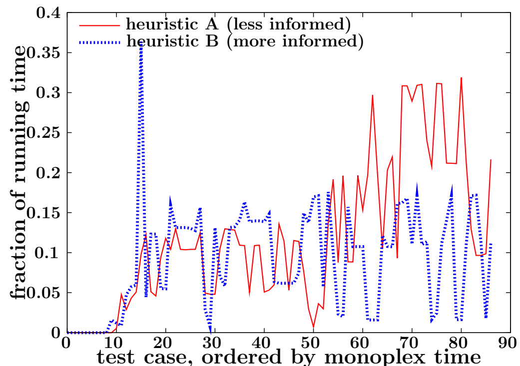


Figure 3: Fraction of running time in `open stacks` domain.

where 'B' is more informed than 'A' (cf. the respective lines in the figure). Again comparing `monoplex` to replanning, we note that the performance of `monoplex` improves with the use of a more informed heuristic and that it, in particular, preserves its superiority over replanning. This meets the intuition that, like the planner, `monoplex` benefits from a more focused search, resulting in a smaller search tree.

6 Related Work

The use of some form of plan annotation to monitor the continued validity of a plan during execution has been exploited in a number of systems (e.g., (Fikes, Hart, & Nilsson 1972; Wilkins 1985; Ambros-Ingerson & Steel 1988; Kambhampati 1990)), however none identified the formal foundations of the annotations as regression of the goal to relevant plan steps. Annotations (implicitly the regression) were computed as part of the planning algorithm (backward chaining, POP, or HTN planning). Our formalization in Section 3 elucidates this approach making it easily applicable to other planning algorithms.

To the best of our knowledge, the SHERPA system (Koenig, Furcy, & Bauer 2002) is the sole previous work that addresses the problem of monitoring the continued optimality of a plan, though only in a limited form. SHERPA lifts the Life-Long Planning A^* (LPA^*) search algorithm to symbolic propositional planning. LPA^* was developed for the purpose of replanning in problems like robot navigation (i.e. path (re-)planning). As such this algorithm only applies to replanning problems where the current state remains unchanged, but the costs of actions in the search tree have changed. This is a major limitation. Similar to our approach, SHERPA retains the search tree to determine how changes may affect the current plan. But again, these changes are limited to action costs, while our approach guarantees optimality in the general case.

Another advantage of our approach is that it facilitates active sensing on the part of the agent. Our annotations can be exploited by the agent to quickly discern conditions that are relevant to a situation. Others have recognized the importance of this issue (cf. e.g. (Doyle, Atkinson, & Doshi 1986)), but to the best of our knowledge we are the first to address it with respect to the relevant features for optimality, rather than just for validity.

Also related is (Veloso, Pollack, & Cox 1998) in which the authors exploit the 'rationale', the reasons for choices made during planning, to deal with discrepancies that occur during *planning*. The authors acknowledge the possibility that previously sub-optimal alternatives become better than the current plan candidate as the world evolves during planning, but the treatment of optimality is informal and limited.

In (De Giacomo, Reiter, & Soutchanski 1998) the authors formalize an execution monitoring framework in the situation calculus to monitor the execution of Golog programs. However, they too are only concerned with plan validity. They do not follow the approach of goal regression but instead use projection every time a discrepancy is detected.

Also worth noting is the distinction between our approach to execution monitoring and so-called "Universal Plans" (Schoppers 1987). Roughly, universal plans perform replanning ahead of time, by regressing the goal over all possible action sequences. This approach is infeasible, as it grows exponentially in the number of domain features. The complexity of our approach on the other hand is bounded by the size of the search tree expanded during planning.

Also related are the results of (Nebel & Koehler 1995). They show that plan reuse – and plan repair in reaction to unforeseen discrepancies is a special case of this – is as complex as planning from scratch in worst case, and that this result even holds for cases where the considered planning instances are very similar. These results further motivate our work, as they highlight the benefit of avoiding replanning entirely whenever possible.

7 Summary and Future Work

When executing plans in dynamic environments, discrepancies between the expected and actual state of the world can arise for a variety of reasons. When such circumstances cannot be anticipated and accounted for during planning, they bring into question whether discrepancies are relevant, and whether they render the current plan invalid or sub-optimal. While there are several approaches for monitoring validity, no approaches exist for monitoring optimality. Instead it is common practice to replan when a discrepancy occurs or to ignore the discrepancy, accepting potentially sub-optimal behavior. Time-consuming replanning is impractical in highly dynamic domains and many discrepancies are irrelevant and thus replanning unnecessary, but to maintain optimality we have to determine which these are.

This paper makes several contributions to this problem. We provided a formal characterization of a common technique for monitoring plan validity based on annotating the plan with conditions for the continued validity, which are checked during execution. We then generalized this to the notion of plan optimality, providing a sufficient condition for optimality and an algorithm that exploits knowledge about the actual discrepancy to quickly test this condition at execution. This approach guarantees plan optimality while minimizing unnecessary replanning.

Relevant properties of a situation as provided in the annotation, can further serve to focus on-line sensing when faced with limited sensing resources: the agent knows which features to sense and which can simply be ignored as they do not influence its objective.

The requirements for the applicability of our method are easy to fulfill: The used action language has to be expressive enough to represent the user's preferences and regression has to be defined.

We implemented our algorithm and tested it on systematically generated execution discrepancies in the `TPP` and `open stacks` domains. The results show that many discrepancies are irrelevant, leaving the current plan optimal, and that our approach is much faster than replanning, with an average speed-up of two orders of magnitude.

In future work we plan to broaden our view to several other planning paradigms, first and foremost, decision theoretic planning. In real world domains it is likely that we will be able to model and plan for some of the uncertainty, while other contingencies remain unaccounted for. In the latter, a future version of our algorithm for decision theoretic planning can again be used to assert continued plan optimality to minimize on-line replanning time. Finally, we are trying to find complexity results for deriving and testing necessary (not just sufficient) conditions, but believe that the evaluation of such a condition would not outperform replanning.

Acknowledgments: We would like to thank the anonymous reviewers for their helpful comments, Jorge Baier for providing detailed comments on an early version of this paper, and the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Ontario Ministry of Research and Innovation (MRI) for financial support.

References

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

De Giacomo, G.; Reiter, R.; and Soutchanski, M. 1998. Execution monitoring of high-level robot programs. In *Proc. KR'98*, 453–465.

Doyle, R.; Atkinson, D.; and Doshi, R. 1986. Generating perception requests and expectations to verify the execution of plans. In *Proc. AAAI'86*, 81–88.

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

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

Koenig, S.; Furcy, D.; and Bauer, C. 2002. Heuristic search-based replanning. In *Proc. AIPS'02*, 294–301.

Nebel, B., and Koehler, J. 1995. Plan reuse versus plan generation: A theoretical and empirical analysis. In *Artificial Intelligence (Special Issue on Planning and Scheduling), 76(1-2)*, 427–454.

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.

Schoppers, M. J. 1987. Universal plans for reactive robots in unpredictable environments. In *Proc. IJCAI'87*.

Veloso, M.; Pollack, M.; and Cox, M. 1998. Rationale-based monitoring for continuous planning in dynamic environments. In *Proc. AIPS'98*, 171–179.

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