Planning in The Face of Frequent Exogenous Events

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

Abstract

Generating optimal plans in highly dynamic environments is challenging. Plans are predicated on an assumed initial state, but this state can change unexpectedly during plan generation, potentially invalidating the planning effort. In this paper we make three contributions: (1) We propose a novel algorithm for generating optimal plans in settings where frequent, unexpected events interfere with planning. It is able to quickly distinguish relevant from irrelevant state changes, and to update the existing planning search tree if necessary. (2) We argue for a new criterion for evaluating plan adaptation techniques: the *relative* running time compared to the "size" of changes. This is significant since during recovery more changes may occur that need to be recovered from subsequently, and in order for this process of repeated recovery to terminate, recovery time has to *converge* to zero. (3) We show empirically that our approach can converge and find optimal plans in environments that would ordinarily defy planning because of their high dynamics. In this paper we use the situation calculus to formalize and prove correctness of our approach. Nevertheless, our approach is implementable using a variety of planning formalisms.

1 Introduction

Many real-world planning applications are situated within highly dynamic environments. In such environments, the initial state assumed by the planner frequently changes in unpredictable ways during planning, possibly invalidating the current planning effort. We argue that neither boldly ignoring such changes nor replanning from scratch is an appealing option. While the former is unlikely to produce a good plan, the latter may never be able to complete a plan when unexpected events keep interrupting. Instead we propose an integrated planning and recovery algorithm that explicitly reasons about the relevance and impact of discrepancies between assumed and observed initial state.

As a motivating example, consider a soccer playing robot in RoboCup, which, having the ball, deliberates about how to score. In RoboCup it is common to receive sensor readings 10 times per second. The game environment is very dynamic, resulting in frequent discrepancies between assumed and observed initial state. Such discrepancies may or may not affect the current planning process. But how can the robot tell? And how should the robot react when discrepancies are deemed relevant? For instance, assume that at some point during planning, the current most promising plan starts with turning slightly to face the goal and then driving there, pushing the ball. If the ball unexpectedly rolls 10 centimeters away while deliberating, the initial turn action may cause the robot to lose the ball, so this discrepancy is relevant and another plan, starting by re-approaching the ball, should be favored. But if the ball rolls closer, the original plan remains effective and the discrepancy should be ignored and planning continued.

The contributions of this paper are three-fold: (1) We propose a novel algorithm for plan generation that monitors the state of the world during planning and recovers from unexpected state changes that impact planning. The algorithm produces plans that are optimal with respect to the state where execution begins. It is able to distinguish between relevant and irrelevant discrepancies, and updates the planning search tree to reflect the new initial state if necessary. This is generally much faster than replanning from scratch, and works for arbitrary state changes that are representable in the domain specification. (2) We introduce a new criterion for evaluating plan adaptation algorithms: their *relative* running time compared to the "size" of the discrepancy. We argue that this measure is of greater practical significance than either theoretical worst case considerations or the absolute recovery time for the following reason: In highly dynamic domains unexpected state changes occur during planning as well as during plan adaptation. In order to obtain a plan that is known to be optimal when execution commences, the cycle of planning and recovery has to terminate by a completed recovery before the state changes any further. This is possible when the time for recovery is roughly proportional to the size of the change. Imagine planning takes 10 seconds and recovering from any state changes that occurred during that time takes 8 seconds. If we assume that in 8 seconds on average fewer changes happen than in 10, it seems reasonable to expect that we can recover from those in less than 8 seconds, say on average 6. This continues, until recovery has "caught up with reality". We informally say that an algorithm with this property *converges*. Repeated replanning from scratch obviously does not converge, as it does not differentiate between "big" and "small" discrepancies. (3) We show empirically that our algorithm can converge and find optimal plans in domains that were previously not amenable to planning due to their high dynamics. Particularly "on-the-fly" recovery, i.e. recovering immediately upon discrepancy detection, has a higher chance of convergence than the al-

ternative of completing the original planning task first and recovering only afterwards.

We explicitly assume that the number and extent of discrepancies is on average proportional to the time interval, i.e. that greater discrepancies are incurred in longer time intervals. This seems reasonable to us and holds for many interesting application domains. This, together with the observation that our algorithm can recover from a few small changes faster than from many large ones, provides the convergence of our approach. We demonstrate this and the resulting convergence of our approach empirically, on domain simulations which satisfy this assumption.

We understand optimality to be defined in terms of what is currently known, and we want to execute plans only when they are considered optimal at the moment execution begins. This seems rational, since future events cannot generally be predicted. Nevertheless, we point out that this may lead to behavior that, in hindsight, is sub-optimal compared to a seemingly worse but quickly produced plan, namely when bad or catastrophic events in the environments can be avoided by planning and acting more quickly. We also explicitly assume that everything that matters for optimality is modeled in the theory. In particular, we assume that planning time itself does not directly affect optimality.

After reviewing some preliminaries in the next section, we describe our approach in Section 3, followed by empirical results and a discussion including related work.

2 Background

The approach we present works with any action specification language for which regression can be defined, including STRIPS and ADL. For the exposition in this paper, we use the situation calculus with a standard notion of arithmetic.

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, called *fluents* (set $\mathcal{F}$), relativized to a *situation* s, e.g., $F(\vec{x}, s)$. A situation is a *history* of the primitive 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 often suppressed. Also, $do(a_n, do(a_{n-1}, \ldots do(a_1, s)))$ is abbreviated to $do([a_1, \ldots, a_n], s)$ or $do(\vec{a}, s)$ and we define $do([\,], s) = s$. In this paper we distinguish between a finite set of *agent actions*, $\mathcal{A}_{\text{agent}}$, and a possibly infinite set of *exogenous actions* (or *events*), $\mathcal{A}_{\text{exog}}$, ($\mathcal{A} = \mathcal{A}_{\text{agent}} \cup \mathcal{A}_{\text{exog}}$). The agent can only perform agent actions, and exogenous events can happen at any time, including during planning.

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). Included in the domain-dependent axioms are the following sets:

Initial State: 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 $\vec{x}, a, s$. $\Phi_F(\vec{x}, a, s)$ characterizes the truth value of the fluent $F(\vec{x})$ in the situation $do(a, s)$ in terms of what is true in situation s. These axioms can be automatically generated from effect axioms (e.g. add- and delete-lists).

Action precondition axioms: specify the conditions under which an action is possible. There is one axiom for each $a \in \mathcal{A}_{\text{agent}}$ 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 assume exogenous events $e \in \mathcal{A}_{\text{exog}}$ always possible ($Poss(e, s) \equiv true$).

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, one step regression is defined inductively using the successor state axiom for a fluent $F(\vec{x})$ as above (Reiter 2001):

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

We use $\mathcal{R}[\psi(s), \alpha]$ to denote $Regr[\psi(do(\alpha, s))]$, and $\mathcal{R}[\psi(s), \vec{\alpha}]$ to denote the repeated regression over all actions in the sequence $\vec{\alpha}$ (in reverse order). Note that the resulting formula has a free variable s of sort situation. Intuitively, it is the condition that has to hold in s in order for ψ to hold after executing $\vec{\alpha}$ (i.e. in $do(\vec{\alpha}, s)$). It is predominantly comprised of the fluents occurring in the conditional effects of the actions in $\vec{\alpha}$. Due to the Regression Theorem (Reiter 2001) we have that $\mathcal{D} \models \psi(do(\vec{\alpha}, s)) \equiv \mathcal{R}[\psi(s), \vec{\alpha}]$ for all situations s.

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: We use α to denote arbitrary but explicit actions and S to denote arbitrary but explicit situations, that is $S = do(\vec{\alpha}, S_0)$ for some explicit action sequence $\vec{\alpha}$. Further $\vec{\alpha} \cdot \alpha$ denotes the result of appending action α to the sequence $\vec{\alpha}$.

Going back to our RoboCup example, regressing the goal "ball in goal" over the action "drive to goal", yields a condition "have ball". The further regression over a "turn" action states "distance to ball $< 10cm$" as a condition for the success of the considered plan, if, e.g., the robot's 10cm long grippers enable turning with the ball.

3 Planning with Unexpected Events

In this paper we consider a planner based on A^* search that uses positive action costs as a metric, but the conceptual approach is amenable to a variety of other forward-search based planning techniques and paradigms.

Intuitively, our approach annotates the search tree with all relevant information for determining the optimal plan. By regressing the goal, preconditions, and metric function

over all considered action sequences, this information is expressed in terms of the current state. When unexpected events change the current state of the world, this allows us to reason symbolically about their relevance and their potential impact on the current search tree and choice of plan—much faster then replanning from scratch.

For instance, our soccer robot from above knows from regressing the goal that the plan ["turn", "drive to goal"] will succeed whenever "distance to ball $< 10cm$" holds. Hence it can determine the relevance of the aforementioned ball displacements, and also that, for instance, unexpected actions of its teammates can be ignored for now. A complication of this arises from our interest in an optimal, rather than just *some* plan, however. We will need to also consider alternative action sequences, and also handle impacts on the regressed metric function.

3.1 Regression-based A^* planning

In this section we present an A^* planner that returns not only a plan, but also the remaining open list upon termination of search, as well as a search tree annotated with any relevant regressed formulae.

To provide a formal characterization, we assume that the planning domain is encoded in a basic action theory $\mathcal{D}$. Given an initial situation S, a goal formula $\mathit{Goal}(s)$, a formula $\mathit{Cost}(a, c, s)$ defining costs c of action a, and an admissible heuristic specified as a formula $\mathit{Heu}(h, s)$, A^* search finds a sequence of actions $\vec{\alpha}$ such that the situation $do(\vec{\alpha}, S)$ satisfies the goal while minimizing the accumulated costs. The (extra-logical) formula $\mathit{Value}(v, do([\alpha_1, \dots, \alpha_N], s)) \stackrel{\text{def}}{=} (\exists h, c_1, \dots, c_N).\mathit{Heu}(h, do([\alpha_1, \dots, \alpha_N], s)) \wedge \mathit{Cost}(\alpha_1, c_1, s) \wedge \dots \wedge \mathit{Cost}(\alpha_N, c_N, do([\alpha_1, \dots, \alpha_{N-1}], s)) \wedge v = h + c_1 + \dots + c_N$, guides this search. Starting with an open list, *Open*, containing only one element representing the empty action sequence, the search proceeds by repeatedly removing and expanding the first element from the list until that element satisfies the goal, always maintaining the open list's order according to Value. We assume that the goal can only, and will unconditionally, be achieved by a particular agent action finish. Any planning problem can naturally be transformed to conform to this by defining the preconditions of finish according to the original goal.

Our regression-based version of A^* is shown in Figure 1. It interacts with the basic action theory $\mathcal{D}$ to reason about the truth-values of formulae. We say ψ holds, to mean that it is entailed by $\mathcal{D}$. The algorithm is initially invoked as $\mathit{regrA}^*(\mathcal{D}, S, \mathit{Goal}, \mathit{Cost}, \mathit{Heu}, [(0, \infty, [\,])], nil)$. The last argument denotes a data structure representing the annotated search tree and is initially empty. The elements of the open list are tuples $(g, h, \vec{\alpha})$, where $\vec{\alpha} = [\alpha_1, \dots, \alpha_n]$ is an action sequence, g are the costs accumulated when executing this sequence in S, and h is the value s.t. $\mathit{Heu}(h, do(\vec{\alpha}, S))$ holds. When an element is expanded, it is removed from the open list and the following is performed for each agent action α': First, the preconditions of α' are regressed over $\vec{\alpha}$ (Line 6). If the resulting formula, stored in $T(\vec{\alpha}).P(s)$, holds in S according to $\mathcal{D}$ (Line 7), the cost formula for α' is regressed over $\vec{\alpha}$, the heuristic is regressed over $\vec{\alpha} \cdot \alpha'$, and the resulting

$\mathit{regrA}^*(\mathcal{D}, S, \mathit{Goal}, \mathit{Cost}, \mathit{Heu}, \mathit{Open}, T)$:
1 **if** $\mathit{Open} = [\,]$ **then return** $([\,], T)$
2 **else** $[(g, h, \vec{\alpha}) \mid \mathit{Open}'] = \mathit{Open}$ *// slice first element*
3 **if** $\mathit{Goal}(do(\vec{\alpha}, S))$ holds **then return** (Open, T)
4 **else foreach** $\alpha' \in \mathcal{A}_{\text{agent}}$ **do**
5 $\vec{\alpha}' \leftarrow \vec{\alpha} \cdot \alpha'$ *// append action to sequence*
6 $T(\vec{\alpha}').P(s) \leftarrow \mathcal{R}[\mathit{Poss}(\alpha', s), \vec{\alpha}]$
7 **if** $T(\vec{\alpha}').P(S)$ holds **then**
8 $T(\vec{\alpha}').p \leftarrow \mathit{true}$ *// action currently possible*
9 $T(\vec{\alpha}').C(c, s) \leftarrow \mathcal{R}[\mathit{Cost}(\alpha', c, s), \vec{\alpha}]$
10 $T(\vec{\alpha}').H(h, s) \leftarrow \mathcal{R}[\mathit{Heu}(h, s), \vec{\alpha}']$
11 $T(\vec{\alpha}').c \leftarrow c'$ with c' s.t. $T(\vec{\alpha}').C(c', S)$ holds
12 $T(\vec{\alpha}').h \leftarrow h'$ with h' s.t. $T(\vec{\alpha}').H(h', S)$ holds
13 insert $(g + c',\ h',\ \vec{\alpha}')$ into Open'
14 **else** $T(\vec{\alpha}').p \leftarrow \mathit{false}$ *// action currently impossible*
15 **return** $\mathit{regrA}^*(\mathcal{D}, S, \mathit{Goal}, \mathit{Cost}, \mathit{Heu}, \mathit{Open}', T)$

Figure 1: Pseudo-code for regression-based A^* planning.

formulae are evaluated in S yielding values c' and h' (Lines 9–12). Intuitively, the regression of these formulae over $\vec{\alpha}$ describes in terms of the current situation, the values they will take after performing $\vec{\alpha}$. Finally, a new tuple is inserted into the open list (Line 13). It is done according to $g + c' + h'$ to maintain the open list's order according to $\mathit{Value}(v, s)$.

A^* keeps expanding the first element of the open list, until this element satisfies the goal, in which case the respective action sequence describes an optimal plan. This is because an admissible heuristic never over-estimates the actual remaining costs from any given state to the goal. Due to the Regression Theorem (Reiter 2001), this known fact about A^* also holds for our regression-based version. Similarly the completeness of A^* is preserved.

In service of our recovery algorithm described below, we explicitly keep the search tree T and annotate its nodes with the regressed formulae for preconditions ($T(\vec{\alpha}).P(s)$), costs ($T(\vec{\alpha}).C(c, s)$), and heuristic value ($T(\vec{\alpha}).H(h, s)$) and their values according to the (current) initial situation ($T(\vec{\alpha}).p, T(\vec{\alpha}).c$, and $T(\vec{\alpha}).h$). Roughly, when the initial state changes due to an unexpected event e, it reevaluates $T(\vec{\alpha}).P(s), T(\vec{\alpha}).C(c, s)$, and $T(\vec{\alpha}).H(h, s)$ in $s = do(e, S)$, and updates their values and the open list accordingly.

However, we can gain significant computational savings by reducing reevaluations to only those formulae actually affected by the state change. And since all formulae are *regressed*, we can determine which ones are affected, by simply considering the fluents they mention. For this purpose we create an index *Index* whose keys are fluent atoms (e.g. *distanceTo(ball)*) and whose values are lists of pointers to all stored formulae that mention it.

3.2 Recovering from Unexpected Events

While generating a plan for an assumed initial situation S, an unexpected event e, say "*distanceTo(ball)* $\leftarrow$ 20", may occur, changing the state of the world and putting us into situation $do(e, S)$. When this happens, our approach consults the aforementioned index to pinpoint all formulae affected by this change (e.g. $T([\mathit{turn, driveTo(goal), finish}]).P(s)$). After reevaluating these formulae in $do(e, S)$ and updating their values, the search tree will be up-to-date in the sense that all

```
recover(D, S1, S2, Cost, Heu, Open, T, Index) :
1    F_Δ ← {F ∈ keys(Index) | F(S1) ≢ F(S2) holds }
2    Δ ← ⋃_{F∈F_Δ} Index(F)                          // affected formulae
3    foreach (α⃗, 'p') ∈ Δ do                   // update preconditions
4      if T(α⃗).p = true and ¬T(α⃗).P(S2) holds then
5        T(α⃗).p ← false                     // action now impossible
6        foreach (g, h, α⃗′) ∈ Open do
7          if α⃗ is prefix of α⃗′ then
8            remove (g, h, α⃗′) from Open
9      elseif T(α⃗).p = false and T(α⃗).P(S2) holds then
10       T(α⃗).p ← true                           // action now possible
11       α⃗′ · α_last = α⃗                  // get last action in sequence
12       T(α⃗).C(c, s) ← R[Cost(α_last, c), α⃗′]
13       T(α⃗).H(h, s) ← R[Heu(h), α⃗]
14       T(α⃗).c ← c′ with c′ s.t. T(α⃗).C(c′, S2) holds
15       T(α⃗).h ← h′ with h′ s.t. T(α⃗).H(h′, S2) holds
16       g′ ← getGval(T, α⃗)
17       insert (g′, h′, α⃗) into Open and update Index
18   foreach (α⃗, 'c') ∈ Δ do          // update accumulated costs
19     get c′ s.t. T(α⃗).C(c′, S2) holds
20     offset ← c′ − T(α⃗).c
21     foreach (g, h, α⃗′) ∈ Open do
22       if α⃗ is prefix of α⃗′ then g ← g + offset
23     T(α⃗).c ← c′
24   foreach (α⃗, 'h') ∈ Δ do              // update heuristic values
25     if (∃g, h).(g, h, α⃗) ∈ Open then
26       h ← h′ with h′ s.t. T(α⃗).H(h′, S2) holds
27       T(α⃗).h ← h
28 return (sort(Open), T)
```

Figure 2: Pseudo-code of our recovery algorithm.

its contained values are with respect to $do(e, S)$ rather than the originally assumed initial situation S. After propagating this change to the open list, search can continue, producing the same result as if A^* (or *regrA**) had set out in $do(e, S)$ (cf. Theorem 1). *Note that the regressed formulae never change.* Assuming that most unexpected state changes only affect a few fluents and thus often only affect a small subset of all formulae, our annotation allows for great computational savings when recovering from changes, as we show empirically in the next section.

The recovery algorithm is specified in Figure 2. T denotes the annotated search tree, *Open* is the open list, and *Index* the index. The latter contains entries of the form $(\vec{\alpha}, type)$, where $\vec{\alpha}$ is a sequence of actions and $type$ is either of `'p'`, `'c'`, or `'h'`. The algorithm modifies the values of the tree and the open list (ll. 22 and 26) to reflect their value with respect to a new situation S_2 (e.g. $do(e, S_1)$) rather than an originally assumed initial situation S_1. If the event changes the truth value of action preconditions, the content of the open list is modified accordingly (ll. 8, 17). When a previously impossible action has now become possible (Line 9) the annotation for this node is created and a new entry added to the open list (ll. 11-17). The function $getGval(T, \vec{\alpha})$ computes the sum of all costs ($T(\cdot).c$) annotated in T along the branch from the root to node $\vec{\alpha}$.

The algorithm can be used in one of at least two ways: during planning ("on-the-fly"), dealing with unexpected state changes immediately, or right after planning ("at-the-end"), dealing at once with all events that occurred during planning. The former has the advantage that the planning effort is focused more tightly on what is actually relevant given everything that has happened so far. This approach can be implemented by inserting code right before Line 15 of *regrA** that checks for events and invokes *recover* if necessary, changing $S, Open'$, and T accordingly. The appeal of the latter stems from the fact that recovering from a bulk of events simultaneously can be more efficient than recovering from each individually. It may, however, be necessary to resume *regrA** search afterwards, if, for instance, the current plan is no longer valid in the new initial state or a new opportunity exists, which may lead to a better plan. With both approaches, additional events may happen during recovery, making additional subsequent recoveries necessary.

The following theorem states the correctness of *recover* in terms of the "at-the-end" approach: calling *recover* and continuing *regrA** with the new open list and tree, produces an optimal plan and in particular the same as replanning from scratch in S_2. Recall that the first element of the open list contains the optimal plan. For "on-the-fly", correctness can be shown analogously (cf. Lemma 1 in the Appendix).

Theorem 1 (Correctness)**.** Let $\mathcal{D}$ be a basic action theory, *Goal* a goal formula, $Cost(a, c)$ a cost formula, and $Heu(h)$ an admissible heuristic. Then, for any two situations S_1, S_2 in $\mathcal{D}$ we have that after the sequence of invocations:

$$(Open_1, T_1) \leftarrow regrA^*(\mathcal{D}, S_1, Goal, Cost, Heu, [(0, \infty, [\,])], nil),$$
$$\text{create } Index \text{ from } T_1,$$
$$(Open_2, T_2) \leftarrow recover(\mathcal{D}, S_1, S_2, Open_1, T_1, Index),$$
$$(Open_3, T_3) \leftarrow regrA^*(\mathcal{D}, S_2, Goal, Cost, Heu, Open_2, T_2)$$

the first element of $Open_3$ will be the same as in $Open'$ of $(Open', T') \leftarrow regrA^*(\mathcal{D}, S_2, Goal, Cost, Heu, [(0, \infty, [\,])], nil)$, or both $Open_3$ and $Open'$ are empty. *Proof:* Appendix.

This, in particular, works for any situation pair $S_1, S_2 = do(\vec{e}, S_1)$, for any sequence of events $\vec{e}$. Note that such events can produce arbitrary changes to the state of the world. The algorithm does not make any assumptions about possible events, any fluent may assume any value at any time.

In complex domains, many state changes are completely irrelevant to the current planning problem, overall or at the current stage of planning, and others only affect a small subset of elements in the search tree. During recovery, we exploit this structure to gain significant speed-ups compared to replanning from scratch. More importantly this allows our algorithm to recover from small perturbations faster than from large ones, where "large" may refer to the number of fluents that changed or the amount by which continuous fluents changed (cf. Section 4). This is what allows our algorithm to *converge*, i.e. "catch up with reality", as we defined informally in the introduction. We verified this empirically.

4 Empirical Results

We present empirical results obtained using a current implementation of our algorithm to generate optimal plans for differently sized problems of the metric TPP and Zenotravel domains of the International Planning Competition. We begin by showing that the time required for recovering from unexpected state changes is roughly and on average proportional to the extent of the change. We then show that our

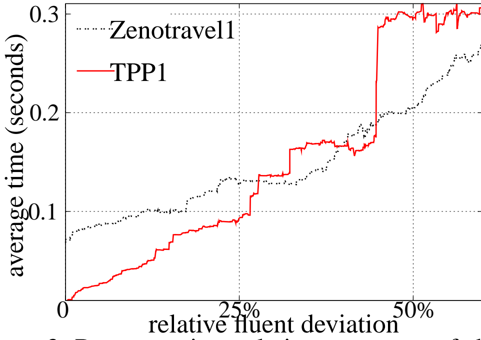

Figure 3: Recovery time relative to amount of change.

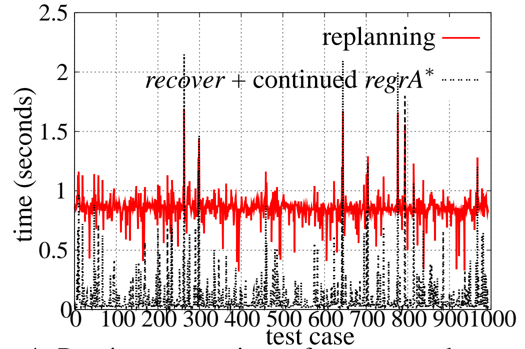

Figure 4: Runtime comparison of our approach vs. replanning from scratch on the TPP domain.

approach is able to find optimal plans even when the initial state changes frequently. We compare the two mentioned recovery strategies on-the-fly and at-the-end, showing that the former clearly outperforms the latter in terms of likelihood of convergence. Finally, and not surprisingly, we show that our approach generally outperforms replanning from scratch. All experiments were run on an Intel Xeon 2.66 GHz with 2GB RAM.

Figure 3 plots the average time the combination of *recover* + continued *regrA** search took to find a new optimal plan, after the value of a randomly selected continuous fluent was randomly modified after generating an optimal plan. A deviation $x\%$ means that the fluent was multiplied by $1 \pm 0.x$, e.g. 50% means a factor of 1 ± 0.5. Note that we used continuous fluents in our experiments only because they lend themselves better to a quantitative evaluation. Our approach is equally applicable to discrepancies on discrete valued, including Boolean, fluents. As one can see, the time to recover from a drastic change takes on average longer than for minor deviations. While this doesn't seem surprising, we present it here since it provides the intuition for the convergence behavior of our approach, which we study next.

We assume that over longer periods of time more things change and in greater amounts than over shorter periods of time. Recovery generally takes less time than the original planning did (see below). Hence, we assume less or fewer changes will happen during recovery than during planning. A second recovery – from the events that occurred during the first recovery – is thus predicted to take less time than the first. This process often continues until convergence. We studied the conditions under which our algorithm converges by simulating domains with frequent changes to the initial state. At high frequencies during planning and subsequent recoveries, we randomly perturbed some fluent by an amount of up to a certain maximum between 5-80%. We considered the two approaches described earlier: completing the original planning task and recovering only once a plan is found (at-the-end), followed by further *regrA** search if needed, or reacting to state changes immediately (on-the-fly), pausing further *regrA** expansion until *recover* has brought the current search tree up-to-date. In both cases, several episodes of recovery and additional *regrA** search were generally required before finding an optimal and up-to-date plan. Their number varied strongly, as a result of some discrepancies having larger impact than others. Table 1 shows the percentages of simulations in which an optimal plan was found, i.e. the algorithm converged within the time limit, for different frequencies and amounts of perturbation. As time limit we used 30 times the time required for solving the respective original problems without perturbations using a conventional A^* search planner. These were 0.52s for TPP1, 2.17s for TPP2, 3.03s for TPP3, and 0.34s, 0.82s, and 1.58s for Zenotravel 1, 2, and 3 respectively. The frequencies shown in the table are relative to these as well. For instance, the value 100 for Zenotravel1 on-the-fly, 5Hz, 40% states that even when every $0.34s/5 = 68$ ms the value of a random fluent changed by up to 40% in the considered Zenotravel problem, the on-the-fly approach still converged 100% of the time. This simulates a quite erratic environment, possibly harsher than many realistic application domains.

The on-the-fly recovery strategy clearly outperforms at-the-end recovery. This makes intuitive sense, as no time is wasted continuing planning for an already flawed instance. This also motivates an integrated approach, showing its benefit over the use of plan adaptation approaches which are only applicable once a first plan has been produced.

The table also shows that convergence was much better on TPP than on Zenotravel. Interestingly, this was predictable given Figure 3: since the curve for Zenotravel1 intersects the y-axis at around 0.07 seconds, it seems unreasonable to expect convergence on this problem when the initial state changes at intervals shorter than that. This explains the low probability of convergence when events occur at 10Hz times planning time, i.e. every 0.034s.

Since replanning from scratch takes the same amount of time, no matter how small the discrepancy is, assuming the problem does not get significantly easier through this, it has no chance of ever catching up with reality when events happen at time intervals shorter than the time required for plan generation. Our approach thus enables the application of planning in domains where this was not previously possible.

Not surprisingly, our approach generally outperforms replanning from scratch. To demonstrate this, we compared the times required by both approaches for recovering from a single change of the initial state. The setup was as follows: We solved a planning problem, perturbed the state of the world by randomly changing some fluent's value, and then ran both (a) *recover* followed by further *regrA** search based on the modified open list if necessary, and (b) replanning

Frequency:	3Hz · planning time					5Hz · planning time					10Hz · planning time				
Deviation:	5%	10%	20%	40%	80%	5%	10%	20%	40%	80%	5%	10%	20%	40%	80%
tpp1 at-the-end	100	100	100	83	60	100	100	83	63	43	100	100	76	43	20
tpp1 on-the-fly	100	100	100	86	83	100	100	96	80	83	100	100	93	80	70
tpp2 at-the-end	96	86	60	63	43	100	80	51	44	34	89	48	34	24	10
tpp2 on-the-fly	100	93	86	86	83	96	86	75	86	82	96	86	86	79	82
tpp3 at-the-end	100	73	50	66	41	94	72	55	42	52	76	31	42	32	20
tpp3 on-the-fly	100	96	87	92	72	94	84	86	87	89	89	81	81	89	86
zenotravel1 at-the-end	100	96	100	100	76	66	76	63	43	56	3	6	0	6	16
zenotravel1 on-the-fly	100	100	100	100	100	96	96	100	100	86	66	73	70	93	93
zenotravel2 at-the-end	66	43	30	26	3	30	16	6	6	6	10	0	0	0	0
zenotravel2 on-the-fly	86	70	53	53	40	36	16	30	26	23	13	6	0	6	20
zenotravel3 at-the-end	100	80	56	28	8	97	72	12	7	7	33	10	0	0	2
zenotravel3 on-the-fly	100	92	60	56	66	90	75	60	30	43	43	28	33	25	21

Table 1: Percentage of test cases where our approach converged within the time limit, by event frequencies and deviation amounts.

from scratch using a conventional A^* search implementation using the same heuristic. Figure 4 shows the time both approaches require to recover from single events on our TPP1 problem. Recall that with both approaches the resulting plan is provably optimal. The average speed-up was 24.11. We performed the same experiment on the Zenotravel1 problem. There we tested using two different, hand-coded heuristics, where the first is more informed (better) than the second. Using the first, which we also used in the earlier described experiments, the average recovery time was 0.14s, and the average replanning time was 0.51s, whereas with the second heuristic recovery time averaged to 0.35s and replanning to 1.07s. This shows that even when the planner, and thus replanner, is improved by the use of a better heuristic, our approach is still generally superior to replanning from scratch. This is because it equally benefits from a smaller search tree, resulting from the use of a better heuristic.

5 Discussion

In this paper we made three contributions: (1) We presented a novel integrated planning and recovery algorithm for generating optimal plans in environments where the state of the world frequently changes unexpectedly during planning. At its core, the algorithm reasons about the relevance and impact of discrepancies, allowing the algorithm to recover from changes more efficiently than replanning from scratch. (2) We introduced a new criterion for evaluating plan adaptation approaches, called convergence, and argued for its significance. (3) We empirically demonstrated that our approach is able to converge even under high frequencies of unexpected state changes. Our experiments also show that an interleaved planning-and-recovery approach which recovers from such discrepancies on-the-fly is superior to an approach that only recovers once planning has completed.

Our approach needs to be paired with an equally effective execution monitor, since during execution the optimality of the executing plan may also be jeopardized by exogenous events. We describe one way of monitoring plan optimality during execution in (Fritz & McIlraith 2007).

In the future, we intend to apply this work to a highly dynamic real-world domain such as the mentioned RoboCup or Unmanned Aerial Vehicles. To do so, an optimized version of our implementation is required. While the approach is able to handle changes in the state, which can also be used to model changes in executability and cost of actions, we would like to study changing goals, too. We also think that the ideas behind the presented approach may be beneficially applied to planning under initial state uncertainty, in particular when such uncertainty ranges over continuous domains.

The presented approach is one of the first to monitor and react to unexpected state changes during planning. The approach taken by Veloso, Pollack, & Cox (1998) exploits the "rationale", the reasons for choices made during planning, to deal with discrepancies that occur during planning. They acknowledge the possibility that previously sub-optimal alternatives may become better than the current plan candidate as the world evolves during planning, but the treatment of optimality is informal and limited. No guarantees are made regarding the optimality of the resulting plan. Also, by using best-first search, our approach is compatible with many state-of-the-art planners, while the approach of Veloso, Pollack, & Cox is particular to the PRODIGY planner.

Several approaches exist for adapting a plan in response to unexpected events that occur during execution, rather than during plan generation, e.g. (Koenig, Furcy, & Bauer 2002; Hanks & Weld 1995; Gerevini & Serina 2000). Arguably we could use these approaches for our purpose of recovering from discrepancies during planning, by first ignoring the changes and then recovering once a plan is generated. We think this is inferior to our approach for the following reasons: (1) except for the first, the listed approaches do not guarantee optimality, (2) we have shown that an integrated approach which recovers from state changes on-the-fly has convergence advantages, and (3) it is not clear whether such a use of these replanners would at all lead to convergence.

The SHERPA system presented by Koenig, Furcy, & Bauer (2002) monitors the continued optimality of a plan 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 replanning) with simple, unconditional actions, and only applies to replanning problems where the costs of actions have changed but the current state remains the same. Similar to our approach, SHERPA retains the search tree to determine how changes may affect the current plan. Our approach subsumes this approach and further allows for the general case where the initial (current) state may change arbitrarily and the dynam-

ics of the domain may involve complex conditional effects. SHERPA's limitations equally apply to more recent work by Sun & Koenig (2007). The presented Fringe-Saving A* (FSA*) search algorithm, which sometimes performs better than LPA*, is further limited to grid world applications and the use of the Manhattan distance heuristic. This algorithm retains the open list of previous searches as well.

The idea of deriving and utilizing knowledge about relevant conditions of the current state for monitoring and possibly repairing a plan, has been used before, e.g. Kambhampati (1990), and reaches back to the early work on Shakey the Robot by Fikes, Hart, & Nilsson (1972). Fikes, Hart, & Nilsson used triangle tables to annotate the plan with the regressed goal, in order to determine whether replanning was necessary when the state of the world changed unexpectedly.

Nebel & Koehler (1995) show that plan reuse has the same worst case complexity as planning from scratch. This result is interesting in theory, but not so relevant in the practical case of optimal plan generation in the face of frequent unexpected events. In this case, we have shown that if we want to have a plan that we know to be optimal at the start of execution, then the recovery time relative to the impact of an event is more important.

Acknowledgments: We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Ontario Research Foundation Early Researcher Award.

References

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

Fritz, C., and McIlraith, S. A. 2007. Monitoring plan optimality during execution. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS)*, 144–151.

Gerevini, A., and Serina, I. 2000. Fast plan adaptation through planning graphs: Local and systematic search techniques. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS)*, 112–121.

Hanks, S., and Weld, D. S. 1995. A domain-independent algorithm for plan adaptation. *J. Artif. Intell. Res. (JAIR)* 2:319–360.

Kambhampati, S. 1990. A theory of plan modification. In *Proc. of the 8th National Conference on Artificial Intelligence*, 176–182.

Koenig, S.; Furcy, D.; and Bauer, C. 2002. Heuristic search-based replanning. In *Proc. of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS)*, 294–301.

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

Pednault, E. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 324–332.

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

Sun, X., and Koenig, S. 2007. The fringe-saving A* search algorithm - a feasibility study. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2391–2397.

Veloso, M.; Pollack, M.; and Cox, M. 1998. Rationale-based monitoring for continuous planning in dynamic environments. In *Proc. of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS)*, 171–179.

Appendix: Proof of Theorem 1

Definition 1. Given a situation S, a goal *Goal*, a cost formula $\mathit{Cost}(a,c)$, an admissible heuristic $\mathit{Heu}(h)$, and a list L of tuples $(g,h,\vec{\alpha})$, we call L *a consistent open list w.r.t.* S, *Goal*, *Cost, and Heu* if w.r.t. S

(i) L contains no infeasible action sequences;

(ii) any feasible action sequence is either itself contained in L, or a prefix or extension of it is, however, for any feasible sequence that satisfies *Goal*, either itself or a prefix is contained (but not an extension);

(iii) every element $(g,h,[\alpha_1,\alpha_2,\dots,\alpha_n]) \in L$ is such that h is indeed the heuristic value for $[\alpha_1,\alpha_2,\dots,\alpha_n]$ according to S, i.e. $\mathcal{D} \models \mathit{Heu}(h, do([\alpha_1,\alpha_2,\dots,\alpha_n],S))$; and

(iv) the accumulated costs g are such that $\mathcal{D} \models (\exists c_1,\dots,c_n).\mathit{Cost}(\alpha_1,c_1,S) \wedge \mathit{Cost}(\alpha_2,c_2,do(\alpha_1,S)) \wedge \cdots \wedge \mathit{Cost}(\alpha_n,c_n,do([\alpha_1,\alpha_2,\dots,\alpha_{n-1}],S)) \wedge g = c_1 + \cdots + c_n$.

Proposition 1. Any open list output by A^*, and hence *regrA**, is consistent w.r.t. to the given initial situation, goal, cost-, and heuristic function.

Lemma 1. In the sequence of invocations of Theorem 1, Open_2 is a consistent open list w.r.t. S_2, *Goal*, *Cost*, and *Heu*.
Proof: We prove each item of Definition 1 in turn.
Proof of (i): Let $(g,h,[\alpha_1,\alpha_2,\dots,\alpha_n]) \in \mathit{Open}_2$ and assume to the contrary that $[\alpha_1,\alpha_2,\dots,\alpha_n]$ is infeasible in S_2, i.e. there is an $1 \le i \le n$ such that $\mathcal{D} \not\models \mathit{Poss}(\alpha_i, do([\alpha_1,\dots,\alpha_{i-1}],S_2))$. It must be the case that $[\alpha_1,\alpha_2,\dots,\alpha_n]$ was either element of Open_1, or it was introduced by *recover*. We lead both cases to a contradiction.

In the former case the preconditions of α_i have different truth values in S_1 and S_2, since they must have been true in S_1, or else the sequence wouldn't have been included in Open_1. Hence, there must be at least one fluent F mentioned in $\mathit{Poss}(\alpha_i,s)$ for which we have that $\mathcal{D} \models F(S_1) \not\equiv F(S_2)$. Then, however, by definition of *recover* (Lines 1, 2), $([\alpha_1,\dots,\alpha_i],P)$ was included in Δ'. Since $\mathcal{D} \models \mathit{Poss}(\alpha_i, do([\alpha_1,\dots,\alpha_{i-1}],S_1))$ at that point, following the definition of *regrA** (Lines 6, 7, 8), $T([\alpha_1,\dots,\alpha_i]).p = \mathit{true}$ and $\mathcal{D} \not\models \mathit{Poss}(\alpha_i, do([\alpha_1,\dots,\alpha_{i-1}],S_2))$ (due to $T([\alpha_1,\dots,\alpha_i]).P(s) = \mathcal{R}[\mathit{Poss}(\alpha_i),[\alpha_1,\dots,\alpha_{i-1}]$ (definition of *regrA**), and the Regression Theorem (Reiter 2001)), and the fact that $[\alpha_1,\dots,\alpha_{i-1}]$ is a prefix of $[\alpha_1,\dots,\alpha_n]$, $(g,h,[\alpha_1,\alpha_2,\dots,\alpha_n])$ would be removed from the open list, concluding the contradiction for this case.

In the latter case, i.e. *recover* introduced this element, we get a contradiction just as easily. The only place where recover inserts new elements into *Open* is in Line 17, i.e. in the body of an **elseif** statement with condition $T(\vec{\alpha}).p = \mathit{false} \wedge \mathcal{D} \models T(\vec{\alpha}).P(S_2)$. This condition is violated by the assumption that $\mathcal{D} \not\models \mathit{Poss}(\alpha_i, do([\alpha_1,\dots,\alpha_{i-1}],S_2))$ (again, due to the Regression Theorem and the definition of *regrA** stating that $T([\alpha_1,\dots,\alpha_i]).P(s) = \mathcal{R}[\mathit{Poss}(\alpha_i),[\alpha_1,\dots,\alpha_{i-1}])$. Hence, Line 17 is never reached for this sequence, and thus

recover cannot have inserted this element. Hence, no infeasible action sequence is contained in $Open_2$.

Proof of (ii): Assume again to the contrary that there is an action sequence $[\alpha_1, \alpha_2, \ldots, \alpha_n]$ which is feasible in S_2, but neither itself, nor a prefix or extension of it is included in $Open_2$. This sequence (like any other) is either feasible in S_1, or infeasible in S_1. We lead both cases to a contradiction.

By Proposition 1, $Open_1$ is consistent. Hence, *recover* must have removed an appropriate sequence. However, as already seen above, *recover* only removes sequences that have a prefix whose last action is not feasible in the corresponding situation in S_2, and hence the entire sequence $[\alpha_1, \alpha_2, \ldots, \alpha_n]$ wouldn't be feasible in S_2, a contradiction.

Otherwise, if $[\alpha_1, \alpha_2, .., \alpha_n]$ is not feasible in S_1, there is a minimal $1 \leq i \leq n$ such that $\mathcal{D} \not\models Poss(\alpha_i, do([\alpha_1, .., \alpha_{i-1}], S_1))$, and we have $T([\alpha_1, .., \alpha_i]).p = false$, by Line 14 of *regrA**. However, by assumption, $\mathcal{D} \models Poss(\alpha_i, do([\alpha_1, .., \alpha_{i-1}], S_2))$ and thus also $\mathcal{D} \models T([\alpha_1, .., \alpha_i]).P(S_2)$, by definition of *regrA** and the Regression Theorem. Hence, the condition on Line 9 is satisfied for the sequence $[\alpha_1, .., \alpha_i]$, as there must be a fluent mentioned in $\mathcal{R}[Poss(\alpha_i), [\alpha_1, .., \alpha_{i-1}]]$ with opposite truth values in S_1 and S_2, so that this sequence is included in Δ'. Following the condition on Line 9 the action sequence $[\alpha_1, .., \alpha_i]$ is added to the open list, concluding the second contradiction.

Now let's turn to the second part of (ii). Assume there was an element $(g, h, [\alpha_1, \alpha_2, \ldots, \alpha_n]) \in Open_2$ such that there exists a minimal index $1 \leq i < n$ with $\mathcal{D} \models Goal(do([\alpha_1, \ldots, \alpha_i], S_2))$. Since we assume that the goal can only be achieved through the action *finish*, and when this action can execute it will always produce the goal, we know that $\alpha_i = finish$. Then however, the element $(g, h, [\alpha_1, \alpha_2, \ldots, \alpha_n])$ cannot be in $Open_1$ since otherwise also $\mathcal{D} \models Goal(do([\alpha_1, \ldots, \alpha_i], S_1))$, because *regrA** does not introduce infeasible action sequences into the open list, and since this sequence would satisfy the goal, it would not have been expanded further (cf. Proposition 1). Also, the sequence cannot have been introduced by *recover*, since *recover* only introduces sequences for whose last action the preconditions differ between S_1 and S_2 (but $i < n$). This concludes the contradiction for this part.

Proof of (iii): There are two possible cases: (a) Either $(g, h, [\alpha_1, \alpha_2, \ldots, \alpha_n])$ was added by *recover*, or (b) it was added by the first *regrA**. Case (a) leads easily to a contradiction, since due to Lines 15, 17 of *recover* only elements with correct values according to S_2 are added to the open list. Also, this element cannot be changed in further iterations of *recover* as it cannot be member of Δ'. In case (b) we have that $\mathcal{D} \models Heu(h, do([\alpha_1, \alpha_2, \ldots, \alpha_n], S_1)) \neq Heu(h, do([\alpha_1, \alpha_2, \ldots, \alpha_n], S_2))$ and hence there must be a fluent mentioned in $\mathcal{R}[Heu(h), [\alpha_1, \alpha_2, \ldots, \alpha_n]]$ on whose truth value S_1 and S_2 disagree. Therefore, the sequence $[\alpha_1, \alpha_2, \ldots, \alpha_n]$ is included in Δ' in *recover* and Lines 25–27 executed for it. Since this element is part of the open list, the **elseif**-condition holds and its new value according to S_2 is determined. This value is written back into the entry of the open list 26 (and also into the tree annotation 27). This concludes the contradiction of this case, that the h value for any element of the open list $Open_2$ is wrong w.r.t. S_2.

Proof of (iv): There are again the two possible cases: (a) $(g, h, [\alpha_1, \alpha_2, \ldots, \alpha_n])$ was added by *recover*, or (b) it was added by the first *regrA**.

In case (a) the new value for g is computed by $getGval$ on Line 16 of *recover*. This value is either accurate according to S_2, namely when all annotated costs for actions in the sequence are already with respect to S_2, in which case the contradiction is immediate, or some of them are still with respect to S_1 and are going to be fixed subsequently. In the later case, there must be actions in the sequence for which the costs according to S_1 and S_2 disagree. In each of these cases there are disagreeing fluents mentioned in the corresponding regressed cost formulae that trigger the treatment in Line 18, which will adjust g accordingly (also cf. case (b)). Hence, this cases contradicts the assumption that the value is incorrect with respect to S_2.

In case (b) there again must be a fluent F mentioned in $\mathcal{R}[Cost(\alpha_i, c), [\alpha_1, \ldots, \alpha_{i-1}]]$ for at least one $1 \leq i \leq n$ such that $\mathcal{D} \models F(S_1) \not\equiv F(S_2)$, or else the accumulated cost values for S_1 and S_2 would be the same. Hence, Lines 18–23 are executed for all such i's. In each case, the correct costs for α_i according to S_2 are computed and the offset from the previous value (w.r.t. S_2) is added to any element of the open list which has $[\alpha_1, \ldots, \alpha_i]$ as a prefix (a sequence has trivially itself as a prefix), and in particular $[\alpha_1, \alpha_2, \ldots, \alpha_n]$. Hence, after the **foreach** loop terminates, there are no more nodes along the branch to $[\alpha_1, \alpha_2, \ldots, \alpha_n]$ which show a *Cost* value that is not according to S_2, and value g reflects their sum. This completes the last contradiction. □

Proof of Theorem 1: Let us first consider the case where there is a plan for S_3, i.e. the two open lists $Open_3$ and $Open'$ aren't empty. Let the first element in $Open'$ be $(g', 0, \vec{\alpha}')$, i.e. $\vec{\alpha}'$ is an optimal plan for *Goal*, for the initial state S_2 and g' is its overall cost. The heuristic value for this element is, of course, 0, since the action sequence is a plan.

We need to show that the first element of $Open_3$, let's call it $(g_3, 0, \vec{\alpha}_3)$, is exactly the same. Since we assume that any open list output by *regrA** or *recover* is sorted according to $Value$, it suffices to show that there is a member in $Open_3$ whose action sequence is $\vec{\alpha}'$, no other element $(g_3', h_3', \vec{\alpha}_3') \in Open_3$ is such that $g_3' + h_3' < g_3$ (we assume that tie breaking is done the same way every time an open list is sorted, and omit these details here), and that $g_3 = g'$. Again, since it is output by *regrA**, the heuristic value can only be zero.

All this follows from Lemma 1, and the completeness and optimality of A^* and hence *regrA**, based on the admissibility of the heuristic function.

Now to the case where $Open'$ is empty, i.e. no feasible action sequence to reach the goal. We show that also $Open_3$ is empty. Assume to the contrary that there is an element $(g, h, [\alpha_1, \alpha_2, \ldots, \alpha_n]) \in Open_3$. This sequence satisfies the goal (ignoring preconditions), or else *regrA** would not have returned it. But then $Open_2$ must have already contained an element whose action sequence was a prefix $[\alpha_1, \alpha_2, \ldots, \alpha_i]$ of $[\alpha_1, \alpha_2, \ldots, \alpha_n]$, $i \leq n$, since *regrA** itself is assumed correct and never introduces infeasible action sequences into the open list. The contradiction now follows again from Lemma 1 (Case (i)). □