Generating Safe Assumption-Based Plans for Partially Observable, Nondeterministic Domains

Alexandre Albore, Piergiorgio Bertoli
ITC-IRST
Via Sommarive 18, 38050 Povo, Trento, Italy
{albore,bertoli}@irst.itc.it

Abstract

Reactive planning using assumptions is a well-known approach to tackle complex planning problems for nondeterministic, partially observable domains. However, assumptions may be wrong; this may cause an assumption-based plan to fail. In general, it is not possible to decide at runtime whether an assumption has failed and is putting at danger the success of the plan; thus, plan execution has to be controlled taking into account every possible success-endangering assumption failure. The possibility of tracing such failures strongly depends on the actions performed by the plan. In this paper, focusing on a simple assumption language, we provide two main contributions. First, we formally characterize *safe* assumption-based plans, i.e. plans that not only succeed whenever the assumption holds, but also guarantee that any success-endangering assumption failure is traced by a suitable monitor. In this way, replanning may be triggered only when actually needed. Second, we extend the planner in a reactive platform in order to produce safe assumption-based plans. We experimentally show that safe assumption-based (re)planning is a good alternative to its unsafe counterpart, minimizing the need for replanning while retaining the efficiency in plan generation.

Introduction

Planning for realistic domains, featuring nondeterministic action outcomes, initial state uncertainty and partial observability, is a very complex task. Using assumptions to restrict plan search in a reactive setting is a well-known approach to practically alleviate the complexity of the problem, allowing considerable scale-ups. However, assumptions taken during plan generation may turn out to be incorrect at plan execution time; if this event is not properly detected, an assumption-based plan may lead to an undesired state, or produce non-executable actions. Reactive architectures such as (Muscettola *et al.* 1998; Myers & Wilkins 1998; Singh *et al.* 2000) exploit monitoring components to trace the status of the domain; however, in general, due to the incomplete runtime knowledge on the domain state, it is not possible to decide whether an assumption is correct or not, and replanning must take place whenever a dangerous condition may have been reached because the assumption *might* be incorrect. When the assumption is actually correct, or it fails in a way not affecting the success of the plan, these replanning episodes are unnecessary and undesired; they

could be avoided by faithfully tracing success-endangering assumption failures. Whether this is possible also depends on the actions performed by the plan. The plan generation components in current reactive systems, however, do not take into account such traceability issues; as a consequence, unnecessary replanning may occur, unless restricting to consider specific domains and assumptions that are "easily monitorizable" (see e.g. (Koenig & Smirnov 1997)).

In this paper we provide two main contributions. First, we formally define *safe* assumption-based plans, i.e. plans which not only reach the goal if the assumption holds, but also guarantee that any success-endangering assumption failure is traced, by performing actions that help the monitor inspect the domain status. Second, we implement a reactive planner exploiting safe assumption-based plan generation, extending the MBP planner inside the SYPEM platform (Bertoli, Cimatti, & Traverso 2003) .

Our experiments show that safe assumption-based (re)planning is a convenient alternative to its unsafe counterpart. It introduces no major overhead at plan generation time, thus retaining the benefit of assumption-based plan generation w.r.t. strong contingency planning, and minimizes the amount of needed replanning.

We first introduce some notation, and define the main components of our planning architecture. Then we define conditions establishing whether a plan is safe, and embed them in a forward chaining planning algorithm. We present an experimental analysis, and wrap up with conclusions and future work.

Notation

We use the standard notation $\{x_1, \ldots, x_n\}$ for a set whose elements are $x_1, \ldots, x_n$. We indicate tuples with brackets, e.g. $\langle x, y \rangle$, and sequences with square brackets, e.g. $[x_1, \ldots, x_n]$. Sequences will be also indicated by overlines, e.g. $\overline{o}$ is a sequence; its length is $len(\overline{o})$. The n-th element of a sequence $\overline{o}$ is indicated with $\overline{o}^{(n)}$. Two sequences $\overline{o_1}, \overline{o_2}$ can be concatenated, written $\overline{o_1} \circ \overline{o_2}$.

The Framework

In this work, we refer to a reactive architecture which exploits assumptions, coming e.g. from a database, see Fig. 1. At plan generation, assumptions are used to restrict the search that generates a plan that controls the domain. The domain is thought of as a generic system, responding to *actions* and whose internal state is only partially visible

Figure 1: The framework.

through *observations*. The plan is an automaton; it controls the domain producing actions, based on the observations and on its internal status. At plan execution time, an automaton called a *monitor* gathers the actions applied on the domain, and its responses, to reconstruct estimates of the current domain status. Such estimates are used to evaluate whether the taken assumptions are compatible with the observed domain behaviour, and whether an action is safely executable. Based on this, replanning can be triggered, using the estimate as a new starting point.

We now provide more details on the components involved in plan execution and monitoring.

Planning Domains

As in (Bertoli *et al.* 2003), a planning domain is defined in terms of its *states*, of the *actions* it accepts, and of the possible *observations* that the domain can exhibit. Some of the states are marked as valid *initial states* for the domain. A *transition function* describes how (the execution of) an action leads from one state to possibly many different states. Finally, an *observation function* defines which observations are associated with each state of the domain.

Definition 1 (Planning domain) *A* nondeterministic planning domain with partial observability *is a tuple* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$, *where:*

- $\mathcal{S}$ *is the set of* states.
- $\mathcal{A}$ *is the set of* actions.
- $\mathcal{U}$ *is the set of* observations.
- $\mathcal{I} \subseteq \mathcal{S}$ *is the set of* initial states; *we require* $\mathcal{I} \neq \emptyset$.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to 2^{\mathcal{S}}$ *is the* transition function; *it associates with each current state* $s \in \mathcal{S}$ *and with each action* $a \in \mathcal{A}$ *the set* $\mathcal{T}(s, a) \subseteq \mathcal{S}$ *of next states.*
- $\mathcal{X} : \mathcal{S} \to 2^{\mathcal{U}}$ *is the* observation function; *it associates with each state* s *the set of possible observations* $\mathcal{X}(s) \subseteq \mathcal{U}$, *with* $\mathcal{X}(s) \neq \emptyset$.

We indicate with $[[o]]$ *the set of states compatible with the observation* o: $[[o]] = \{s \in \mathcal{S} : o \in \mathcal{X}(s)\}$. *Given a set of observations* $\mathcal{OF}$, $[[\mathcal{OF}]]$ *indicates the set of states compatible with some observation in* $\mathcal{OF}$.
We say that action α *is* executable *in state* s *iff* $\mathcal{T}(s, \alpha) \neq \emptyset$; *an action is executable in a set of states iff it is executable in every state of the set.*

Example 1

Figure 2: The printer domain.

Our domain model allows uncertainty in the initial state and in the outcome of action execution. Also, the observation associated with a given state is not unique. This allows modeling noisy sensing and lack of information.

Consider the example depicted in Fig. 2. The domain involves a robot in a corridor of five rooms; a printer is in room number 2, and the robot may be initially in any room. The printer may be empty or full, and is initially empty. The states of the domain are $1_e, 2_e, 3_e, 4_e, 5_e, 1_f, 2_f, 3_f, 4_f, 5_f$, *each corresponding to robot position and to a printer state.*

Actions $left$, $right$ *and* $refill$ *are available, with the obvious outcomes. The actions* $left$ *is executable on every state but* 1_e *and* 1_f; $right$ *is executable on every state but* 5_e *and* 5_f. *The action* $refill$ *that refills the printer is only executable when the state is* 2_e.

Three observations are available to the robot: wl, wr, wno, *indicating that there is a wall to the left, to the right, or neither way. They hold iff the robot is in the leftmost, rightmost, or in the middle rooms respectively. That is, the observation function* $\mathcal{X}$ *associates* $\{wl\}$ *with states* 1_e *and* 1_f, $\{wr\}$ *with states* 5_e *and* 5_f, *and* $\{wno\}$ *with the remaining states.*

For ease of presentation, the example only features uncertainty in the initial state; however, this is enough to force reasoning on sets of possible states, or *beliefs*, the key difficulty in handling with nondeterminism.

Goals, Assumptions, Plans

We consider reachability goals: a goal for a domain $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$ is a set of states $\mathcal{G} \subseteq \mathcal{S}$, and a problem is a pair $\langle \mathcal{D}, \mathcal{G} \rangle$, to be reached in a finite number of plan execution steps. For these problems, it is sufficient to consider acyclic plans that branch based on the sensed observation:

Definition 2 (Plan) *A plan for a domain* $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$ *is a nonempty set of pairs* $\langle \mathcal{OF}, \epsilon \rangle$ *and triples* $\langle \mathcal{OF}, \alpha, \pi \rangle$ *where:*

- $\mathcal{OF} \in 2^{\mathcal{U}}$, $\alpha \in \mathcal{A}$, *and* π *is a plan.*
- ϵ *is the empty plan; when reached by the execution of a plan* π, *it indicates* π's *termination.*

Each pair and triple is called a branch of the plan. The sets of observations in the branches of a plan must cover $\mathcal{U}$.

A plan is essentially an acyclic graph whose paths from the root to the leafs are sequences of the form $[\mathcal{OF}_1, \alpha_1, \ldots, \alpha_{n-1}, \mathcal{OF}_n]$, built by recursive concatenation of the observation sets and actions in the branches of the plan. We allow for some forms of syntactic sugar:

- We will use an if-then-else syntax to indicate plans with two branches; for instance, the plan $if \ \{o_1, o_2\} \ then \ \alpha_1; \pi_1 \ else \ \alpha_2; \pi_2$ corresponds to $\{\langle \{o_1, o_2\}, \alpha_1, \pi_1 \rangle, \langle \mathcal{U} - \{o_1, o_2\}, \alpha_2, \pi_2 \rangle\}$.
- $\alpha; \pi$ is equivalent to $\{\langle \mathcal{U}, \alpha, \pi \rangle\}$

- ϵ. is equivalent to $\{\langle \mathcal{U}, \epsilon \rangle\}$, and α. is equivalent to $\alpha; \epsilon$.

Example 2 *Here is a plan P_0 for the above domain:*
```
if wl
    then right; refill.
    else left; if wl
                then right; refill.
                else left; if wl
                            then right; refill.
                            else left; if wl
                                        then right; refill.
                                        else refill.
```

Given a planning problem, two key features for a plan are (1) its executability given possible initial states, and (2) whether the plan solves the problem, by ending its execution in a goal state. To provide these notions, we introduce the possible traces of the plan, i.e. the sequences of observations, actions and states that can be traversed executing the plan from an initial state.

Definition 3 (Traces of a plan.) *A trace of a plan is a sequence*

$$[s_0, o_0, \alpha_0, \ldots, s_{n-1}, o_{n-1}, \alpha_{n-1}, s_n, o_n, End]$$

where s_i, o_i are the domain state and associated observation at step i of the plan execution, and α_i is the action produced by the plan at step i on the basis of o_i (and of the plan's internal state). The final symbol End can either be $Stop$, indicating that the plan has terminated, or $Fail(\alpha)$, indicating execution failure of action α on s_n. A trace is a failure trace *iff it terminates with $Fail(\alpha)$. A trace is a* goal trace *for a set of states $\mathcal{G}$ iff it is not a failure trace, and its final state is in $\mathcal{G}$. The predicate $Reaches(t, \mathcal{G})$ holds true on a trace t iff t is a goal trace for $\mathcal{G}$. We also indicate a trace with $\langle \overline{s}, \overline{o}, \overline{\alpha} \rangle$, splitting it into a state trace, an action sequence, and an observation trace, and omitting the final symbol.*

We indicate with $Trs(\pi, s)$ the set of traces that can be generated by plan π starting from a state s. This set can be easily built by recursion on the plan structure, taking into account executability of actions on the states. The set of traces $Trs(\pi, B)$ for plan π starting from a belief B is the union of the traces from every state $s \in B$.

At this point, the notions of executability and solution can easily be provided as follows:

Definition 4 (Executable plan) *A plan π is executable on state s (on a set of states B) iff no failure trace exists in $Trs(\pi, s)$ ($Trs(\pi, B)$ resp.).*

Definition 5 (Solution) *A plan π is a strong solution for a problem $\langle \mathcal{D}, \mathcal{G} \rangle$ iff every trace from $\mathcal{I}$ is a goal trace for $\mathcal{G}$:*
$$\forall t \in Trs(\pi, \mathcal{I}) : Reaches(t, \mathcal{G})$$

Example 3 *Consider again the domain of Fig. 2, and plan P_0 from the previous example. The plan is a strong solution for the problem of reaching the filled printer status 2_f, since $Trs(\pi_0, \{1_e, 2_e, 3_e, 4_e, 5_e\}) = \{t_1, t_2, t_3, t_4, t_5\}$, where:*

$t_1 = [1_e, wl, right, 2_e, wno, refill, 2_f, wno, Stop]$,

$t_2 = [2_e, wno, left] \circ t_1$,

$t_3 = [3_e, wno, left] \circ t_2$,

$t_4 = [4_e, wno, left] \circ t_3$, *and*

$t_5 = [5_e, wr, left, 4_e, wno, left, 3_e, wno, left,$
$\quad 2_e, wno, refill, 2_f, wno, Stop]$

Assumptions on the domain behaviour are used to simplify a problem, narrowing the search. In this paper, we consider assumptions that restrict the initial state of the domain:

Definition 6 (Assumption) *An assumption for a domain $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$ is a set of states $\mathcal{F} \subseteq 2^{\mathcal{S}}$, intended to restrict the possible initial states of $\mathcal{D}$ to $\mathcal{F} \cap \mathcal{I} \neq \emptyset$.*

The above notions of executable plan and solution plan lift naturally to take into account the assumptions:

Definition 7 (Executable/Solution under assumption) *A plan π is executable for a set of states B under assumption $\mathcal{F}$ iff it is executable for the set of states $B \cap \mathcal{F}$.*

A plan π is a strong solution for problem $\langle \mathcal{D}, \mathcal{G} \rangle$ under assumption $\mathcal{F}$ iff it is a strong solution for $\langle \mathcal{D}_A, \mathcal{G} \rangle$ where $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$ and $\mathcal{D}_A = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I} \cap \mathcal{F}, \mathcal{T}, \mathcal{X} \rangle$.

Naturally, executability under no assumptions implies executability under any assumption, but not vice versa. The same holds for the notion of solution: a plan which is a solution under a given assumption is not, in general, a solution, or even only executable.

Example 4 *Consider again the printer domain, and consider the assumption $\{1_e, 2_e, 3_e\}$. Given this, the following plan P_1 is an assumption-based solution to fill the printer:*
```
if wl
    then right; refill.
    else left; if wl
                then right; refill.
                else refill.
```
However, this plan is not a strong solution, nor executable. In fact, it is "dangerous to execute": if the assumption is wrong and the initial state is 4_e, the sequence of actions $[left, refill]$ is attempted, which is not executable.

Monitoring

In order to monitor assumptions on initial states, and to produce the necessary information for replanning, we exploit *universal monitors*, defined in (Bertoli *et al.* 2003). A universal monitor is an automaton which embeds a faithful model of the domain, and evolves it on the basis of the received actions and observations, which constitute the I/O of the actual domain. In this way, at each execution step i, the monitor evolves, and provides as output, a belief M_i on the domain status. The belief M_i consists of the set of all states compatible with the domain behaviour observed up to step i, and with the domain being initially in some $s \in M_0$. Thus M_i is empty if the domain behavior is not compatible with the initial state being in M_0, indicating that the domain status was initially outside M_0. Based on these properties, given a domain $\mathcal{D} = \langle \mathcal{S}, \mathcal{A}, \mathcal{U}, \mathcal{I}, \mathcal{T}, \mathcal{X} \rangle$, the output of a monitor whose initial belief is $\mathcal{I}$ can be used to check whether an action is guaranteed to be executable, and as a new initial belief for replanning. Moreover, given an assumption-based plan π that reaches the goal iff the initial domain status belongs to $\mathcal{I}_\pi \subseteq \mathcal{I}$, a monitor whose initial belief is $\mathcal{I}_\pi$ can be used to detect whether the initial state is actually outside $\mathcal{I}_\pi$, preventing π's success [1]. Thus the

---

[1] In general, an assumption-based solution π for an assumption $\mathcal{F}$ may reach the goal $\mathcal{G}$ starting from a set of initial states

monitoring in our architecture exploits a pair of universal monitors, whose initial beliefs are $\mathcal{I}$ and $\mathcal{I}_\pi$ respectively.

Generation of safe assumption-based plans

Our aim is to generate *safe* assumption-based plans, i.e. plans that guarantee that (a) whenever the assumption holds, the goal is reached, and (b) whenever an assumption failure inhibits success, this is signalled by the monitor, preventing possible execution failures (as opposed to *unsafe* assumption-based plans). These constraints can be restated based on traces, requiring that (a) any trace starting from $\mathcal{I} \cap \mathcal{F}$ is successful, and (b) any trace from $\mathcal{I} \cap \neg\mathcal{F}$ is either successful, or distinguishable from those from $\mathcal{I} \cap \mathcal{F}$ by monitoring. This can be formally expressed by defining the observation sequences compatible with an initial state/belief, given an action sequence $\overline{\alpha}$. For this purpose, we construct the observation sequences that may take place when executing $\overline{\alpha}$ starting from a state s. We first observe that $\overline{\alpha}$ induces a set of possible sequence of states traversed by the domain during the execution; in turn, each state sequence traversed by the domain can be associated with a set of possible observation sequences for it:

Definition 8 (Induced state traces) *Given an action sequence* $\overline{\alpha} = [\alpha_1, \ldots, \alpha_n]$, *the set of state traces induced by the execution of* $\overline{\alpha}$ *on s, is the set of state traces related to the execution of plan* $\alpha_1; \ldots; \alpha_n$:

$$sTraces(s, \overline{\alpha}) = \{\overline{s} : \langle \overline{s}, \overline{o}, \overline{\alpha} \rangle \in Trs(s, \alpha_1; \ldots; \alpha_n)\}$$

Definition 9 (Compatible observations) *Given a sequence of states* $\overline{s}$, *the set of compatible sequences of observations* $\overline{\mathcal{X}}(\overline{s})$ *is defined as follows:*

$$\overline{\mathcal{X}}(\overline{s}) = \{\overline{o} : \overline{o}^{(i)} \in \mathcal{X}(\overline{s}^{(i)})\}$$

Definition 10 (Observations sequences compatible) *with initial state/belief, given an action sequence.*

$$\overline{\mathcal{X}}_{\overline{A}}(s, \overline{\alpha}) = \bigcup_{\overline{s} \in sTraces(s, \overline{\alpha})} \overline{\mathcal{X}}(\overline{s})$$

$$\overline{\mathcal{X}}_{\overline{A}}(B, \overline{\alpha}) = \bigcup_{s \in B} \overline{\mathcal{X}}_{\overline{A}}(s, \overline{\alpha})$$

We can thus define safe plans, recalling that, given a trace $\langle \overline{s}, \overline{o}, \overline{\alpha} \rangle$, a monitor whose initial belief is M_0 detects whether $\overline{o} \notin \overline{\mathcal{X}}_{\overline{A}}(M_0, \overline{\alpha})$:

Definition 11 (Safe assumption-based solution) *A plan π is a safe assumption-based solution iff:*

1. $\forall \langle \overline{s}, \overline{o}, \overline{\alpha} \rangle \in Trs(\pi, \mathcal{I} \cap \mathcal{F}) : Reaches(\langle \overline{s}, \overline{o}, \overline{\alpha} \rangle, \mathcal{G})$, *and*

2. $\forall \langle \overline{s}, \overline{o}, \overline{\alpha} \rangle \in Trs(\pi, \mathcal{I} \cap \neg\mathcal{F}) : (Reaches(\langle \overline{s}, \overline{o}, \overline{\alpha} \rangle, \mathcal{G}) \vee$ $\overline{o} \notin \overline{\mathcal{X}}_{\overline{A}}(\mathcal{I} \cap \mathcal{F}, \overline{\alpha}))$

Example 5 *The strong plan P_0 from example 2 is safe for the assumption* $\mathcal{F} = \{1_e, 2_e, 3_e\}$: *in the case of success-endangering assumption failures (i.e. when the initial status is 4_e or 5_e), executing P_0 cannot lead to any observation*

$\mathcal{I}_\pi \supseteq \mathcal{I} \cap \mathcal{F}$. $\mathcal{I}_\pi$ can be efficiently computed by symbolic backward simulation of π from $\mathcal{G}$. The states in $\mathcal{I}_\pi / (\mathcal{I} \cap \mathcal{F})$ represent failures of $\mathcal{F}$ which do not affect the success of π.

sequence identical to one which would stem by applying the same actions from 1_e, 2_e or 3_e. For instance, both starting from 3_e and 4_e, the first two actions executed by P_0 are $[left, left]$; *but in one case, the wl observation holds true after that, while in the other it does not.*

The assumption-based solution P_1 is not safe for the same assumption $\mathcal{F}$: both starting from 3_e and 4_e, the same observations are produced before the hazardous $refill$ action is tried.

Consider the following plan P_2, which goes along P_1's line but executes an additional $[left, right]$ sequence before filling the printer:

    if wl
        then right; refill.
        else left; if wl
                then right; refill.
                else left; right; refill.

P_2 is a safe assumption-based solution: just as for P_0, monitoring will detect success-endangering assumption failures, since the actions produced by P_2 in those cases induce observation sequences incompatible with the domain being initially in 1_e, 2_e or 3_e. E.g. if the initial status is 4_e, observation wno holds after executing $[left, left]$, while wl holds iff the status was 3_e.

We now use the above definitions to prune the search in a forward-chaining search algorithm. In particular, we consider the plan generation approach presented in (Bertoli, Cimatti, & Roveri 2001), where an and-or graph representing an acyclic prefix of the search space of beliefs is iteratively expanded: at each step, observations and actions are applied to a fringe node of the prefix, removing loops. Each node in the graph is associated with a belief in the search space, and with the path of actions and observations that is traversed to reach it. When a node is marked success, by goal entailment or by propagation on the prefix, its associated path is eligible as a path of a solution plan.

To constrain the algorithm to produce safe assumption-based plans, success marking of a node must also require that the associated path obeys a safety condition [2]. To achieve this, we evaluate safety considering the set of traces associated with a path, i.e. those traces traversing the path:

Definition 12 (Traces associated with path) *Given a path* $\langle \overline{\mathcal{OF}}, \overline{\alpha} \rangle$ *and a trace* $\langle \overline{s}, \overline{o}, \overline{\beta} \rangle$, *we say that the trace belongs to the path iff* $\overline{\alpha} = \overline{\beta}$, *and* $\forall i \in \{1, \ldots, len(\overline{o})\} : \overline{o}^{(i)} \in \overline{\mathcal{OF}}^{(i)}$. *Given a plan π, a path p of π, and an initial set of states B, we write $Trs(p, B)$ to indicate the set of traces in $Trs(\pi, B)$ that belong to p.*

Thus, a path p can be accepted as a part of a safe plan π for problem $\langle \mathcal{D}, \mathcal{G} \rangle$ iff requirements 1 and 2 from Def. 11 are obeyed, replacing π with p. Req.1 is easily computed symbolically by a $Exec_P$ operator that progresses the belief $\mathcal{I} \cap \mathcal{F}$ on the path p, obtaining the final belief for the path:

$$Exec_P(\mathcal{I} \cap \mathcal{F}, p) \subseteq \mathcal{G} \qquad (1)$$

Req.2 requires, instead, checking that for each non-success trace in $Trs(p, \mathcal{I} \cap \neg\mathcal{F})$ and each trace in $Trs(p, \mathcal{I} \cap \mathcal{F})$,

---

[2]At the same time, loop checking is relaxed: a loop only takes place when the same belief *and* associated safety condition are met.

the observation components do not match. This can be extremely expensive, due to the huge number of paths. To avoid this expensive computation, we build the path annotation symbolically, by pruning at each step of the path those trace pairs that can be distinguished based on current observations. We achieve this by a function $EqPairs(B_1, B_2, p)$ which produces a set of pairs of beliefs, each implicitly associated with a sequence of observations $\overline{o} \in \overline{\mathcal{X}}_{\overline{A}}(B_1, p) \cap \overline{\mathcal{X}}_{\overline{A}}(B_2, p)$. Each pair overestimates the undistinguishable states originated by $\overline{o}$ starting from B_1 and B_2. As a special case, $EqPairs$ returns $\{\langle \mathcal{S}, \mathcal{S} \rangle\}$ to indicate that the executability of the path is not guaranteed, due to a possibly non-executable action (such as $refill$ in example 4). The results of $EqPairs(\mathcal{I} \cap \mathcal{F}, \mathcal{I} \cap \neg\mathcal{F}, p)$, can be used to state that no undetectable assumption failure (if any) evolves outside the goal, defining a sufficient *traceability* condition[3]:

$$\forall \langle B, B' \rangle \in EqPairs(\mathcal{I} \cap \mathcal{F}, \mathcal{I} \cap \neg\mathcal{F}, p) : B' \subseteq \mathcal{G} \quad (2)$$

The definition of $EqPairs$ exploits a $prune$ operator that, given a pair $\langle B_1, B_2 \rangle$ of nonempty beliefs, and a set of observations $\mathcal{OF}$, considers every observation $o \in \mathcal{OF}$ which may take place over both beliefs, and builds an associated belief pair restricting B_1 and B_2 to o, i.e. to those parts of B_1 and B_2 that cannot be distinguished if o takes place:

$$prune(B_1, B_2, \mathcal{OF}) =$$
$$\{\langle B_1 \cap [[o]], B_2 \cap [[o]] \rangle : o \in \mathcal{X}(B_1) \cap \mathcal{X}(B_2) \cap \mathcal{OF}\}$$

In particular, given a plan path $[\mathcal{OF}_0, \alpha_0, \mathcal{OF}_1, \ldots, \alpha_{n-1}, \mathcal{OF}_n]$, $EqPairs(B_1, B_2, p)$ is defined by repeatedly evolving the initial beliefs via the observations and actions in the path, and splitting them via the $prune$ operator:

- $EqPairs(B_1, B_2, [\mathcal{OF}_0] \circ p) =$
  $EqPairs_0(prune(B_1, B_2, \mathcal{OF}_0), p)$

- Let $bp = \{\langle B_1, B_1' \rangle, \ldots, \langle B_n, B_n' \rangle, \}$ be a set of pairs of beliefs. Then:
  - $EqPairs_0(bp, []) = bp$
  - If, for some belief of bp, α is not executable, then $EqPairs_0(bp, [\alpha, \mathcal{OF}] \circ p) = \{\langle \mathcal{S}, \mathcal{S} \rangle\}$;
  - otherwise,

    $$EqPairs_0(bp, [\alpha, \mathcal{OF}] \circ p) =$$
    $$EqPairs_0(\bigcup_{i=1..n} prune(\mathcal{T}(B_i, \alpha), \mathcal{T}(B_i', \alpha), \mathcal{OF}), p)$$

Thus, a sufficient acceptance condition for a path inside a safe assumption-based plan conjoins conditions (1) and (2). As a further optimization, to avoid the blow-up in the size of the pairs set, belief pairs from $prune$ can be pairwise unioned. This further strengthens the traceability condition.

Experimental evaluation

Our experiments intend to evaluate the overhead of adding safety conditions to plan generation, and the impact of safe

---

[3]Traceability is stronger than trace-based safety (Def. 11). Intuitively, it only distinguishes state traces based on their last state, and on traceability of their prefixes. That is, we pay space-saving with a limited form of incompleteness.
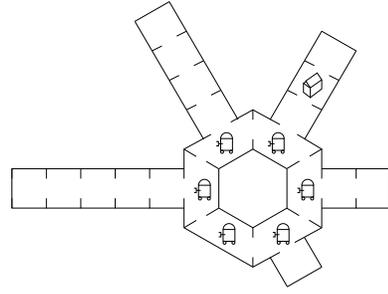

Figure 3: The test domain (for $N = 6$).

plan generation within a reactive framework such as the one we presented. For these purposes, we obtained the SYPEM reactive platform (Bertoli, Cimatti, & Traverso 2003), and modified the MBP planner inside it so that it performs safe assumption-based plan generation. We name our MBP extension SAMBP (Safe Assumption-based MBP), and SAPEM the associated extension to SYPEM. No comparison is taken w.r.t. other systems, since, to the best of our knowledge, no other approach to generate safe-assumption based plans has been presented so far.

An instance of our test domain is shown in Fig. 3; it consists of a ring of N rooms, where each room $n \neq 0$ is connected to a corridor of length n. In the middle of corridor connected to room $\lfloor N/2 \rfloor$, there is a printer which has to be refilled. The robot can be initially anywhere in the ring of rooms; it can traverse the ring by going left or right, step in or out from corridors, and fill the printer. Actions have the obvious applicability constraints, e.g. the robot cannot fill the printer unless at the printer's place. The robot is equipped with a "bump" sensor that signals whether it has reached the end of a corridor. We experiment with assumptions of different strength; in the weaker case (a1), we assume the robot can be anywhere in the ring; in (a2) we assume it is not in room 0; in (a3),(a4),(a5) we assume it is in a room within $\lfloor N/2 \rfloor \pm \Delta/2$, with $\Delta = \lfloor N/4 \rfloor$, $\Delta = 2$, and $\Delta = 0$ respectively. For any assumption, and for a range of sizes of the domain, we run the following tests on a 700 MHz Pentium III PC with 512 MBytes of memory:

1. Generation of unsafe assumption-based plans (by MBP, restricting the initial belief, see Def.9), and of safe assumption-based plans by SAMBP; we also generate strong plans for a reference;

2. Reactive plan generation and execution by SYPEM and SAPEM. We test every initial configuration where the assumption holds true, and every initial configuration where it fails (average timings are indicated with (ok) and (ko) resp.). To ease the comparison, we use no assumption when replanning.

Fig. 4 shows the results for assumptions (a3-a5); for assumptions (a1) and (a2), the behavior is essentially the same for every setting, converging to that of strong planning. Also, SYPEM and SAPEM always replan when the assumption fails, and their performance never differs for more than 20% (SAPEM scoring better); we collapse them for sake of clarity. Finally, since, when the assumption holds, SYPEM never replans, its behavior and performance coincides with that of SAMBP. We observe the following:
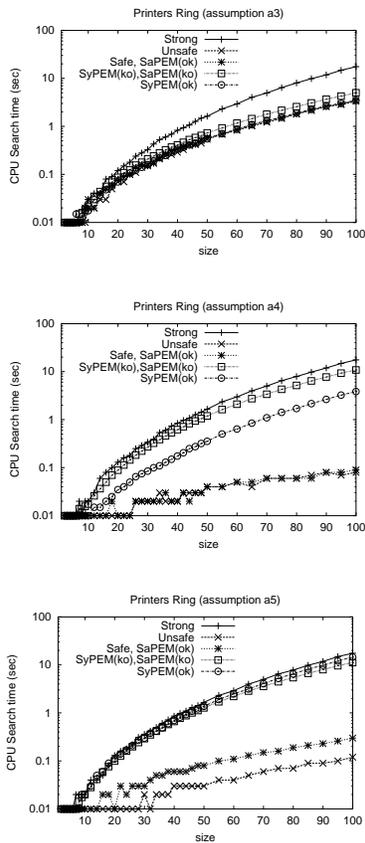
Figure 4: Experimental results for the printer domain.

Assum. on D	Unsafe	Safe, SaPEM(ok)	SyPEM(ok)	SyPEM(ko), SaPEM(ko)	Stg.
≤ 10	0.09	0.12	0.12 (19%)	0.43	1.21
≤ 5	0.02	0.03	0.11 (50%)	0.50	1.21
≤ 4	0.02	0.03	0.15 (71%)	0.49	1.21
≤ 3	0.02	0.02	0.20 (100%)	0.49	1.21
≤ 2	0.01	0.02	0.22 (100%)	0.49	1.21
≤ 1	0.01	0.02	0.23 (100%)	0.50	1.21
$= 0$	0.01	0.02	0.26 (100%)	0.53	1.21

- The overhead of safe plan generation vs. its unsafe counterpart is always limited. The bigger difference (within a factor of 3) takes place for the more restrictive assumption (a5), which, on the other side, is when assumptions are most effective to speed up the search, but unsafe assumption-based plans are most likely to fail. For weaker assumptions, (safe and unsafe) assumption-based plan generation tends to assimilate to strong planning, with no additional overhead paid by safe plan generation vs. its unsafe version.

- When the assumption holds, SAPEM avoids any replanning, while SYPEM has to replan considering possible assumption failures. The stronger the assumption, the more a SYPEM plan is likely to trigger replanning. In fact, for (a4) SYPEM replans 50% of the times, and for (a5) it replans 100% of the times. This reflects into SAPEM performing better than SYPEM. For weak assumptions, both systems tend towards the behavior of strong offline planning.

We obtained qualitatively similar results for a 31×31 maze positioning problem from (Bertoli, Cimatti, & Roveri 2001), considering assumptions related to the distance D from the goal. The following table synthesizes the timings in seconds, and, for SYPEM, the percentage of cases in which useless replanning takes place, the assumption being actually true.

Conclusions and future work

In this paper, we introduced safe assumption-based plan generation as a mean to minimize the amount of replanning required in a reactive setting using assumptions. Our experiments show that this is a viable way to effectively use assumptions in such setting, coupling the high efficiency of assumption-based plan generation with non-redundant replanning. Our work is related and complementary to those on diagnosability of controllers in (Cimatti, Pecheur, & Cavada 2003), where verification of diagnosability of a given controller is tackled given a set of admissible environment behaviors, simulating the behaviour of twin plants.

This work leads to several extensions. In particular, a more qualitative evaluation of plan safety may relax safety requirements for domains where poor or unreliable sensing makes "full" safety impossible; still, this could lead to plans which improve on unsafe plans by avoiding a substantial amount of useless replanning. Moreover, we are working at extending our concepts to a more expressive assumption language, exploiting LTL temporal logics.

References

Bertoli, P.; A., C.; M., P.; and P., T. 2003. A Framework for Planning with Extended Goals under Partial Observability. In *Proc. of ICAPS'03*. AAAI Press.

Bertoli, P.; Cimatti, A.; and Roveri, M. 2001. Conditional Planning under Partial Observability as Heuristic-Symbolic Search in Belief Space. In *Proc. of ECP'01*.

Bertoli, P.; Cimatti, A.; and Traverso, P. 2003. Interleaving Execution and Planning via Symbolic Model Checking. In *Proc. of ICAPS'03 Workshop on Planning under Uncertainty and Incomplete Information*.

Cimatti, A.; Pecheur, C.; and Cavada, R. 2003. Formal verification of diagnosability via symbolic model checking. In *Proc. of IJCAI'03*.

Koenig, S., and Smirnov, Y. 1997. Sensor-Based Planning with the Freespace Assumption. In *Proc. of ICRA '97*.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence* 103(1-2):5–47.

Myers, K. L., and Wilkins, D. E. 1998. Reasoning about Locations in Theory and Practice. *Computational Intelligence* 14(2):151–187.

Singh, S.; Simmons, R.; Smith, T.; Stentz, A. T.; Verma, V.; Yahja, A.; and Schwehr, K. 2000. Recent progress in local and global traversability for planetary rovers. In *Proc. of ICRA-2000*. IEEE.