A Framework for Planning in Continuous-time Stochastic Domains

Håkan L. S. Younes
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
lorens@cs.cmu.edu

David J. Musliner
Honeywell Laboratories
3660 Technology Drive
Minneapolis, MN 55418, U.S.A.
david.musliner@honeywell.com

Reid G. Simmons
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
reids@cs.cmu.edu

Abstract

We propose a framework for policy generation in continuous-time stochastic domains with concurrent actions and events of uncertain duration. We make no assumptions regarding the complexity of the domain dynamics, and our planning algorithm can be used to generate policies for any discrete event system that can be simulated. We use the continuous stochastic logic (CSL) as a formalism for expressing temporally extended probabilistic goals and have developed a probabilistic anytime algorithm for verifying plans in our framework. We present an efficient procedure for comparing two plans that can be used in a hill-climbing search for a goal-satisfying plan. Our planning framework falls into the Generate, Test and Debug paradigm, and we propose a transformational approach to plan generation. This relies on effective analysis and debugging of unsatisfactory plans. Discrete event systems are naturally modeled as generalized semi-Markov processes (GSMPs). We adopt the GSMP as the basis for our planning framework, and present preliminary work on a domain independent approach to plan debugging that utilizes information from the verification phase.

Introduction

Realistic domains for autonomous agents present a broad spectrum of uncertainty, including uncertainty in the occurrence of exogenous events and in the outcome of agent selected actions. In order to ensure satisfactory performance by a single agent, or a system of agents, in such domains, we need to take the uncertainty into account when generating plans.

The problem of planning under uncertainty has been addressed by researchers in operations research, artificial intelligence (AI), and control theory, among others. Numerous approaches have been proposed, but as Bresina *et al.* (2002) recently pointed out, current methods for planning under uncertainty cannot adequately handle planning problems with either concurrent actions and events or uncertainty in action durations and consumption of continuous resources (e.g. power). In this paper we focus on domains with concurrent events and actions with uncertain duration/delay. We present a framework for planning in such domains that falls into the Generate, Test and Debug (GTD) paradigm proposed by Simmons (1988).

We limit our attention to planning domains that can be modeled as *discrete event systems*. The state of such systems changes only at discrete time instants, which could be either at the occurrence of an event or the execution of an action. Many man-made dynamic systems (e.g. production or assembly lines, air traffic control systems, and robotic systems) can be realistically modeled as discrete event systems. We present a general algorithm for generating (partial) stationary policies for discrete event systems, only requiring that we can generate sample execution paths for the systems. The sample execution paths can be generated through discrete event simulation (Shedler 1993).

Drummond & Bresina (1990) recognize the need for maintenance and prevention goals in realistic planning problems, in addition to the traditional planning goals of achievement. We embrace their view, and adopt the *continuous stochastic logic* (CSL) (Aziz *et al.* 2000; Baier, Katoen, & Hermanns 1999) as a formalism for expressing *temporally extended* goals in continuous-time domains. Recent advances in probabilistic verification have resulted in efficient algorithms for verifying CSL properties of continuous-time stochastic systems (Baier, Katoen, & Hermanns 1999; Infante López, Hermanns, & Katoen 2001; Younes & Simmons 2002), but these results have not, to our best knowledge, been used to aid probabilistic plan generation. The work by Younes & Simmons is based on statistical sampling techniques, and therefore handles any model that can be simulated—in particular discrete event systems. In this paper, we present an *anytime* (Dean & Boddy 1988) version of that algorithm for a relevant subset of CSL. We also present an efficient sampling-based algorithm for comparing two plans that can be used in a hill-climbing search for a satisfactory plan.

We propose a *transformational* approach to plan generation. The simulation traces generated during plan verification can be used to guide plan repair. We recognize the need for good search control in order to make the planning algorithm practical. For this purpose, we adopt the *generalized semi-Markov process* (GSMP) as a domain model. The GSMP, first introduced by Matthes (1962), captures the essential dynamical structure of a discrete event system (Glynn 1989). As the name suggests, GSMPs are a generalization of semi-Markov processes (SMPs), allowing for the delay of events and actions to depend not only on the current state of a system, but possibly on the *entire path* taken to that state. The GSMP provides a natural mechanism for modeling concurrency and uncertainty in the duration of actions and events. We can view a GSMP as the composition of concurrent SMPs. When composing concurrent Markov processes the result is a Markov process due to the memoryless property of the exponential distribution, but composition of multiple SMPs requires a more complex model for the whole than for the parts.

In contrast to most previous research on planning under uncertainty, we choose to work with time as a continuous, rather than discrete, quantity as this is a more realistic model of time for real-time systems with concurrent asynchronous events (Alur, Courcoubetis, & Dill 1993). Doing so, we avoid the state-space explosion that comes from using discrete-time models for domains that are inherently continuous.

Our planning algorithm has been partially implemented within the CIRCA framework (Musliner, Durfee, & Shin 1995).

Background

Current approaches to planning under uncertainty can be divided roughly into distinct categories based on their representation of uncertainty, how goals are specified, the model of time used, and assumptions made regarding observability.

Two prevalent representations of uncertainty are *non-deterministic* and *stochastic* models. In non-deterministic models, uncertainty is represented strictly logically, using disjunction, while in stochastic models uncertainty is specified with probability distributions over the possible outcomes of events and actions. The objective when planning with non-deterministic models is typically to generate a *universal plan* (Schoppers 1987) that is guaranteed to achieve a specified goal regardless of the actual outcomes of events and actions. A goal can be a set of desirable states as in the work of Cimatti, Roveri, & Traverso (1998) and Jensen & Veloso (2000), or a modal temporal logic formula as proposed by Kabanza, Barbeau, & St-Denis (1997) and Pistore & Traverso (2001).

Ginsberg (1989) questions the practical value of universal non-deterministic planning. His main concern is that the representation of a universal plan is bound to be infeasibly large for interesting problems. It is impractical, Ginsberg argues, for an agent to precompute its response to every situation in which it might find itself simply because the number of situations is typically prohibitively large. In control theory, Balemi *et al.* (1993) have proposed the use of Ordered Binary Decision Diagrams (Bryant 1986) as a compact representation of supervisory controllers, and this representation has more recently also been used in the AI community for non-deterministic planning (Cimatti, Roveri, & Traverso 1998; Jensen & Veloso 2000). Kabanza, Barbeau, & St-Denis (1997) address the time complexity problem by proposing an incremental algorithm for constructing partial policies that, if given enough time, produces a universal plan, but they rely on domain-specific search control rules for efficiency.

Probabilistic planning can be seen as an attempt to address some of the complexity issues in non-deterministic planning. By requiring a stochastic domain model, a probabilistic planner has a more detailed model of uncertainty to work with, and can choose to focus planning effort on the most relevant parts of the state space. A plan may fail because some contingencies have not been planned for, but this is acceptable as long as the success probability of the plan is high. Drummond & Bresina (1990) present an anytime algorithm for generating partial policies with high probability of achieving goals expressed using a modal temporal logic. Other research on probabilistic planning typically considers only propositional goals. Kushmerick, Hanks, & Weld (1995) and Lesh, Martin, & Allen (1998) work with plans consisting of actions that are executed in sequence regardless of the outcome of the previous actions. Conditional probabilistic plans (Blythe 1994; Draper, Hanks, & Weld 1994; Goldman & Boddy 1994) allow for some adaptation to the situation during plan execution. In the work by Draper, Hanks, & Weld, this adaptation is obtained by means of explicit sensing actions that are made part of the plan.

Sampling techniques have been used for probabilistic plan assessment by Blythe (1994) and Lesh, Martin, & Allen (1998). Our approach differs from previous work using sampling in that we never need to compute any probability estimates. Instead we rely on efficient statistical hypothesis testing techniques, which saves significant effort.

Most of the work in planning under uncertainty assumes a simple model of time, with time progressing in fixed discrete steps at the occurrence of events or the execution of actions. Such time models are not sufficient to accurately capture the dynamics of many realistic real-time systems with concurrent action and event delays. Musliner, Durfee, & Shin (1995) view time as a continuous quantity, and present a planning algorithm for non-deterministic domains with a rich temporal model where action and event delays are specified as intervals. Their domain model is basically a continuous-time GSMP. While both discrete-time and continuous-time SMPs have been used as domain models in decision-theoretic planning (Howard 1971), GSMPs have not previously been considered in a probabilistic or decision-theoretic planning framework, as far as we know. Planning with concurrent temporally extended actions in discrete-time domains is considered by Rohanimanesh & Mahadevan (2001), but they place restrictions on the action dynamics so as to make the domain models semi-Markovian.

Policy search aided by simulation has been successfully applied to decision-theoretic planning with partially observable Markov decision processes (Kearns, Mansour, & Ng 2000; Ng & Jordan 2000; Baxter & Bartlett 2001).

Planning Framework

We now present a general framework for probabilistic planning in stochastic domains with concurrent actions and events. We will use a variation of the transportation domain developed by Blythe (1994) as an illustrative example. The objective of the example problem is to transport a package

load-taxi(pgh-taxi, CMU)
drive(pgh-taxi, CMU, pgh-airport)
unload-taxi(pgh-taxi, pgh-airport)
load-airplane(plane, pgh-airport)
fly(plane, pgh-airport, msp-airport)
unload-airplane(plane, msp-airport)
load-taxi(msp-taxi, msp-airport)
drive(msp-taxi, msp-airport, Honeywell)
unload-taxi(msp-taxi, Honeywell)

Figure 1: Initial plan for transporting a package from CMU to Honeywell.

from CMU in Pittsburgh to Honeywell in Minneapolis, and with probability at least 0.9 have the package arrive within five hours without losing it on the way.

The package can be transported between the two cities by airplane, and between two locations within the same city by taxi. There is one taxi in each city. The Pittsburgh taxi is initially at CMU, while the Minneapolis taxi is at the airport. There is one airplane available, and it is initially at the Pittsburgh airport. Given these initial conditions, Figure 1 gives a sequence of actions that if executed can take the package from CMU to Honeywell.

The plan in Figure 1 may fail, however, due to the effects of exogenous events and uncertainty in the outcome of planned actions. The Minneapolis taxi may be used by other customers while the package is being transported to Minneapolis, meaning that the taxi may not be at the airport when the package arrives there. The package may be lost at an airport if it remains there for too long without being securely stored. The plane may already be full when we arrive at the airport unless we have made a reservation. Figure 2 gives a PDDL-like specification of the load-airplane action. If the action is executed in a state where the atomic proposition "have-reservation" is false, then there is a 0.1 probability of there not being room for the package on the plane. Finally, there is uncertainty in the duration of actions (e.g. drive and fly) and the timing of exogenous events.

Discrete Event Systems

The planning domain introduced above can be modeled as a discrete event system. A discrete event system, $\mathcal{M}$, consists of a set of states S and a set of events E. At any point in time, the system occupies some state $s \in S$. The system remains in a state s until the occurrence of an event $e \in E$, at which point the system instantaneously transitions to a state s' (possibly the same state as s). We divide the set of events into two disjoint sets E_{a} and E_{e}, $E = E_{\mathrm{a}} \cup E_{\mathrm{e}}$, where E_{a} is the set of actions (or controllable events) and E_{e} is the set of exogenous events. We assume that E_{a} always contains a null-action ϵ, representing idleness. A policy, π, for a discrete event system is a mapping from situations to actions.

Returning to the example problem, we can represent the possibility of a taxi moving without us in it by two exogenous events: move-taxi and return-taxi. Examples of actions are given in the plan in Figure 1 (eg. load-taxi and drive).

A discrete event system , $\mathcal{M}$, controlled by a policy, π, is a stochastic process, denoted $\mathcal{M}[\pi]$. The execution history of $\mathcal{M}[\pi]$ is captured by a *sample execution path*, which is a sequence

$$\sigma = s_0 \xrightarrow{t_0,e_0} s_1 \xrightarrow{t_1,e_1} s_2 \xrightarrow{t_2,e_2} \dots$$

with $s_i \in S$, $e_i \in E$, and $t_i > 0$ being the time spent in state s_i before event e_i triggered a transition to state s_{i+1}. We call t_i the holding time for s_i.

Consider the example problem again. Say that the action load-taxi(pgh-taxi, CMU) triggers in the initial state after 1 minute. The holding time for the initial state is 1 in this case. The triggering of the load-taxi action takes us to a state where the package is in the Pittsburgh taxi. Say that in this state, the event move-taxi(msp-taxi) triggers after 2.6 minutes. We are now in a state where the Minneapolis taxi is moving, and the holding time for the previous state is 2.6. This sequence of states and triggering events represents a possible sample execution path for the transportation domain.

Sample execution paths come into play when verifying and repairing plans. For now, we make no assumptions about the underlying dynamical model of a discrete event system other than that we can generate sample execution paths for the system through discrete event simulation. In particular, we do not assume that the system is Markovian. We present a general algorithm for planning with discrete event systems using only the information contained in sample execution paths. We will later adopt a GSMP model of discrete event systems, and this will help us better focus the search for a satisfactory plan.

Problem Specification

A planning problem for a given discrete event system is an initial state s_0 (or possibly a distribution $p_0(s)$ over states) and a goal condition ϕ. We propose CSL as a formalism for specifying goal conditions. CSL—inspired by CTL (Clarke, Emerson, & Sistla 1986) and its extensions to continuous-time systems (Alur, Courcoubetis, & Dill 1993)—adopts probabilistic path quantification from PCTL (Hansson & Jonsson 1994).

The set of well-formed CSL formulas can be defined inductively as follows:

- An atomic proposition, a, is a state formula.
- If ϕ is a state formula, then so is $\neg\phi$.
- If ϕ_i, $i \in [1, n]$, are state formulas, then so is $\phi_1 \wedge \ldots \wedge \phi_n$.
- If ρ is a path formula and $\theta \in [0, 1]$, then $\Pr_{\geq\theta}(\rho)$ is a state formula.
- If ϕ is a state formula, then $X\phi$ ("next state") is a path formula.
- If ϕ_1 and ϕ_2 are state formulas and $t \in [0, \infty)$, then $\phi_1 \, \mathcal{U}^{\leq t} \, \phi_2$ ("bounded until") is a path formula.

In full CSL there are also "until" formulas without time bound, but we require the time bound in our simulation-based approach to guarantee that we need to consider only finite-length sample execution paths. The time bound of

```
(:action load-airplane
    :parameters (?plane - airplane ?loc - airport)
    :precondition (and (at pkg ?loc) (at ?plane ?loc) (not (moving ?plane)) (not (full ?plane)))
    :delay 1
    :effect (and (when (have-reservation)
                    (and (not (at pkg ?loc)) (in pkg ?plane) (moving ?plane)))
                 (when (not (have-reservation))
                    (probabilistic 0.9 (and (not (at pkg ?loc)) (in pkg ?plane) (moving ?plane))
                                   0.1 (full ?plane)))))
```

Figure 2: PDDL-like specification of the load-airplane action in the example transportation domain.

an "until" formula serves as a planning horizon. In this paper we will concentrate on CSL formulas of the form $\phi = \Pr_{\geq\theta}(\rho)$ where ρ does not contain any probabilistic statements. While the verification algorithm presented by Younes & Simmons (2002) can handle nested probabilistic quantification and conjunctive probabilistic statements, it is not immediately obvious how to compare plans if we allow such constructs in goal conditions.

The truth value of a state formula is determined in a state, while the truth value of a path formula is determined over an execution path. The standard logic operators have the usual semantics. A probabilistic state formula, $\Pr_{\geq\theta}(\rho)$ holds in a state s iff the probability of the set of paths starting in s and satisfying ρ is at least θ. The formula $X\phi$ holds over a path σ iff ϕ holds in the second state along σ. The formula $\phi_1 \,\mathcal{U}^{\leq t}\, \phi_2$ holds over σ iff ϕ_2 becomes true in some state s along σ before more than t time units have passed and ϕ_1 is true in all states prior to s along σ.

We can express the goal condition of the example problem as the CSL formula

$$\phi = \Pr_{\geq 0.9}(\neg\mathrm{lost}(\mathrm{pkg})\,\mathcal{U}^{\leq 300}\,\mathrm{at}(\mathrm{pkg},\mathrm{Honeywell})).$$

The problem is then to find a plan such that ϕ holds in the initial state s_0. The solution is a policy mapping situations to actions. Next, we present techniques for generating and repairing stationary policies, $\pi : S \to E_{\mathrm{a}}$.

Planning Algorithm

Algorithm 1 shows a generic procedure, FIND-PLAN, for probabilistic planning based on the GTD paradigm. The procedure takes a discrete event system $\mathcal{M}$, an initial state s_0, a CSL goal condition ϕ, and an initial plan π_0. The initial plan can be generated by an efficient deterministic planner, ignoring any uncertainty, or it can be a null-plan mapping all states to the null-action ϵ. If the initial plan is given as a sequence of events (as in Figure 1), then we derive a stationary policy by simulating the execution of the event sequence and mapping s to action a whenever a is executed in s. In the latter case we have a pure transformational planner (cf. Simmons 1988). The planning algorithm is specified as an anytime algorithm that can be stopped at any time to return the currently best plan found.

The procedure VERIFY-PLAN returns true iff ϕ is satisfied in s_0 by the stochastic process $\mathcal{M}[\pi]$. BETTER-PLAN returns the better of two plans. In the next two sections, we describe how to efficiently implement these two procedures

Algorithm 1 Generic planning algorithm for probabilistic planning based on the GTD paradigm.

```
FIND-PLAN(M, s_0, φ, π_0)
  if VERIFY-PLAN(M, s_0, φ, π_0) then
    return π_0
  else
    π ⇐ π_0
    loop    ▷ return π on break
      repeat
        π′ ⇐ REPAIR-PLAN(π)
        if VERIFY-PLAN(M, s_0, φ, π′) then
          return π′
        else
          π′ ⇐ BETTER-PLAN(π, π′)
      until π′ ≠ π
      π ⇐ π′
```

using acceptance sampling. In the third section we show how the information gathered during plan verification can be used to guide plan repair.

Anytime Plan Verification

Younes & Simmons (2002) propose an algorithm for verifying probabilistic real-time properties using acceptance sampling. Their work shows how to verify CSL properties given error bounds α and β, where α is the maximum probability of incorrectly verifying a true property (false negative) and β is the maximum probability of incorrectly verifying a false property (false positive). We adopt this approach, but develop a true anytime algorithm for verification of CSL properties of the form $\phi = \Pr_{\geq\theta}(\rho)$ that can be stopped at any time to return a decision with a confidence level whether ϕ holds or not. The more time the algorithm is given, the higher the confidence in the decision will be.

Assume we are using the sequential probability ratio test (Wald 1945) to verify a probabilistic property $\phi = \Pr_{\geq\theta}(\rho)$ with an indifference region of width 2δ centered around θ. Typically we fix the error bounds α and β and, given n samples of which d are positive samples, compute the fraction

$$f = \frac{p_1^d(1-p_1)^{n-d}}{p_0^d(1-p_0)^{n-d}}$$

with $p_0 = \theta + \delta$ and $p_1 = \theta - \delta$. We accept ϕ as true if

$$f \leq \frac{\beta}{1-\alpha},$$

reject ϕ as false if

$$f \geq \frac{1-\beta}{\alpha},$$

and generate an additional sample otherwise.

For an anytime approach to verification, we instead want to derive error bounds α and β that can be guaranteed if the sequential probability ratio test were to be terminated after n samples and d positive samples have been seen.

Given n samples and d positive samples, we would accept ϕ as true if we had chosen $\alpha = \alpha_0$ and $\beta = \beta_0$ such that

$$f \leq \frac{\beta_0}{1-\alpha_0} \tag{1}$$

holds. If, instead, we had chosen $\alpha = \alpha_1$ and $\beta = \beta_1$ satisfying the inequality

$$f \geq \frac{1-\beta_1}{\alpha_1}, \tag{2}$$

then we would reject ϕ as false at the current stage. What decision should be returned if the test procedure was terminated at this point, and what is the confidence in this decision in terms of error probability?

We will from here on assume that β_i can be expressed as a fraction of α_i; $\beta_i = \gamma\alpha_i$. We can think of γ as being the fraction $\frac{\beta}{\alpha}$, with α and β being the target error bounds for the verification of ϕ if enough time is given the algorithm. We can accept ϕ as true with probability of error at most β_0 if (1) is satisfied. We have

$$f \leq \frac{\gamma\alpha_0}{1-\alpha_0} \implies 1-\alpha_0 \leq \frac{\gamma\alpha_0}{f} \implies \alpha_0 \geq \frac{1}{1+\frac{\gamma}{f}}.$$

From this follows that if we had aimed for error bounds $\alpha = \frac{1}{1+\gamma/f}$ and $\beta = \frac{\gamma}{1+\gamma/f}$, then we would have accepted ϕ as true at this stage.

If (2) is satisfied, then we can reject ϕ as false with probability of error at most α_1. In this case we have

$$f \geq \frac{1-\gamma\alpha_1}{\alpha_1} \implies f\alpha_1 \geq 1-\gamma\alpha_1 \implies \alpha_1 \geq \frac{1}{\gamma+f}.$$

It now follows that if we had aimed for error bounds $\alpha = \frac{1}{\gamma+f}$ and $\beta = \frac{\gamma}{\gamma+f}$, then we would have rejected ϕ as false at this stage.

For the moment ignoring any decisions derived prior to the current sample, if we need to choose between accepting ϕ as true and rejecting ϕ as false at this stage, then we should choose the decision that can be made with the lowest error bounds (highest confidence). We choose to accept ϕ as true if

$$\frac{1}{1+\frac{\gamma}{f}} < \frac{1}{\gamma+f}, \tag{3}$$

we choose to reject ϕ as false if

$$\frac{1}{1+\frac{\gamma}{f}} > \frac{1}{\gamma+f}, \tag{4}$$

and we choose a decision with equal probability otherwise. Note that because we have assumed that $\beta_i = \gamma\alpha_i$, if $\alpha_i < \alpha_j$ then also $\beta_i < \beta_j$, so it is sufficient to compare the α_i's. If condition (3) holds, then the probability of error for the chosen decision is at most $\frac{\gamma}{1+\gamma/f}$, while if condition (4) holds the probability of error is at most $\frac{1}{\gamma+f}$. We denote the minimum of the α_i's at the current stage, $\min(\alpha_0, \alpha_1)$, by $\breve{\alpha}$.

Now, let us take into account decisions that could have been made prior to seeing the current sample. Let $\alpha^{(n)}$ denote the lowest error bound achieved up to and including the nth sample. If $\breve{\alpha} < \alpha^{(n)}$, assuming the current stage is $n+1$, then the decision chosen at the current stage without considering prior decisions is better than any decisions made at earlier stages, and should therefore be the decision returned if the algorithm is terminated at this point. Otherwise, a prior decision is better than the current, so we retain that decision as our choice. We set the lowest error bounds for stage $n+1$ in agreement with the selected decision: $\alpha^{(n+1)} = \min(\breve{\alpha}, \alpha^{(n)})$.

For the sequential probability ratio test to be well defined, it is required that the error bounds α and β are less than $\frac{1}{2}$. It is possible that either $\breve{\alpha}$ or $\gamma\breve{\alpha}$ violates this constraint if $\gamma \neq 1$. If that is the case, we simply ignore the current decision and make a random decision. This finalizes the algorithm, summarized in pseudo-code as Algorithm 2, for anytime verification of probabilistic properties. The result of the algorithm, if terminated after n samples, is the decision $d^{(n)}$ with a probability of error at most $\beta = \gamma\alpha^{(n)}$ if $d^{(n)} = \mathrm{true}$ and $\alpha = \alpha^{(n)}$ otherwise.

Figure 3(a) plots the error probability over time as the plan in Figure 1 is verified for the example problem using $\delta = 0.01$ and $\gamma = 1$. The decision for all confidence levels, except a few in the very beginning, is that the plan does not satisfy the goal condition. The confidence in the decision increases rapidly initially, and exceeds 0.99 within 0.08 seconds after 199 samples have been generated.

Plan Comparison

We need to compare two plans in order to perform hill-climbing search for a satisfactory plan. In this section we show how to use a sequential test, developed by Wald (1945), to determine which of two plans is better.

Let $\mathcal{M}$ be a discrete event system, and let π and π' be two plans. Given a planning problem with an initial state s_0 and a CSL goal condition $\phi = \Pr_{\geq\theta}(\rho)$, let p be the probability that ρ is satisfied by sample execution paths of $\mathcal{M}[\pi]$ starting in s_0 and let p' be the probability that ρ is satisfied by paths of $\mathcal{M}[\pi']$. We then say that π is better than π' for the given planning problem iff $p > p'$.

The Wald test is carried out by pairing samples for the two processes $\mathcal{M}[\pi]$ and $\mathcal{M}[\pi']$. We now consider samples of the form $\langle b, b' \rangle$, where b is the result of verifying ρ over a sample execution path for $\mathcal{M}[\pi]$ (similarly for b' and π'). We count samples only when $b \neq b'$. A sample $\langle \mathrm{true}, \mathrm{false} \rangle$ is counted as a positive sample because, in this case, π is performing better than π', while a sample $\langle \mathrm{false}, \mathrm{true} \rangle$ counts as a negative sample. Let $\tilde{p}$ be the probability of observing a positive sample. It is easy to verify that if π and π' are equally good, then $\tilde{p} = \frac{1}{2}$. We should prefer π if $\tilde{p} > \frac{1}{2}$. The situation is similar to when we are verifying a probabilistic

Algorithm 2 Procedure for anytime verification of $\phi = \Pr_{\geq\theta}(\rho)$ with an indifference region of width 2δ.

```
VERIFY-PLAN(M, s, φ, π)
  α^(0) ⇐ 1/2, d^(0) ⇐ either, n ⇐ 0
  f ⇐ 1, p0 ⇐ θ + δ, p1 ⇐ θ − δ
  loop    ▷ return d^(n) on break
    generate sample execution path σ starting in s
    if M[π], σ ⊨ ρ then
      f ⇐ f · p1/p0
    else
      f ⇐ f · (1−p1)/(1−p0)
    α0 ⇐ 1/(1+γ/f), α1 ⇐ 1/(γ+f)
    if α0 < α1 then
      d ⇐ true
    else if α1 < α0 then
      d ⇐ false
    else
      d ⇐ either
    ᾰ = min(α0, α1)
    if max(ᾰ, γᾰ) < 1/2 then
      if ᾰ < α^(n) then
        α^(n+1) ⇐ ᾰ, d^(n+1) ⇐ d
      else if ᾰ = α^(n) and d ≠ d^(n) then
        α^(n+1) ⇐ α^(n), d^(n+1) ⇐ either
      else
        α^(n+1) ⇐ α^(n), d^(n+1) ⇐ d^(n)
    else
      α^(n+1) ⇐ α^(n), d^(n+1) ⇐ d^(n)
    n ⇐ n + 1
```

property $\Pr_{\geq\theta}(\rho)$, so we can use the sequential probability ratio test with probability threshold $\tilde{\theta} = \frac{1}{2}$ to compare π and π'.

Algorithm 3 shows the procedure for doing the plan comparison. Note that this algorithm assumes that we reuse samples generated in verifying each plan (Algorithm 2).

Domain Independent Plan Repair

During plan verification, we generate a set of sample execution paths $\boldsymbol{\sigma} = \{\sigma_1, \ldots, \sigma_n\}$. Given a goal condition $\phi = \Pr_{\geq\theta}(\rho)$, we verify the path formula ρ over each sample path σ_i. We denote the set of sample paths over which ρ holds $\boldsymbol{\sigma}^+$, and the set of paths over which ρ does not hold $\boldsymbol{\sigma}^-$. The sample paths in $\boldsymbol{\sigma}^-$ provide information on how a plan can fail. We can use this information to guide plan repair without relying on specific domain knowledge.

In order to repair a plan for goal condition $\phi = \Pr_{\geq\theta}(\rho)$, we need to lower the probability of paths not satisfying ρ. A negative sample execution path

$$\sigma_i^- = s_{i0} \xrightarrow{t_{i0},e_{i0}} s_{i1} \xrightarrow{t_{i1},e_{i1}} \ldots \xrightarrow{t_{i,m-1},e_{i,m-1}} s_{im}$$

is evidence showing how a plan can fail to achieve the goal condition. We could, conceivably, improve a plan by modifying it so that it breaks the sequence of states and events along a negative sample path.

Algorithm 3 Procedure returning the better of two plans.

```
BETTER-PLAN(π, π′)
  n ⇐ min(|b|, |b′|)
  f ⇐ 1, p0 ⇐ 1/2 + δ, p1 ⇐ 1/2 − δ
  for all i ∈ [1, n] do
    if b_i ∧ ¬b′_i then
      f ⇐ f · p1/p0
    else if ¬b_i ∧ b′_i then
      f ⇐ f · (1−p1)/(1−p0)
  α0 ⇐ 1/(1+1/f), α1 ⇐ 1/(1+f)
  if α0 ≤ α1 then
    return π    ▷ confidence 1 − α0
  else
    return π′   ▷ confidence 1 − α1
```

Algorithm 4 shows a generic procedure for plan repair. First, a state is non-deterministically selected from the set of states that occur along some negative sample path. Given a state, we select an alternative action for that state and return the modified plan. The sample paths help us focus on the relevant parts of the state space when considering a repair for a plan. The number of states and alternative actions to choose from can still be quite large. For satisfactory performance, it is therefore important to make an informed choice. In the next section we introduce a GSMP domain model, and we present preliminary work on using the added domain information to further focus the search for a satisfactory plan.

Algorithm 4 Generic non-deterministic procedure for repairing a plan.

```
REPAIR-PLAN(π)
  S⁻ ⇐ set of states occurring in σ⁻
  s ⇐ some state in S⁻
  a ⇐ some action in E_a \ π(s)
  π′ ⇐ π, but with the mapping of s to a
  return π′
```

Improved Search Control

So far we have presented a framework for generating policies for discrete event system without making any assumptions regarding the system model. We now show how we can use GSMPs to model concurrency, as well as uncertainty in action and event duration. We also present some techniques for making informed repair decisions.

GSMP Domain Model

The essential dynamical structure of a discrete event system is captured by a GSMP (Glynn 1989). A GSMP has the following components:

- A finite set S of states.
- A finite set E of events.
- For each state $s \in S$, a set $E(s) \subset E$ of events enabled in s.

- For each pair $\langle s, e\rangle \in S \times E(s)$, a probability distribution $p(s'; s, e)$ over S giving the probability of the next state being s' if event e triggers in state s.
- For each event e, a cumulative probability distribution function $F(t; e)$, s.t. $F(0; e) = 0$, giving the probability that e has triggered t time units after it was last enabled.

As before, we divide the set of events into two disjoint sets E_{a} and E_{e}. The null-action ϵ is enabled in every state.

Consider the exogenous events move-taxi and return-taxi of the example domain. We can associate an exponential trigger time distribution, $E(\frac{1}{40})$, with move-taxi meaning that a taxi is requested on average every 40 minutes. A uniform distribution, $U(10, 20)$, for return-taxi means that each request can take between 10 and 20 minutes. The move-taxi event is enabled in states where the taxi is idle at the airport, while the return-taxi event is enabled in states where the taxi is moving without us in it. These events have deterministic outcome, i.e. they always cause the same state transitions when triggered. The load-airplane action (Figure 2) is an example of a controllable event with probabilistic outcome.

The dynamics of a GSMP is best described in terms of discrete event simulation (Shedler 1993). We associate a real-valued clock $c(e)$ with each event $e \in E$ that indicates the time remaining until e is scheduled to occur. The system starts in some initial state s with events $E(s)$ enabled. For each enabled event $e \in E(s)$, we sample a duration according to the cumulative distribution function $F(t; e)$ and set $c(e)$ to the sampled value. Let e^* be the event in $E(s)$ with the shortest duration, and let $c^* = c(e^*)$. The event e^* becomes the triggering event in s. When e^* triggers, we sample a next state s' according to the probability distribution $p(s'; s, e^*)$. Update the clock for each event $e \in E(s')$ enabled in the next state as follows:

- If $e \in E(s) \setminus \{e^*\}$, then subtract c^* from $c(e)$.
- If $e \notin E(s) \setminus \{e^*\}$, then sample a new duration according to the cumulative distribution function $F(t; e)$ and set $c(e)$ to the sampled value.

The first condition highlights the fact that GSMPs are non-Markovian, as the durations for events are not independent of the history. The system evolves by repeating the process of finding the triggering event in the current state, and updating clock values according to the scheme specified above.

Consider the example problem. In the initial state, the exogenous event move-taxi(msp-taxi) is enabled. Say that the initial clock value for this event is 3.6 minutes. The action of choice in the initial state is load-taxi(pgh-taxi, CMU), which has a fixed delay of 1, so in this case the triggering event in the initial state is the load-taxi action. The event move-taxi(msp-taxi) is still enabled in the following state, so instead of sampling a new duration for the event, we just decrement its clock value by 1 (the time spent in the previous state). We now sample a duration for the action drive(pgh-taxi, CMU, pgh-airport). The delay distribution for this action is $U(20, 40)$, and say in this case the trip to the airport is scheduled to take 25 minutes. The move-taxi event now has the shortest duration and triggers after 2.6 minutes. This means that in the next state the Pittsburgh taxi is scheduled to arrive at the airport in 22.4 minutes. This illustrates how easily concurrent events and actions with varying duration are handled in a GSMP framework.

Heuristic Plan Repair

We can take advantage of information obtained through plan verification and the structure provided by a GSMP domain model to focus the repair of an unsatisfactory plan. To begin with, when choosing an alternative action for state s, we only need to consider the actions in $E(s)$, which typically is a much smaller set than E_{a}. Knowing the underlying dynamical model of the system we are considering also helps us choose repair actions in a more informed manner.

As before, consider the negative sample execution paths $\boldsymbol{\sigma}^-$. We can view a state-event-state triple $\langle s_{ij}, e_{ij}, s_{i,j+1}\rangle$ along a negative sample path σ_i^- as a *possible bug*. To repair the current plan we want to eliminate bugs, and we can do so for a bug $\langle s, e, s'\rangle$ by preempting e in s (i.e. planning an action in s likely to trigger before e), attempt to avoid s altogether, or plan the action assigned to s in a predecessor to s thereby giving the action more time to trigger. The last repair takes advantage of the GSMP structure of the domain. Each negative sample path can contain multiple possible bugs, and each bug can appear in more than one negative sample path or multiple times along the same path. The problem then becomes to select which bug to work on, and given a bug, select a repair.

To rank possible bugs, we assign a real value to each bug. The value of a bug $b = \langle s, e, s'\rangle$ for a specific sample path is given by

$$v(b; \sigma_i^-) = \min_j \begin{cases} -\gamma^{m-j-1} & b = \langle s_{ij}, e_{ij}, s_{i,j+1}\rangle \\ 0 & \text{otherwise} \end{cases} .$$

We can think of this in terms of utility, where we assign a utility of -1 to the last state along a negative sample path, and propagate the utility backwards along the path with discount factor γ. The value of a bug $\langle s, e, s'\rangle$ is the value of the state s'. If a bug occurs more than once along a path, the bug gets the minimum value over the path. To combine the value of a bug from multiple sample paths, we add the value of the bug over all negative sample paths:

$$v(b; \boldsymbol{\sigma}^-) = \sum_i v(b; \sigma_i^-)$$

This gives us a real value for each bug, and we select the bug with the lowest value.

Returning to our example problem, we have already said that the initial plan for our example problem can be verified not to satisfy the goal condition. The actual probability of success can be estimated independently by sampling, and is approximately 0.77. Obtaining an accurate probability estimate is costly, however, so we never actually compute any probability estimates during planning. Given the sample execution paths generated during plan verification, we can compute the value of the bugs encountered. For a particular verification run, the worst bug is the event lose-package(msp-airport) causing the package to get lost from the state where the package and the plane are at the Minneapolis airport, the Minneapolis taxi is moving, and the

Pittsburgh taxi is at the airport (value -13). We choose this bug and try to repair it. The bug is that the package gets lost while we are waiting for the taxi at the Minneapolis airport. We can preempt the lose-package(msp-airport) event by either loading the plane or storing the package at the airport. Once we preempt the event, we should plan a path from the state we anticipate to reach with the planned action to a state where the package is at Honeywell (goal state). If, for example, we choose to store the package, we should plan to retrieve the package once the taxi arrives, or else the package will remain in storage indefinitely. We can find such a path using a heuristic deterministic planner, much in the same way as we expect the initial plan to be generated. We choose the repair action with the shortest path to a goal state, which in this case is the store action (not the load-plane action).

We now have a new (hopefully improved) plan that we need to verify. Running the verification algorithm on this plan reveals that it still does not satisfy the goal condition. The error probability over time when verifying the second plan is shown in Figure 3(b). The actual success probability (0.89) is close to the threshold (0.9), so it takes longer time and more samples (963 in this case) to reach a high confidence than for the initial plan. This demonstrates an interesting feature of a sequential test, viz. that it adapts to the difficulty of the problem. When comparing the two plans, we find with high confidence that the second plan is better than the first plan. For the verification data generated during the runs depicted in Figures 3(a) and 3(b), the confidence is 0.94 when using $\delta = 0.05$.

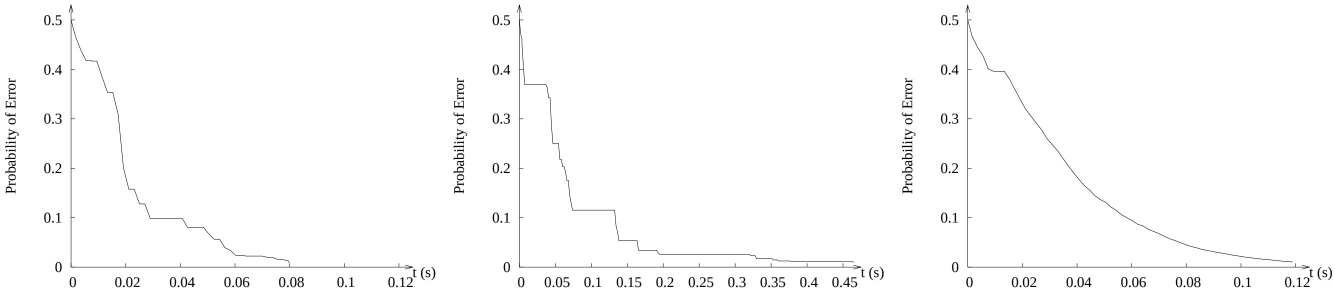
Because we determine with high confidence that the new plan is better than the old one, we select the new plan and try to repair it. The worst bug is now that the package is lost at the Pittsburgh airport after discovering that the plane is full (value -56). We can avoid losing the package by storing it at the airport, but there is no way to get the package to Honeywell once the plane is full so this repair action can be determined to be fruitless. Since there are no feasible repairs for the worst bug, we instead turn to the second worst bug, which in this case is that the plane is discovered to be full when we attempt to load the package at the Pittsburgh airport (value -47.75). The only way to disable the harmful effect of the load-airplane action is to make a reservation before leaving CMU. Making the suggested change to the current plan, we obtain a new plan that passes the verification phase, so we return that plan as a solution. We can independently verify through sampling that the success probability of the final plan is 0.99.

Discussion

We have presented a framework for policy generation in continuous-time stochastic domains. Our planning algorithm makes practically no assumptions regarding the complexity of the domain dynamics, and we can use it to generate stationary policies for any discrete event system that we can generate sample execution paths for. By adopting a GSMP domain model, we can naturally represent concurrent actions and events as well as uncertainty in the duration and outcome of these. While most previous approaches to probabilistic planning requires time to be discretized, we work with time as a continuous quantity, thereby avoiding the state-space explosion that comes from using discrete-time models for domains that are inherently continuous. To efficiently handle continuous time, we rely on sampling-based techniques. We use CSL as a formalism for specifying goal conditions, and we have presented an anytime algorithm based on sequential acceptance sampling for verifying whether a plan satisfies a given goal condition. Our approach to plan verification differs from previous simulation-based algorithms for probabilistic plan verification (Blythe 1994; Lesh, Martin, & Allen 1998) in that we avoid ever calculating any probability estimates. Instead we use efficient statistical hypothesis testing techniques specifically designed to determine whether the probability of some property holding is above a target threshold. We believe that the verification algorithm in itself is a significant contribution to the field of probabilistic planning.

Our approach to probabilistic planning falls into the Generate, Test and Debug paradigm, and we have presented some initial work on how to debug and repair stationary policies for continuous-time stochastic domains modeled as GSMPs. Our algorithm utilizes information from the verification phase to guide plan repair. In particular, we are using negative sample execution paths to determine what can go wrong, and then try to modify the current plan so that the negative behavior is avoided. Our planning framework is not tied to any particular plan repair technique, however, and our work on plan repair presented in this paper is only preliminary. As with any planning, the key to focused search is good search control heuristics. Information from simulation traces helps us focus the repair effort on relevant parts of the state space. Still, we have not fully addressed the problem of choosing among possible plan repairs. We are currently ignoring positive sample execution paths, but these could contain valuable information useful for guiding the planner towards satisfactory plans. Timing information in sample execution paths is also ignored for now. We could use the sampled holding times to better select which action to enabled for a state. We may think that an event will trigger after an action just by looking at the delay distributions, but the event may often trigger before the action in reality due to history dependence. The information contained in the sample execution paths could reveal this to us. We plan to address these issues in future work, as well as the problem of comparing and repairing plans when the goal is a conjunction of probabilistic statements.

We have limited our attention to stationary policies in this paper. Because we are considering finite-horizon goal conditions, there will be cases when a stationary policy is not sufficient and we need to consider non-stationary policies, $\pi_{\mathrm{ns}} : S \times \mathbb{R} \rightarrow E_{\mathrm{a}}$, in order to find a solution. The domain of a non-stationary policy for continuous-time domains is infinite (and even uncountable), so we would first need to find a compact representation for such policies in order to handle them efficiently. Furthermore, the holding times in the sample execution paths must be considered when debugging a plan. This means that seeing two identical bugs becomes highly unlikely, and any effective bug



(a) (b) (c)

Figure 3: Probability of error over time, using $\delta = 0.01$ and $\gamma = 1$, for the verification of (a) the initial plan, (b) the plan storing the package at the Minneapolis airport when the taxi is not there, and (c) the plan making a reservation before leaving CMU. The decision in (a) and (b)—except in the very beginning for some error bounds above 0.4—is that the goal condition is not satisfied, while in (c) the decision is that the goal condition is satisfied.

analysis technique would have to generalize the information in the sample paths. Thrun (2000) suggests using nearest neighbor learning for handling continuous-space Markov decision processes (with partial observability), and a similar approach may prove to be useful for handling non-stationary policies in continuous-time domains. It should be noted that the verification algorithm does not rely on the policy to be stationary. In fact, it could be used without modification to verify models with continuous state variables and partial observability. All we need is a way to generate sample execution paths. The challenge to extend our planning framework to non-stationary policies, continuous state variables, and partial observability lies in developing efficient repair techniques to handle the added complexity.

We are also looking into extending the planning framework to decision-theoretic planning. Work in this direction has already begun within the CIRCA framework (Ha & Musliner 2002). Plan assessment is more difficult in a decision-theoretic framework. The value of a sample execution path is no longer simply true or false, but a real value drawn from an unknown distribution. The current approach is to estimate the expected utility of a plan using sampling, but the number of samples tends to be high. We are currently considering the use of non-parametric statistical hypothesis testing techniques to make the decision-theoretic planner more efficient.

Acknowledgments. This material is based upon work supported by DARPA/ITO and the Space and Naval Warfare Systems Center, San Diego, under contract no. N66001–00–C–8039, DARPA and ARO under contract no. DAAD19–01–1–0485, and a grant from the Royal Swedish Academy of Engineering Sciences (IVA). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the sponsors. Approved for public release; distribution is unlimited.

References

Alur, R.; Courcoubetis, C.; and Dill, D. 1993. Model-checking in dense real-time. *Information and Computation* 104(1):2–34.

Aziz, A.; Sanwal, K.; Singhal, V.; and Brayton, R. 2000. Model-checking continuous-time Markov chains. *ACM Transactions on Computational Logic* 1(1):162–170.

Baier, C.; Katoen, J.-P.; and Hermanns, H. 1999. Approximate symbolic model checking of continuous-time Markov chains. In *Proc. 10th International Conference on Concurrency Theory*, volume 1664 of *LNCS*, 146–161. Springer.

Balemi, S.; Hoffmann, G. J.; Gyugyi, P.; Wong-Toi, H.; and Franklin, G. F. 1993. Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control* 38(7):1040–1059.

Baxter, J., and Bartlett, P. L. 2001. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research* 15:319–350.

Blythe, J. 1994. Planning with external events. In *Proc. Tenth Conference on Uncertainty in Artificial Intelligence*, 94–101. Morgan Kaufmann Publishers.

Bresina, J.; Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D. E.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proc. Eighteenth Conference on Uncertainty in Artificial Intelligence*, 77–84. Morgan Kaufmann Publishers.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* C-35(8):677–691.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proc. Fifteenth National Conference on Artificial Intelligence*, 875–881. AAAI Press.

Clarke, E. M.; Emerson, E. A.; and Sistla, A. P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8(2):244–263.

Dean, T., and Boddy, M. S. 1988. An analysis of time-dependent planning. In *Proc. Seventh National Conference on Artificial Intelligence*, 49–54. AAAI Press.

Draper, D.; Hanks, S.; and Weld, D. S. 1994. Probabilistic planning with information gathering and contingent execution. In *Proc. Second International Conference on Artificial Intelligence Planning Systems*, 31–36. AAAI Press.

Drummond, M., and Bresina, J. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proc. Eighth National Conference on Artificial Intelligence*, 138–144. AAAI Press.

Ginsberg, M. L. 1989. Universal planning: An (almost) universally bad idea. *AI Magazine* 10(4):40–44.

Glynn, P. W. 1989. A GSMP formalism for discrete event systems. *Proceedings of the IEEE* 77(1):14–23.

Goldman, R. P., and Boddy, M. S. 1994. Epsilon-safe planning. In *Proc. Tenth Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers.

Ha, V., and Musliner, D. J. 2002. Toward decision-theoretic CIRCA with application to real-time computer security control. In *Papers from the AAAI Workshop on Real-Time Decision Support and Diagnosis Systems*, 89–90. AAAI Press. Technical Report WS-02-15.

Hansson, H., and Jonsson, B. 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6(5):512–535.

Howard, R. A. 1971. *Dynamic Probabilistic Systems*, volume II. New York, NY: John Wiley & Sons.

Infante López, G. G.; Hermanns, H.; and Katoen, J.-P. 2001. Beyond memoryless distributions: Model checking semi-Markov chains. In *Proc. 1st Joint International PAPM-PROBMIV Workshop*, volume 2165 of *LNCS*, 57–70. Springer.

Jensen, R. M., and Veloso, M. M. 2000. OBDD-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research* 13:189–226.

Kabanza, F.; Barbeau, M.; and St-Denis, R. 1997. Planning control rules for reactive agents. *Artificial Intelligence* 95(1):67–113.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 2000. Approximate planning in large POMDPs via reusable trajectories. In *Advances in Neural Information Processing Systems 12: Proc. 1999 Conference*. Cambridge, MA: The MIT Press. 1001–1007.

Kushmerick, N.; Hanks, S.; and Weld, D. S. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1–2):239–286.

Lesh, N.; Martin, N.; and Allen, J. 1998. Improving big plans. In *Proc. Fifteenth National Conference on Artificial Intelligence*, 860–867. AAAI Press.

Matthes, K. 1962. Zur Theorie der Bedienungsprozesse. In *Trans. Third Prague Conference on Information Theory, Statistical Decision Functions, Random Processes*, 513–528. Publishing House of the Czechoslovak Academy of Sciences.

Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74(1):83–127.

Ng, A. Y., and Jordan, M. I. 2000. Pegasus: A policy search method for large MDPs and POMDPs. In *Proc. Sixteenth Conference on Uncertainty in Artificial Intelligence*, 406–415. Morgan Kaufmann Publishers.

Pistore, M., and Traverso, P. 2001. Planning as model checking for extended goals in non-deterministic domains. In *Proc. Seventeenth International Joint Conference on Artificial Intelligence*, 479–484. Morgan Kaufmann Publishers.

Rohanimanesh, K., and Mahadevan, S. 2001. Decision-theoretic planning with concurrent temporally extended actions. In *Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence*, 472–479. Morgan Kaufmann Publishers.

Schoppers, M. J. 1987. Universal plans for reactive robots in unpredictable environments. In *Proc. Tenth International Joint Conference on Artificial Intelligence*, 1039–1046. Morgan Kaufmann Publishers.

Shedler, G. S. 1993. *Regenerative Stochastic Simulation*. Boston, MA: Academic Press.

Simmons, R. G. 1988. A theory of debugging plans and interpretations. In *Proc. Seventh National Conference on Artificial Intelligence*, 94–99. AAAI Press.

Thrun, S. 2000. Monte Carlo POMDPs. In *Advances in Neural Information Processing Systems 12: Proc. 1999 Conference*. Cambridge, MA: The MIT Press. 1064–1070.

Wald, A. 1945. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics* 16(2):117–186.

Younes, H. L. S., and Simmons, R. G. 2002. Probabilistic verification of discrete event systems using acceptance sampling. In *Proc. 14th International Conference on Computer Aided Verification*, volume 2404 of *LNCS*, 223–235. Springer.