A Framework for Planning in Continuous-time Stochastic Domains (extended abstract)

Håkan L. S. Younes
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
lorens@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 also present an efficient procedure for comparing two plans that can be used in a hill-climbing search for a goal-satisfying plan.

Introduction

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.

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. The simulation traces generated during plan verification are used to guide plan repair. We recognize the need for good search control in order to make the planning algorithm practical. Initial work on heuristics for guiding plan repair is described in the full-length version of this paper presented at the main conference.

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 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

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.

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. 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 (e.g. 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.

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 syntax of CSL is defined as

$$\phi \;::=\; a \;\Big|\; \neg\phi \;\Big|\; \phi \wedge \phi \;\Big|\; \Pr_{\geq\theta}(\phi\, \mathcal{U}^{\leq t}\, \phi)$$

where $p \in [0,1]$, $t > 0$, and a is an atomic proposition. The standard logic operators have the usual semantics. A probabilistic formula, $\Pr_{\geq\theta}(\phi_1\, \mathcal{U}^{\leq t}\, \phi_2)$ holds in a state s iff the probability of the set of paths starting in s and satisfying $\phi_1\, \mathcal{U}^{\leq t}\, \phi_2$ is at least θ. The formula $\phi_1\, \mathcal{U}^{\leq t}\, \phi_2$ holds over σ iff ϕ_2 becomes true in some state s_i along σ before more than t time units have passed and ϕ_1 is true in all states prior to s_i along σ. 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.

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

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

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.

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

```
FIND-PLAN($\mathcal{M}, s_0, \phi, \pi_0$)
  if VERIFY-PLAN($\mathcal{M}, s_0, \phi, \pi_0$) then
    return $\pi_0$
  else
    $\pi \Leftarrow \pi_0$
    loop    ▷ return $\pi$ on break
      repeat
        $\pi' \Leftarrow$ REPAIR-PLAN($\pi$)
        if VERIFY-PLAN($\mathcal{M}, s_0, \phi, \pi'$) then
          return $\pi'$
        else
          $\pi' \Leftarrow$ BETTER-PLAN($\pi, \pi'$)
      until $\pi' \neq \pi$
      $\pi \Leftarrow \pi'$
```

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 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 $\breve{\alpha}$.

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

```
VERIFY-PLAN($\mathcal{M}, s, \phi, \pi$)
  $\alpha^{(0)} \Leftarrow \frac{1}{2},\ d^{(0)} \Leftarrow \text{either},\ n \Leftarrow 0$
  $f \Leftarrow 1,\ p_0 \Leftarrow \theta + \delta,\ p_1 \Leftarrow \theta - \delta$
  loop    ▷ return $d^{(n)}$ on break
    generate sample execution path $\sigma$ starting in $s$
    if $\mathcal{M}[\pi], \sigma \models \rho$ then
      $f \Leftarrow f \cdot \frac{p_1}{p_0}$
    else
      $f \Leftarrow f \cdot \frac{1-p_1}{1-p_0}$
    $\alpha_0 \Leftarrow \frac{1}{1+\gamma/f},\ \alpha_1 \Leftarrow \frac{1}{\gamma+f}$
    if $\alpha_0 < \alpha_1$ then
      $d \Leftarrow \text{true}$
    else if $\alpha_1 < \alpha_0$ then
      $d \Leftarrow \text{false}$
    else
      $d \Leftarrow \text{either}$
    $\breve{\alpha} = \min(\alpha_0, \alpha_1)$
    if $\max(\breve{\alpha}, \gamma\breve{\alpha}) < \frac{1}{2}$ then
      if $\breve{\alpha} < \alpha^{(n)}$ then
        $\alpha^{(n+1)} \Leftarrow \breve{\alpha},\ d^{(n+1)} \Leftarrow d$
      else if $\breve{\alpha} = \alpha^{(n)}$ and $d \neq d^{(n)}$ then
        $\alpha^{(n+1)} \Leftarrow \alpha^{(n)},\ d^{(n+1)} \Leftarrow \text{either}$
      else
        $\alpha^{(n+1)} \Leftarrow \alpha^{(n)},\ d^{(n+1)} \Leftarrow d^{(n)}$
    else
      $\alpha^{(n+1)} \Leftarrow \alpha^{(n)},\ d^{(n+1)} \Leftarrow d^{(n)}$
    $n \Leftarrow n + 1$
```

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 $d^{(n)} = \text{true}$ and $\alpha = \alpha^{(n)}$ otherwise.

Figure 2 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 $\langle \text{true}, \text{false}\rangle$ is counted as a positive sample because, in this case, π is performing better than π', while a sample $\langle \text{false}, \text{true}\rangle$ 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 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).

Algorithm 3 Procedure returning the better of two plans.

```
BETTER-PLAN($\pi, \pi'$)
  $n \Leftarrow \min(|\boldsymbol{b}|, |\boldsymbol{b}'|)$
  $f \Leftarrow 1,\ p_0 \Leftarrow \frac{1}{2} + \delta,\ p_1 \Leftarrow \frac{1}{2} - \delta$
  for all $i \in [1, n]$ do
    if $b_i \wedge \neg b'_i$ then
      $f \Leftarrow f \cdot \frac{p_1}{p_0}$
    else if $\neg b_i \wedge b'_i$ then
      $f \Leftarrow f \cdot \frac{1-p_1}{1-p_0}$
  $\alpha_0 \Leftarrow \frac{1}{1+1/f},\ \alpha_1 \Leftarrow \frac{1}{1+f}$
  if $\alpha_0 \leq \alpha_1$ then
    return $\pi$    ▷ confidence $1 - \alpha_0$
  else
    return $\pi'$    ▷ confidence $1 - \alpha_1$
```

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

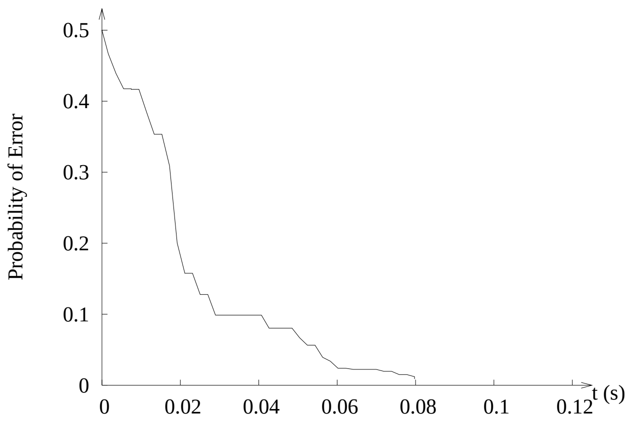

Figure 2: Probability of error over time, using $\delta = 0.01$ and $\gamma = 1$, for the verification of the initial plan. The decision in—except in the very beginning for some error bounds above 0.4—is that the goal condition is not satisfied.

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

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

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. A generic repair procedure could non-deterministically select a state occurring along some negative sample path, and for the selected state assign an alternative action. The sample paths help us focus on the relevant parts of the state space when considering a repair for a plan.

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. While most previous approaches to probabilistic planning requires time to be discretized, we work with time as a continuous quantity. 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 planning 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.

References

Alur, R.; Courcoubetis, C.; and Dill, D. 1993. Model-checking in dense real-time. *Information and Computation* 104.

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

Baier, C.; Katoen, J.-P.; and Hermanns, H. 1999. Approximate symbolic model checking of continuous-time Markov chains. In *Proc. CONCUR'99*.

Blythe, J. 1994. Planning with external events. In *Proc. UAI'94*.

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. UAI'02*.

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.

Dean, T., and Boddy, M. S. 1988. An analysis of time-dependent planning. In *Proc. AAAI'88*.

Drummond, M., and Bresina, J. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proc. AAAI'90*.

Hansson, H., and Jonsson, B. 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6.

Infante López, G. G.; Hermanns, H.; and Katoen, J.-P. 2001. Beyond memoryless distributions: Model checking semi-Markov chains. In *Proc. PAPM-PROBMIV'01*.

Lesh, N.; Martin, N.; and Allen, J. 1998. Improving big plans. In *Proc. AAAI'98*.

Simmons, R. G. 1988. A theory of debugging plans and interpretations. In *Proc. AAAI'88*.

Wald, A. 1945. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics* 16.

Younes, H. L. S., and Simmons, R. G. 2002. Probabilistic verification of discrete event systems using acceptance sampling. In *Proc. CAV'02*.