Probabilistic, Prediction-based Schedule Debugging for Autonomous Robot Office Couriers

Michael Beetz, Maren Bennewitz, and Henrik Grosskreutz

Dept. of Computer Science III,
University of Bonn, Germany
beetz,bennewit@cs.uni-bonn.de

Department of Computer Science V
Aachen Univ. of Technology, Germany
grosskreutz@cs.rwth-aachen.de

Abstract. Acting efficiently and meeting deadlines requires autonomous robots to schedule their activities. It also requires them to act flexibly: to exploit opportunities and avoid problems as they occur. Scheduling activities to meet these requirements is an important research problem in its own right. In addition, it provides us with a problem domain where modern symbolic AI planning techniques can enable robots to exhibit better performance than they possibly could without planning.
This paper describes PPSD, a novel planning technique that enables autonomous robots to impose order constraints on concurrent percept-driven plans to increase the plans' efficiency. The basic idea is to generate a schedule under simplified conditions and then to iteratively detect, diagnose, and eliminate behavior flaws caused by the schedule based on a small number of randomly sampled symbolic execution scenarios. The paper discusses the integration of PPSD into the controller of an autonomous robot office courier and gives an example of its use.

1 Introduction

Carrying out their jobs efficiently and meeting deadlines requires service robots to schedule their activities based on predictions of what will happen when the robot executes its intended course of action. Efficiency also requires robots to act flexibly: to exploit opportunities and avoid problems as they occur. Unfortunately, scheduling activities with foresight and acting flexibly at the same time is very hard. Even more so when jobs are underspecified or the robot is to carry out opportunistic plan steps which are triggered by enabling conditions. AI researchers have proposed several techniques to tackle these kinds of problems.

Constraint-based scheduling techniques have been successfully applied to very large scheduling problems. They gain efficiency by assuming that schedules can be checked without making detailed simulations of what will happen when a schedule gets executed. Sacrificing simulation-based plan checking will reduce the success rate of plans for robots that act in changing environments or carry out subplans that interact in subtle ways. Computationally, the situation gets even worse for robots that maintain a probabilistic belief state to estimate a high-dimensional world state.

Decision-theoretic partial-order planning systems [KHW95,WH94] aim at synthesizing plans with maximal expected utility by weighing the utility of each possible outcome with the probability that the plan will produce it. Given realistic computational resources, the probability distribution over possible outcomes can often not be estimated accurately enough to determine the plans with the highest expected utility [Yam94].

Decision-theoretic planning based on solving (partially observable) Markov decision problems ((PO)MDPs) [BDH98] model the robot's operation and effects on the world as a finite state automaton in which actions cause stochastic state transitions. The robot is rewarded for achieving its goal quickly and reliably. A solution for such problems is a *policy*, a mapping from states to actions that maximizes the accummulated reward. The use of (PO)MDPs to generate concurrent plans with continuous actions, such as navigation actions, requires us to discretize continuous actions into steps. In addition, if the plans are to contain event-enabled subplans (if you notice that room A is open while you pass the door then pick up the red letter) the size of plan steps is determined by the frequency of events (such as noticing that a door is open). Even worse, every conceivable event that might trigger a subplan has to be explicitly planned for. Whether we can encode control problems of such complexity as (PO)MDP problems that are so compact that they can be solved effectively is still an open question.

Our proposed solution to scheduling flexible activity is **PPSD *(Probabilistic Prediction-based Schedule Debugging)***, a method that generates candidate schedules fast ignoring the possibility of certain kinds of flaws and then iteratively debugs scheduled plans by revising them to eliminate flaws resulting from the simplifying assumptions (cf. [Sim92]). Based on inferences drawn from a small number of scenarios randomly projected wrt the robot's probabilistic belief state PPSD guesses whether and if so which flaws will probably occur if the robot executes its plan. This weaker kind of inference can be drawn fast and with reliability varying with the available resources [BM97].

Compared with the scheduling approaches discussed above we can categorize PPSD as follows. PPSD uses constraint-based scheduling techniques for generating inititial schedules under idealizing assumptions. The resulting candidate plans are then tested using symbolic prediction techniques that are extensions of those used by the decision-theoretic AI planning systems. The main advantage of PPSD over the (PO)MDP planning approaches is its parsimonity in the consumption of computational resources. PPSD uses a very compact representation of actions and states by reasoning about action models that describe continuous behavior, yet predicting only those state transitions that might affect the course of plan execution [BG98]. In addition, PPSD is less likely to waste computation time on reasoning about very unlikely plan outcomes. As a consequence, farsighted scheduling of the operation of autonomous service robots in realistic environments lie well within the applicability scope of PPSD, even of its prototypical implementation discussed in this paper. Of course, those advantages over the other approaches can only be obtained by contending ourselves with scheduling methods that produce with high probability "pretty good" schedules.[1][2]

Using PPSD we have implemented a special purpose planner that revises concurrent reactive plans for robots performing sets of complex, interacting, and user-specified tasks. This planner that is employed by the controller of an autonomous robot office

[1] The same restrictions apply to many constrained-based scheduling techniques as well as to the heuristic methods that are often used to approximate solutions to bigger (PO)MDPs.

[2] What "pretty good" means depends on the constrained-based scheduler used to generate the initial schedule, the utility function for robot's performance, and the realization of the schedule debugging step.

courier is a specific example where modern AI planning technology can contribute to autonomous robot control by improving the robot's behavior.

Several features of PPSD make it attractive as a resource for improving the performance of robots: (1) PPSD uses qualitatively correct models of the working principles of modern autonomous robot controllers. In particular, it models how monitoring processes and triggered reactions might interfere with the intended course of action. This enables the robot to forestall a variety of realistic behavior flaws that are not addressed by other planning approaches. (2) PPSD can be applied resource-adaptively. Revisions with a success rate sufficient for performance improvements can be performed within seconds and the success rate increases given more time. (3) PPSD detects a flaw using *Monte Carlo projection* with a likelihood depending only on the probability of the flaw itself but not on the size of the state space.

This paper makes three important contributions: It (1) describes PPSD, a novel scheduling technique; (2) shows how the technique can be instantiated for an autonomous robot office courier; and (3) describes an experiment where the technique improves the robot's performance. In the remainder of the paper we proceed as follows. After an illustrative example, we characterize the control problem abstractly, and describe the three contributions listed above in detail.

2 An Illustrative Example

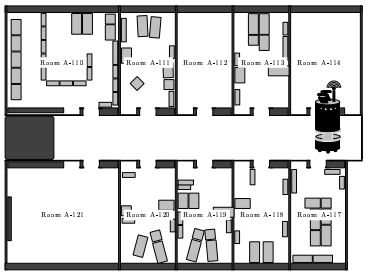

Fig. 1. Environment.

To illustrate the kinds of control problems that we address, consider the following situation in the environment pictured in Fig. 1. A robot office courier is to deliver a letter in a yellow envelope from room A-111 to A-117 (*cmd-1*) and another letter for which the envelope's color is not known from A-113 to A-120 (*cmd-2*). The robot has already tried to accomplish *cmd-2* but because it recognized room A-113 as closed (using its range sensors) it revised its intended course of action into achieving *cmd-2* opportunistically. That is, if it later detects that A-113 is open it will interrupt its current activity and reconsider its intended course of action under the premise that the steps for accomplishing *cmd-2* are executable.

To perform its tasks fast the robot should schedule the pick-up and deliver actions to minimize execution time and assure that letters are picked up before delivered. To ensure that the schedules will work, the robot has to take into account how the state of the robot and the world changes as the robot carries out its scheduled activities. Aspects of states (state variables) that the robot has to consider when scheduling its activities are locations of the letters. Constraints on the state variables that schedules have to satisfy are that they only ask the robot to pick up letters that are at that moment at the robot's location and that the robot does not carry two letters in envelopes with the same color.

Suppose our robot standing in front of room A-117 has received evidence that A-113 has been opened in the meantime. This requires the robot to reevaluate its options

for accomplishing its jobs with respect to its changed belief state. Executing its current plan without modifications might yield mixing up letters because the robot might carry two letters in envelopes with the same color. The different options are: (1) to skip the opportunity, (2) to immediately ask (via email) for the letter from A-113 to be put into an envelope that is not yellow (to exclude mixing ups when taking the opportunity later); (3) to constrain later parts of the schedule such that no two yellow letters will be carried even when the letter in A-113 turns out to be yellow; and (4) to keep picking up the letter in A-113 as an opportunistic subplan. Which option the robot should take depends on its belief state with respect to the states of doors and locations of letters. To find out which schedules will probably work, in particular, which one might yield mixing up letters, the robot must apply a model of the world dynamics to the state variables.

3 The Robot Office Courier Problem

Formulated abstractly, we investigate computational methods for robot action scheduling that enable autonomous service robots to perform their jobs efficiently and flexibly and reduce the number of behavior flaws. These methods have to work for robots with limited and noisy sensing capabilities. They also have to deal with multiple non-repetitive and possibly interfering jobs in a changing and partly unknown environment.

We make several assumptions to get a computationally feasible problem which are reasonable for a large fraction of the control problems we are interested in. The first assumption is that the controller maintains a probabilistic belief state about the state variables that may cause scheduling flaws. In our particular example the belief state comprises probability distributions about which envelopes are on which desks and the current state of doors in the form of probability distributions over the values of predefined random variables (e.g., P(nr-of-yellow-envelopes-on-desk-3 = 2) = 0.7). The second assumption about the controller is that the robot's problem solving behavior is specified as a plan with a partial order on plan steps (such as picking up and delivering letters) and a number of concurrent threats with higher priority called ***policies***. Policies specify constraints on the execution of the plan which might overwrite the order constraints within the plan. For example, the robot courier uses a policy to reschedule the plan whenever it detects that a door previously assumed to be closed is open and vice versa. In the following we will call such plans *concurrent reactive plans*. To specify such flexible and efficient behavior without policies, in particular those that reschedule plans dynamically, the robot would have to use a "universal" plan in which decisions for all possible situations the robot might run into are precomputed. To compute "good" schedules fast we assume that the ***performance measure*** is roughly proportional to the number of flaws in the schedule. Resource consumption (in our case, the travel time) has a smaller impact.

4 The Prediction-based Schedule Debugger

PPSD is to solve the following computational task: Given a probabilistic belief state, a set of jobs, and a concurrent reactive partially ordered plan. Find a totally ordered plan,

that accomplishes all jobs, and has a high performance (i.e., it is flawless and fast). To accomplish the computational task fast we only consider plans that result from the given plan through a sequence of revisions (addition of steps and ordering constraints).

For many tasks, such as courier service for hierarchically structured indoor environments, programmers can provide ***schedule generators*** that use heuristics and domain-specific methods for quickly computing schedules that minimize travel time. However, such scheduling methods cannot take into account the state changes of the robot and the world that occur as the activities get executed. They are therefore limited with respect to avoiding flaws caused by hidden interferences implied by the interactions of subplans.

The computational structure of prediction-based schedule debugging is a cycle that consists of the following steps: First, call the schedule generator to produce a candidate schedule O'. Second, randomly project the plan ⟨STEPS,O'⟩ constrained by a set of policies with respect to the belief state BS N times to produce the execution scenarios ES. Apply the flaw detection module to the execution scenarios ES and collect all flaws FLAWS that occur at least K times in the N scenarios. Next, RULE is set to a revision rule that is applicable to one of the flaws in FLAWS. The new plan ⟨STEPS,O⟩ results from the application of RULE to the current plan. The loop is iterated until no probable flaw in the schedule is left. The following pseudo code sketches the main computational steps of the PPSD algorithm.

```
algorithm PPSD(PLAN, BS, N)
1  ⟨STEPS,O⟩ ← PLAN
2  loop
3      O' ← GENERATE-SCHEDULE(⟨STEPS,O⟩)
4      ES ← RANDOMLY-PROJECT(⟨STEPS,O'⟩, BS, N)
5      FLAWS ← DETECT-SCHEDULE-FLAWS(ES,K)
6      RULE ← CHOOSE(RULES(CHOOSE(FLAWS)))
7      ⟨STEPS,O⟩ ← APPLY(RULE)
8    until FLAWS = {}
```

There is, in general, no guarantee that eliminating the cause for one flaw won't introduce new flaws or that debugging will eventually reduce the number of flaws [Sim92].

PPSD is implemented within the XFRM planning framework [McD92]. In some ways, XFRM is like a tool for building expert systems: it is an empty shell for building transformational planners for robot controllers written in RPL. XFRM provides powerful and general tools for the prediction-based revision of concurrent reactive plans:

- The **Reactive Plan Language (RPL)** [McD91] allows programmers to specify concurrent reactive plans. RPL comes with an execution system that runs on autonomous robots [Bee99]. It also represents controllers as syntactic objects that can be inspected and manipulated by plan revision methods [McD92].
- The **projection module** PTOPA (see [McD94]) that generates symbolic representations of possible execution scenarios. PTOPA takes as its input an RPL plan, rules for generating exogenous events, and a set of probabilistic rules describing the effects of exogenous events and concurrent reactive control processes. PTOPA randomly samples execution scenarios from the probability distribution that is implied by the rules. An execution scenario describes how the execution of a robot controller

might go, that is, how the environment changes as the plan gets executed. It is represented as a timeline, a linear sequence of dated events, which cause new world states. World states are characterized by a set of propositions.
– **XFRMML**, a declarative notation for formulating queries about projected execution scenarios, as well as plan transformation rules. Using XFRMML, PPSDs can reason about and apply plan transformation rules to eliminate schedule flaws.

To realize a PPSD debugger within the XFRM framework, a programmer has to do several things:

1. Specify a domain-specific fast heuristic schedule generator and implement it as an XFRMML plan revision rule.
2. Specify a causal model of the world and the robot controller using the PTOPA rule language, which is used to predict the state transitions caused by executing the robot controller. The causal model includes PTOPA rules that use the robot's belief state to sample the initial state of the timeline.
3. Provide XFRMML rules that represent what behavior flaws are and how plans can be revised to eliminate them. Flaws are defined through states (state sequences) that are declared to be forebidden.
4. Choose an appropriate parameterization of the flaw detection module (see below).
5. Provide a mechanism for updating the belief state, that is the probability distributions over the values of the random variables that are needed by the causal model.

The Flaw Detection Module A main factor that determines the performance of PPSD for a particular application is the ***flaw detector***. In general, different kinds of flaw detectors differ with respect to (1) the time resources they require; (2) the reliability with which they detect flaws that should be eliminated; and (3) the probability that they hallucinate flaws (that is, that they signal a flaw that is so unlikely that eliminating the flaw would decrease the expected utility).

PPSD should classify a flaw as to be eliminated if the probability of the flaw wrt the agent's belief state is greater than θ. PPSD should classify a flaw as hallucinated if the probability of the flaw wrt the agent's belief state is smaller than τ. We assume that flaws with probability between τ and θ have no large impact on the robot's performance.

To be more specific, consider a schedule flaw f that occurs in the distribution of execution scenarios of a given scheduled plan with respect to the agent's belief state with probability p. Further, let $X_i(f) = 1$ represent the event that behavior flaw f occurs in the ith execution scenario ($X_i(f) = 0$ else).

The random variable $Y(f,n) = \sum_{i=1}^{n} X_i(f)$ represents the number of occurrences of the flaw f in n execution scenarios. Define a probable schedule flaw detector DET such that DET(f,n,k) = *true iff* $Y(f,n) \geq k$, which means that the detector classifies a flaw f as to be eliminated if and only if f occurs in at least k of n randomly sampled scenarios.

Now we can build a model for the schedule flaw detector. Since the occurrence of schedule flaws in randomly sampled execution scenarios are independent from each other, the value of $Y(f)$ can be described by the binomial distribution $b(n,p)$. Using $b(n,p)$ we can compute the likelihood of overlooking a schedule flaw f with probability p in n scenarios: $P(Y(f) < j) = \sum_{k=0}^{j-1} \binom{n}{k} * p^k * (1-p)^{n-k}$.

In our prototypical implementation, we choose θ starting at 50% and τ smaller than 5% and typically use DET*(f,3,2)*, DET*(f,4,2)*, or DET*(f,5,2)*. Projecting an execution scenario takes about three to eight seconds depending on the complexity of the plan and the number of probabilistically occurring exogenous events. This enables the robot to perform a schedule debugging step in less than thirty seconds (on average). Fig. 2(left) shows the probability that the flaw detector DET*(f,n,2)* for *n* = 3,...,5 will detect a schedule flaw with probability θ. The probability of a flaw less likely than τ to be eliminated is smaller than 2.3% (for all $n \leq 5$).

	Prob. of Flaw θ				
	50%	60%	70%	80%	90%
DET*(f,3,2)*	50.0	64.8	78.4	89.6	97.2
DET*(f,4,2)*	68.8	81.2	91.6	97.3	99.6
DET*(f,5,2)*	81.2	91.3	96.9	99.3	99.9

	θ					
	1%	10%	20%	40%	60%	80%
$\tau = .1\%$	1331	100	44	17	8	3
$\tau = 1\%$	$\perp$	121	49	17	8	3
$\tau = 5\%$	$\perp$	392	78	22	9	3

Fig. 2. Reliability of the probable schedule flaw detector (left). Number of scenarios to get 95% accurateness (right).

As projection gets faster, another question arises: how many execution scenarios (n) must be projected to recognize flaws of probability $\geq \theta$ with probability at least β? Or put another way, which flaw detector $DET(n,k)$ should we use? Instead of determining the optimal detector $DET(n,k)$ for given parameters τ, θ and β, we will consider the detectors $DET(n(\theta+\tau)/2, n)$ that will signal a flaw if $Y > n(\theta+\tau)/2$. Although $n(\theta+\tau)/2$ is not the optimal k, it is often a reasonable guess. Note, if $1-\beta \geq \alpha$, which is often the case, the probability of overlooking a flaw with probability θ is greater than the probability of hallucinating a flaw of probability τ. The less probable a flaw becomes, the more probable it will be overlooked. Thus, in the remainder we will only consider the probability of overlooking a flaw with probability θ, using $DET(n(\theta+\tau)/2, n)$.

When n is large (particularly, $np(1-p) > 9$) the binomial distribution $b(n,p)$ can be approximated a by a normal distribution $N(\sigma^2 = np(1-p), \mu = np)$. In addition, the normal distribution can be restated as a standard normal distribution: $W(\xi \leq x) = \Phi((x-\mu)/\sigma)$; Φ beeing the standard normal distribution function.

So, how probable is it that we overlook a flaw? A flaw with probability θ is overlooked if $Y \leq n(\theta+\tau)/2$ with Y distributed according to $N(n\theta, n\theta(1-\theta))$. The probability, written as a standard normal distribution, is thus $\Phi(\frac{n(\theta+\tau)/2-\mu}{\sigma})$. To use β as an upper bound, $\frac{n(\theta+\tau)/2-\mu}{\sigma}$ must be $\leq \lambda_\beta$, with λ_β such that $\Phi(\lambda_\beta) \leq \beta$. Replacing σ with $\sqrt{n\theta(1-\theta)}$ and μ with $n\theta$, we need to determine the n that implies $\frac{n((\theta+\tau)/2-\theta)}{\sqrt{n\theta(1-\theta)}} \geq \lambda_{beta}$. Solving this equation for n yields $n \geq 4\lambda_{beta}^2 \theta(1-\theta)/(\tau-\theta)^2$. Fig. 2(right) shows the number of necessary projections to achieve $\beta = 95\%$ ($\lambda_\beta = 1.65$) (the difference compared to Fig. 2 result from the approximation).

5 Online Scheduling of Office Delivery Jobs

This section applies PPSD to the control of an autonomous robot office courier, called RHINO, operating in the environment shown in Fig. 1. The robot courier uses a library

of routine plans that specify among other things that objects are delivered by specifying that RHINO is to navigate to the pickup place, wait until the letter is loaded, navigate to the destination of the delivery, and wait for the letter to be unloaded. The plans are specified as concurrent reactive plans written in the plan language RPL [McD91]. The overall controller is implemented as a *Structured Reactive Controller* [Bee99] a particular controller that integrates and synchronizes planning processes with plan execution and allows for revising plans during their execution.

RHINO's Schedule Generator The algorithm for generating schedules is simple. Essentially, it sorts the navigation tasks (going to a target location) (counter)clockwise to get a candidate schedule. After this initial sort the scheduler iteratively eliminates and collects all steps s such that s has to occur after s' with respect to the required ordering constraints but occurs before s' in the candidate schedule. Then the steps s are iteratively inserted into the candidate schedule such that the ordering constraints are satisfied and the cost of insertion is minimal. While this greedy algorithm is very simple and fast it tends to produce fast schedules because the benign structure of the environment.

The Causal Model As the causal models of the concurrent reactive control routines, in particular the navigation behavior, we use a variation of the one proposed by Beetz and Grosskreutz [BG98] which makes probabilistically approximately accurate predictions. The predictions are represented compactly and generated within a few seconds. The probability distribution over which scenarios are generated is implied by the robot's belief state.

The **Flaw Detector** is realized through plan steps that generate "fail" events. Thus to detect schedule flaws an XFRMML query has to scan projected scenarios for the occurrence of failure events. For example, to detect deadline violations we run a monitoring process that sleeps until the deadline passes concurrently with the scheduled activity. When it wakes up it checks whether the corresponding command has been completed. If not a deadline violation failure event is generated.

Revision Rules. For schedule revision we use revision rules that are very similar to the ones originally developed by Beetz and Bennewitz [BB98]. The main difference is that the applicability of their rules is checked on real situation that occur during the execution of scheduled activity. The ones used in this paper include in addition rules that are triggered by predicted situations and applied to predicted execution scenarios. The advantage of these revision rules and their realization is that they can be run and revise the schedule ***while*** the scheduled plan is executed. Revision rules applied by the PPSD of the robot courier include one that adds ordering constraints to avoid holding two letters of the same color and one that inserts an email action asking the sender of a letter to use a particular envelope into the plan.

Belief State Manager. The belief state of the robot is updated through the interpretation of email messages, rules for updating the beliefs about dynamic states as time passes, and sensor data. For example, the rule "if office A-111 is closed it typically stays closed for about fifteen minutes" specifies that if the robot has not received any evidence about the door of room A-111 for fifteen minutes, the probability distribution for the door state is reset to the a priori probility. Email messages may contain information about probility distributions or facts that change conditional probabilities.

6 The Example Revisited

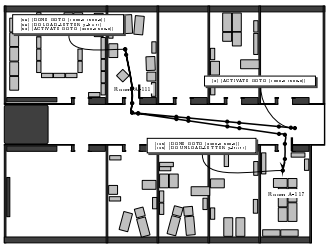

Fig. 3. A Possible projected execution scenarios for the initial plan.

Recall the example from the second section in which the robot standing at its initial position has perceived evidence that door A-113 has been opened. Therefore its belief state assigns probability p for the value true of random variable open-A113. The belief state also contains probabilities for the colors of letters on the desk in A-113.

Wrt. this belief state, different scenarios are possible. The first one, in which A-113 is closed, is pictured in Fig. 3. Points on the trajectories represent predicted events. The events without labels are actions in which the robot changes its heading (on an approximated trajectory) or events representing sensor updates generated by passive sensing processes. For example, a passive sensor update event is generated when the robot passes a door. In this scenario no intervention by prediction-based debugging is necessary and no flaw is projected.

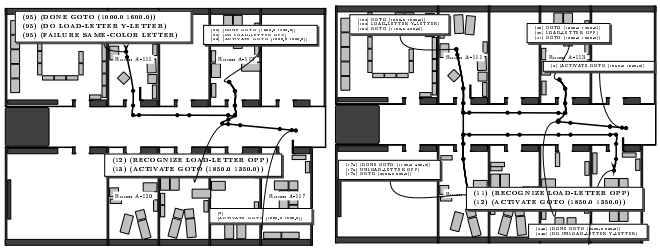

Fig. 4. The other possible execution scenarios for the initial plan.

In the scenarios in which office A-113 is open the controller is projected to recognize the opportunity and to reschedule its enabled plan steps as described above [3] . The resulting schedule asks the robot to first enter A-113, and pickup the letter for cmd-2, then enter A-111 and pick up the letter for cmd-1, then deliver the letter for cmd-2 in A-120, and the last one in A-117. This category of scenarios can be further divided into two categories. In the first subcategory shown in Fig. 4(left) the letter to be picked up is yellow. Performing the pickup thus would result in the robot carrying two yellow letters and therefore an execution failure is signalled. In the second subcategory shown in Fig. 4(right) the letter has another color and therefore the robot is projected to succeed by taking for all these scenarios the same course of action. Note, that the possible flaw is introduced by the reactive rescheduling because the rescheduler doesn't consider how the state of the robot will change in the course of action, in particular that a state may be caused in which the robot is to carry two letters with the same color.

[3] Another category of scenarios is characterized by A-113 becoming open after the robot has left A-111. This may also result in an execution failure if the letter loaded in A-113 is yellow, but is not discussed here any further.

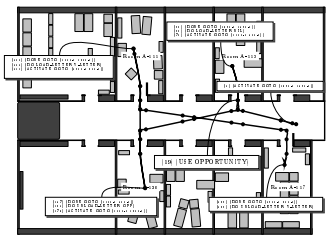

Fig. 5. Projected scenario for the revised plan.

In this case, PPSD will probably detect the flaw if it is probable with respect to the robot's belief state. This enables the debugger to forestall the flaw, for instance, by introducing an additional ordering constraint, or by sending an email that increases the probability that the letter will be put in a particular envelope. These are the revision rules introduced in the last section. Fig. **??** shows a projection of a revised plan.

Fig. 6(left) shows the event trace generated by the initial plan and ***executed*** with the RHINO control system [TBB+98] for the critical scenario without prediction based schedule debugging; Fig. 6(right) the one with the debugger adding the additional ordering constraint. This scenario shows that reasoning about the future execution of plans in PPSD is capable of improving the robot's behavior.

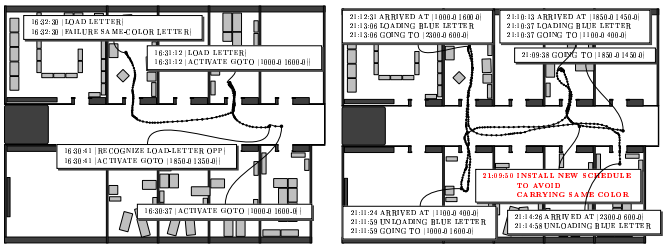

Fig. 6. Trajectory without PPSD (left). Trajectory when the flaw is forestalled by PPSD (right).

We have carried out three kinds of experiments to verify that PPSD-based scheduling can improve the behavior of autonomous robots. In the first one we have validated that PPSD-like plan revisions can be carried out reliably while the robot is operating. We have implemented a high-level controller for an interactive museums tourguide robot. The tourguide robot, called MINERVA, has operated for a period of thirteen days in the Smithsonian's National Museum of American History.[4] In this period, it has been in service for more than ninetyfour hours, completed 620 tours, showed 2668 exhibits, and travelled over a distance of more than fortyfour kilometers. MINERVA used plan revisions for the installment of new commands, the deletion of completed plans, and tour scheduling. The MINERVA experiment demonstrates that SRCs can (1) reliably control an autonomous robot over extended periods of time and (2) reliably revise plans during their execution.

The second experiment evaluated PPSD in a series of 12 randomly generated scenarios with about six delivery jobs most of them added asynchronously. Each scenario was tested in five runs (taking about twenty to twentyfive minutes for PPSD- and situation-based scheduling. As expected PPSD-based scheduling did on average not worse than situation-based scheduling methods.

In the last experiment we made up five scenarios in which the computation of good schedules required foresight. In those scenarios PPSD outperformed situation-based

[4] See `http://www.cs.cmu.edu/~minerva` for details.

scheduling by about eight percent (in this experiment, if the robot recognizes that it is about to load two letters of the same color, it puts the current job aside; thus confusing two letters never occurs. In the example pictured in Fig. 6(left) the robot would leave the room *without* loading the second letter. PPSD forestalls such detours.) We compared the expected durations for the paths taken in the scenario in order to eliminate the large variations in navigation time caused by different load averages for computers, interference by the reactive collision avoidance, etc.

7 Related Work

Probabilistic prediction-based schedule debugging is a planning approach rooted in the tradition of transformational planners, like HACKER [Sus77] and GTD [Sim92] that diagnoses "bugs" or, in our case, plan failures, in order to revise plans appropriately. In spirit, the control strategy of PPSD is very similar to the Generate/Test/Debug strategy proposed by Simmons [Sim92]. Our approach, like the XFRM system [McD92], differs from other transformational planners in that it tries to debug a simultaneously executed plan instead of constructing a correct plan. Also, we reason about full-fledged robot plans and are able to diagnose a larger variety of failures.

A number of approaches have been applied to activity scheduling in autonomous agent control. McDermott [McD92] has developed a prediction-based scheduler for location specific plan steps, which can probabilistically guess locations at which plan steps are executed if the locations are not specified explicitly. The main contribution of our approach is that McDermott's approach has been applied to a simulated agent in a grid-based world whereas ours controls a physical autonomous robot. Pell *et al.* [PBC+97] (re)schedule the activities of an autonomous spacecraft. While their system must generate working schedules we are interested in good schedules wrt. a given utility/cost function. In addition, scheduling activities of autonomous service robots typically requires faster and resource adaptive scheduling methods. Our scheduling approach differs from the one proposed by McVey *et al.* [MADS97] in that theirs generates real-time guaranteed control plans while our scheduler optimizes wrt. a user defined objective function.

8 Conclusions

In this paper we have developed a scheduling technique that enables autonomous robot controllers to schedule flexible plans, that is plans that allow autonomous robots to exploit unexpected opportunities[5] and to reschedule dynamically. The technique makes informed scheduling decisions by reasoning through concurrent sensor-driven plans that even reschedule themselves during execution. PPSD uses a fast heuristic schedule generator that might propose flawed schedules and then iteratively detects and eliminates schedule flaws based on a small number of randomly sampled execution scenarios.

Besides the technique itself, the paper gives a specific example in which modern AI planning technology can contribute to autonomous robot control by improving the

[5] By unexpected opportunities we mean states like an open door. The robot knows that doors can be open and closed but it does not know which door is open when. This is the unexpected part.

robot's behavior. The planning techniques, in particular the temporal projection and the plan revision techniques can do so because they (1) extend standard scheduling techniques and reactive plan execution techniques with means for predicting that the execution of a scheduled activity ***will result*** in a behavior flaw; (2) predict states relevant for making scheduling decisions that the robot ***won't be able*** to observe; (3) uses information about predicted states ***before*** the robot can observe them.

References

[BB98] M. Beetz and M. Bennewitz. Planning, scheduling, and plan execution for autonomous robot office couriers. In R. Bergmann and A. Kott, editors, *Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, volume Workshop Notes 98-02. AAAI Press, 1998.

[BDH98] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of AI research*, 1998.

[Bee99] M. Beetz. Structured reactive controllers — a computational model of everyday activity. In *Proceedings of the Third International Conference on Autonomous Agents*, 1999. to appear.

[BG98] M. Beetz and H. Grosskreutz. Causal models of mobile service robot behavior. In *Fourth International Conference on AI Planning Systems*, page 163, 1998.

[BM97] M. Beetz and D. McDermott. Fast probabilistic plan debugging. In *Recent Advances in AI Planning. Proceedings of the 1997 European Conference on Planning*, pages 77–90. Springer Publishers, 1997.

[KHW95] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995.

[MADS97] C. McVey, E. Atkins, E. Durfee, and K. Shin. Development of iterative real-time scheduler to planner feedback. In *"Proceedings of the 15th Int. Joint Conf. on Artificial Intelligence (IJCAI-87)"*, pages 1267–1272, 1997.

[McD91] D. McDermott. A reactive plan language. Research Report YALEU/DCS/RR-864, Yale University, 1991.

[McD92] D. McDermott. Transformational planning of reactive behavior. Research Report YALEU/DCS/RR-941, Yale University, 1992.

[McD94] D. McDermott. An algorithm for probabilistic, totally-ordered temporal projection. Research Report YALEU/DCS/RR-1014, Yale University, 1994.

[PBC+97] B. Pell, D. Bernard, S. Chien, E. Gat, N. Muscettola, P. Nayak, M. Wagner, and B. Williams. An autonomous spacecraft agent prototype. In *Proceedings of the First International Conference on Autonomous Agents*, 1997.

[Sim92] Reid Simmons. The role of associational and causal reasoning in problem solving. *AI Journal*, 53, 1992.

[Sus77] G. Sussman. *A Computer Model of Skill Acquisition*, volume 1 of *Aritficial Intelligence Series*. American Elsevier, New York, NY, 1977.

[TBB+98] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R.P. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA, 1998.

[WH94] M. Williamson and S. Hanks. Utility-directed planning. In *Proc. of AAAI-94*, page 1498, 1994.

[Yam94] E. Yampratoom. Using simulation-based projection to plan in an uncertain and temporally complex world. Technical Report 531, University of Rochester, CS Deptartment, 1994.