Computing Robust Plans in Continuous Domains

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

Abstract

We define the robustness of a sequential plan as the probability that it will execute successfully despite uncertainty in the execution environment. We consider a rich notion of uncertainty over continuous domains that includes stochastic action effects, and changes to state variables due to unpredictable exogenous events. Given a characterization of this uncertainty in terms of probability distributions (e.g., Gaussian) our contributions are two-fold: First, we describe a novel approach to computing the robustness of a plan in the situation calculus, which (a) separates the projection problem from the problem of reasoning about probability, and (b) explicitly reveals the relevance and statistical independence of random variables and events (i.e., conditions that contain random variables). Then, building on this approach, we describe a forward search based planner that generates maximally robust plans, exploiting the revealed structure for speed-up. Preliminary empirical results demonstrate that our approach can realize exponential savings in both time and space compared to the classical sampling approach.

1 Introduction

We define the robustness of a sequential plan as its probability of achieving the goal despite uncertainty about the values of certain state variables (fluents). Robust plans are required when the execution environment is not observable, unlike in (PO)MDP research. But unlike in classical conformant planning, we assume that the uncertainty is quantifiable. We consider a rich notion of uncertainty that is due to stochastic action effects or unpredictable exogenous events. We are particularly interested in, and here focus on, functional fluents that range over the real numbers and changes that follow known distributions, e.g., Gaussian or exponential.

Existing approaches to this problem (e.g., (Fox, Howey, and Long 2006; Blackmore et al. 2007)) conceptually follow state estimation techniques: they project the uncertainty forward to obtain sets of possible future states, sometimes called *belief states*, over which the goal and preconditions in the plan can be evaluated. In general, this can only be done using sampling, and the number of samples needed to assure accuracy of approximation increases quickly with the amount of uncertainty in the domain.

However, the problem of computing plan robustness is slightly different from the problem of state estimation because at any point in time only certain aspects of the state matter, namely those that are relevant to the achievement of the goal. This observation leads us to a more efficient technique for computing plan robustness which we propose in this paper. In particular, we develop a relevance-based technique that cleanly separates the projection problem, i.e., reasoning about action and change, from reasoning about probability. This is achieved by the use of *regression*. It turns out that applying regression in the context of robustness has a lot of potential. By regressing the goal and the preconditions of all actions in a plan, we are often able to not only determine that certain aspects of state are irrelevant, but also that certain relevant conditions are statistically independent. Exploiting this structure allows for exponential savings in computing robustness. Sometimes regression even produces closed form mathematical formulae for the robustness of a plan. At the same time, our approach can support rich notions of uncertainty, including complex interactions between sources of uncertainty.

After reviewing preliminaries, we describe our notion of uncertainty and robustness and show how all probabilistic reasoning can be regressed into the initial state. In Section 4 we describe a prototype planner for finding most robust plans and measure its empirical performance compared to a planner that uses the belief state approach, showing exponential savings in both time and space. We conclude with a brief discussion of related and future work.

2 Preliminaries

For the purpose of formal characterization, we here use the situation calculus, but the presented approach can be used with other action description languages as well. For illustration purposes, we will use the TPP domain from the International Planning Competition. In this domain, an agent needs to purchase goods at markets. Driving between markets takes time, and both money and time are limited resources. In our examples, we will consider various types of uncertainty about both prices and driving times.

The situation calculus is a logical language for specifying and reasoning about dynamical systems (Reiter 2001). In the situation calculus, the state of the world is expressed in terms of *fluents*, functions and relations relativized to a *situation* s, e.g., $F(\vec{x}, s)$. A situation is a history of the primitive actions

a performed from a distinguished initial situation S_0. The function $do(a, s)$ maps an action and a situation into a new situation thus inducing a tree of situations rooted in S_0. The relation $\sqsubset$ provides an ordering on situations of the same branch, and $s \sqsubseteq s'$ abbreviates $s = s' \vee s \sqsubset s'$. We abbreviate $do(a_n, do(a_{n-1}, \ldots do(a_1, s)))$ to $do([a_1, \ldots, a_n], s)$ or $do(\vec{a}, s)$.

A basic action theory in the situation calculus, $\mathcal{D}$, comprises four domain-independent foundational axioms, and a set of domain-dependent axioms. Details of the form of these axioms can be found in (Reiter 2001). Included in the domain-dependent axioms are the following sets:

Initial state axioms, $\mathcal{D}_{S_0}$: a set of first-order sentences relativized to situation S_0, specifying what is true in the initial state, e.g. $price(M_1, S_0) = 10$, where M_1 denotes a market, and $money(S_0) = 14$.

Successor state axioms: provide a parsimonious representation of frame and effect axioms under an assumption of the completeness of the axiomatization. There is one successor state axiom for each fluent, F, of the form $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$, where $\Phi_F(\vec{x}, a, s)$ is a formula with free variables among $\vec{x}, a, s$. $\Phi_F(\vec{x}, a, s)$ characterizes the truth value of the fluent $F(\vec{x})$ in the situation $do(a, s)$ in terms of what is true in situation s. These can be automatically generated from effect axioms such as: $time\big(do(DriveTo(d), s)\big) = time(s) - drivetime(at(s), d)$.

Action precondition axioms: specify the conditions under which an action is possible. There is one axiom for each action a of the form $\mathit{Poss}(a(\vec{x}), s) \equiv \Pi_a(\vec{x}, s)$ where $\Pi_a(\vec{x}, s)$ is a formula with free variables among $\vec{x}, s$. For instance, $\mathit{Poss}(Buy(m), s) \equiv price(m, s) < money(s)$.

A situation s is *executable* if all actions in s have their preconditions satisfied in the situation where they are performed: $\mathit{executable}(s) \stackrel{\text{def}}{=} (\forall a, s').do(a, s') \sqsubseteq s \supset \mathit{Poss}(a, s')$.

Definition 1. A *planning problem* is a tuple $\langle \mathcal{D}, G(s) \rangle$, where $\mathcal{D}$ is a basic action theory, and $G(s)$ a goal formula. A *plan* is a sequence of actions $\vec{a} = [a_1, \ldots, a_n]$ such that $\mathcal{D} \models \mathit{executable}(do(\vec{a}, S_0)) \wedge G(do(\vec{a}, S_0))$.

Regression The *regression* of a formula ψ through an action sequence $\vec{a}$ is a formula ψ' that holds prior to $\vec{a}$ being performed if and only if ψ holds after $\vec{a}$ is performed. In the situation calculus regression is defined inductively using the successor state axiom for a fluent $F(\vec{x}, s)$ as above:

$$\begin{array}{l|l} \mathcal{R}[F(\vec{x}, S_0)] = F(\vec{x}, S_0) & \mathcal{R}[\neg\psi] = \neg\mathcal{R}[\psi] \\ \mathcal{R}[F(\vec{x}, do(a, s))] = \mathcal{R}[\Phi_F(\vec{x}, a, s)] & \mathcal{R}[(\exists x)\psi] = (\exists x)\mathcal{R}[\psi] \\ \mathcal{R}[\psi_1 \wedge \psi_2] = \mathcal{R}[\psi_1] \wedge \mathcal{R}[\psi_2] & \end{array}$$

We use $\mathcal{R}[\psi, \vec{a}]$ as a shorthand for $\mathcal{R}\big[\psi(do(\vec{a}, S_0))\big]$. Intuitively, $\mathcal{R}[\psi, \vec{a}]$ is predominantly comprised of the fluents occurring in the conditional effects of the actions in $\vec{a}$. The only situation term it contains is S_0 and the regression theorem (Reiter 2001, p.65) states that for all formulae ψ: $\mathcal{D} \models \psi(do(\vec{a}, S_0))$ iff $\mathcal{D}_{S_0} \models \mathcal{R}[\psi, \vec{a}]$. Hence, regression can be used to reduce reasoning about future states of the world (projection) to reasoning about the initial state alone.

Regression is a purely syntactic operation. Nevertheless, it is often beneficial to simplify the resulting formula for later evaluation. Regression was first proposed by Waldinger (Waldinger 1977) and has been defined in many action specification languages, including STRIPS, ADL, and PDDL.

Plan validity as defined in Def. 1 requires a solution to the projection problem for reasoning about future states ($do(\vec{a}, S_0)$). However, for a given action sequence, using regression, it is easy to transform this definition into one that is solely with respect to the initial state (S_0):

Definition 2. The *validity condition* of an action sequence $\vec{a} = [a_1, \ldots, a_n]$ w.r.t. $\langle \mathcal{D}, G(s) \rangle$ is: $\mathit{Poss}(a_1, S_0) \wedge \mathcal{R}\big[\mathit{Poss}(a_2), [a_1]\big] \wedge \cdots \wedge \mathcal{R}\big[\mathit{Poss}(a_n), [a_1, \ldots, a_{n-1}]\big] \wedge \mathcal{R}\big[G, \vec{a}\big]$.

Corollary 1. Let $\vec{a}$ be an action sequence, $\Psi(S_0)$ its validity condition w.r.t. $\langle \mathcal{D}, G(s) \rangle$. Then $\vec{a}$ is a plan iff $\mathcal{D}_{S_0} \models \Psi(S_0)$.

The corollary is a consequence of the regression theorem and states that if the regression of the preconditions of all actions in the plan and the regression of the goal hold in the initial state, then the plan is going to succeed according to the model. In our approach, we will use this corollary to also reason about the *probability* of successful plan execution, i.e., plan robustness, in terms of the initial state alone. But first we need to define a notion of uncertainty.

3 Uncertainty and Plan Robustness

We assume that uncertainty can be quantified in terms of statistically independent random variables that follow known distributions. In our approach, we use regression to separate projection from probabilistic reasoning and we perform the latter "outside" of the situation calculus. As a result, we do not need to give semantics to random variables in the situation calculus, in contrast to others (e.g., (Mateus et al. 2001)). This is good news, as it keeps the theory nice and simple. Instead, we can simply represent a random variable as a non-fluent function that takes as many parameters as the density function describing the distribution of the variable, plus a constant that serves as an identifier. For instance, we use $f_{\mathcal{N}}(X, \mu, \sigma^2)$ as a placeholder for a random variable X with a normal (Gaussian) distribution with mean μ and variance σ^2. We allow these functions to appear in effect axioms and their arguments may be constants, fluents, or other random variables. For instance, $price(M_1, do(A, s)) = f_{\mathcal{N}}(X, price(M_1, s), duration(A, s) \cdot price(M_1, s))$ may express that after performing A, the price at market M_1 follows a normal distribution whose mean is the previous price, and whose variance is A's duration times that price. We disallow these functions from appearing as fluent arguments.

3.1 Regressing Random Variables

Since regression is a purely syntactic operation and in particular independent from S_0, the interpretation of these placeholder functions does not matter. After regressing a formula back to S_0 and replacing all fluents by their values defined by $\mathcal{D}_{S_0}$, we can give any interpretation to these functions, including probabilistic ones. For example, after regressing the preconditions for the Buy action (cf. Sec. 2), over the above action A and another action B that reduces our money by 3, say, we would obtain the formula $f_{\mathcal{N}}(X, price(M_1, S_0), duration(A, S_0) \cdot price(M_1, S_0)) < money(S_0) - 3$. After replacing all fluents by their values in S_0 and algebraic simplification, we

obtain $f_{\mathcal{N}}(X, 10, 2) < 11$, assuming A takes 0.2 time units. By the regression theorem, the purchase will be possible after performing $[B, A]$ iff this formula is true. Hence, the *probability* for the purchase to be possible can be defined as the probability for this formula to hold, when interpreting the placeholder function as a normally distributed variable $X \sim \mathcal{N}(10, 2)$, as intended.

In this particular example we can compute this probability very efficiently, by evaluating the corresponding cumulative distribution function (CDF). This is implemented by various software packages. In our implementation we use the freely available GNU Scientific Library[1] (GSL). When the regressed formula is not as simple as in the example—and in general it is not—one can sample from the distribution to approximate the probability of the formula to hold. However, there is a number of known simplification results about arithmetic operations over random variables. For instance, a mixture of Gaussians follows again a Gaussian distribution. This and other simplifications can result in the form above, for which probabilities can be computed very efficiently.

In what follows, for any formula ψ whose only situation term is S_0, we use $\psi_{\mathcal{D}_{S_0}}$ to denote the result of substituting all fluents in ψ by their value in S_0 as defined by $\mathcal{D}_{S_0}$. The resulting formula contains only numbers, Boolean operators and constants, and the described functions that denote random variables. We can hence interpret it according to probability theory and we denote its probability by $P(\psi_{\mathcal{D}_{S_0}})$. For instance, $P\big(f_{\mathcal{N}}(X, 10, 2) < 11\big) = 0.7602$.

3.2 Computing Robustness

Combining the above with Corollary 1, we immediately obtain the means to define and reason about the robustness of plans. Recall that the validity condition is a formula relativized to S_0 that captures what must be true in the initial state, in order for an action sequence to be a valid plan.

Definition 3. Let $\langle \mathcal{D}, G(s) \rangle$ be a planning problem, $\vec{a}$ an action sequence in $\mathcal{D}$, and $\Psi(S_0)$ its validity condition. Then the *robustness* of $\vec{a}$ is the probability $P(\Psi(S_0)_{\mathcal{D}_{S_0}})$.

Recall that we required all random variables referred to in the effect axioms to be independent from one another. This allows us to interpret the conjunction and disjunction of formulae that mention pairwise different random variables using multiplication and summation of the respective probabilities. However, this is not always the case. After regression, the random variables may interact in complex ways and random events (conditions that mention random variables) may *not* be statistically independent. Consider the two formulae $\varphi_1 = f_{\mathcal{N}}(X, 0, 1) > 2$ and $\varphi_2 = f_{\mathcal{N}}(Y, f_{\mathcal{N}}(X, 0, 1), 2) > 5$. Because the two random events mention the same random variable they are dependent. Thus we cannot compute the probability for $\varphi_1 \wedge \varphi_2$ by simply determining their individual probabilities and then multiplying the results. Instead, in these cases, we need to determine the probability for the formula as a whole, which can be done by sampling from all involved distributions simultaneously. Unfortunately, this requires a combinatorial increase in the number of samples, to obtain the same approximation quality.

[1] GSL: `http://www.gnu.org/software/gsl/`

Algorithm 1: Regr.-Based Uniform Robustness Search.

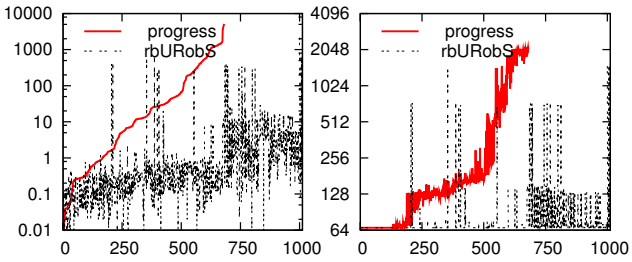
Input: action theory $\mathcal{D}$, goal $G(s)$, threshold δ

```
1  begin
2    open ← [ ] ;                                   // initialize open list
3    best.plan ← none ;                             // best plan so far
4    best.prob ← 0.0 ;                              // best plan's robustness
5    node ← (1.0, [ ], true) ;                      // the current search node
6    while best.prob < node.p do
7      foreach action a do
8        ā ← node.path · a ;                        // append action
9        ψ_poss ← R[Poss(a), node.path]_{D_S0} ∧ node.cond ;
10       p ← COMPUTEPROB(ψ_poss) ;
11       if p > δ then                              // prune low robustness plans
12         insert (p, ā, ψ_poss) into open according to p ;
13         Ψ_valid ← R[G, ā]_{D_S0} ∧ ψ_poss ;
14         p' ← COMPUTEPROB(Ψ_valid) ;
15         if p' > best.prob then                   // found a better plan
16           best.plan ← ā ;
17           best.prob ← p' ;
18     node ← POP(open), or break if open is empty
19   return best.plan, best.prob
20 end
```

4 Generating Robust Plans

Given a definition of plan robustness and the means for computing it, we turn to the problem of finding the most robust plan for a planning problem. The algorithm we propose is similar to uniform cost search, which is why we name it *regression-based uniform robustness search* (rbURobS). It is complete and approximately optimal, in the sense that it will generate a plan whose robustness is within ϵ from the most robust plan, where ϵ is the approximation error when computing probabilities. For analytically computed probabilities, ϵ is zero, and when sampling, $\lim_{n\to\infty} \epsilon = 0$ where n is the number of samples. The elements of the open list have the form $(p, path, cond)$, where $path$ is an action sequence, p its probability of being executable, and $cond$ the conjunction of regressed preconditions. Space precludes a detailed description of the algorithm, but intuitively it operates like uniform cost search and expands plan-prefixes in decreasing order of executability probability. Most crucial are lines 9 and 13, where preconditions and goal are regressed and all fluents evaluated in S_0. The search succeeds when a plan is found whose robustness is greater or equal to the executability probability of the head of the open list. Search fails when no plan with robustness $\geq \delta$ exists, where δ is a threshold parameter. Optimality follows from the fact that executability probabilities are non-increasing: by appending actions to a plan-prefix the probability cannot increase.

The auxiliary function COMPUTEPROB(ψ) computes $P(\psi)$, i.e., the probability that the condition ψ holds, given the probability distributions for all occurring random variables, if any. Given the structure of the regressed formulae, this function is able to exploit the statistical independence of random events. For instance, if $\psi = \psi_1 \wedge \psi_2$ and ψ_1 and ψ_2 do not have any random variables in common, as de-



(a) time in seconds (b) used memory in MB

Figure 1: Comparison of regression-based uniform robustness search vs. a belief-state based progression approach. For readability, we sorted the test cases by the time of the latter.

scribed above, COMPUTEPROB computes the probabilities for these two events separately and multiplies the results. In such a case, the sub-probabilities can be cached and then often reused later as well. In practice, this saves a lot of time, since formulae of consecutive branch nodes share many subconditions. When possible, the function computes probabilities using CDFs. In general though, simultaneous sampling for all random variables mentioned in ψ is necessary.

4.1 Empirical Results

We present preliminary empirical results, obtained using a proof-of-concept implementation of rbURobS, using Prolog for the regression and the GNU Scientific Library for probabilistic reasoning (in C). For comparison, we implemented the same uniform robustness search algorithm but using the classical belief state approach for reasoning about robustness (progress). Fig. 1 shows the time and memory (limit: 2GB) required by the two approaches for finding the most robust plans in problems from the described TPP domain. We considered five different settings of uncertainty about drive times and prices, described by effect axioms such as:

$$price\big(m, do(DriveTo(d), s)\big) = price(m, s) \cdot \big(1 + f_{\mathcal{N}}(C_1, 0, drivetime(at(s), d)/60)^2\big)$$
$$time\big(do(DriveTo(d), s)\big) = time(s) - drivetime(at(s), d) \cdot \big(1 + f_{\text{Exp}}(C_2, 4)\big)$$

where f_{Exp} stands for an exponentially distributed variable.

In the belief state approach, for each random variable appearing in the effects of an action, a set of samples is drawn to generate possible successor states. These successors form the next belief state. To maximize objectivity, the two implementations use the same code base, except for regression/progression. Both use the GSL for sampling from distributions, use the same number of samples per distribution appearing in a condition, and use the same threshold ($\delta = 0.1$) to prune low-probability branches.

The reason for the exponential savings of rbURobS, seen in the graphs (note the logarithmic scale), is that regression reveals which random events are relevant to the validity of a plan and also which ones are mutually independent. By exploiting this structure, we significantly reduce the amount of computation involved in determining the robustness of action sequences. Regression further creates a symbolic representation of plan robustness more compact than belief states.

5 Discussion

We have presented a new approach to computing the robustness of sequential plans in the face of uncertainty. Our approach distinguishes itself from previous approaches for reasoning about plan robustness (e.g., (Fox, Howey, and Long 2006; Blackmore et al. 2007)) by the richness of expressible uncertainty, and its ability to exploit the structure of the problem to compute probabilities more efficiently. By using regression, we are able to reveal the relevance structure of the problem and the statistical independence of random events. We furthermore separate the projection problem from the problem of reasoning about probabilities, which distinguishes us from previous approaches for defining uncertainty in the situation calculus (e.g., (Mateus et al. 2001; Bacchus, Halpern, and Levesque 1999)).

Going beyond the mere computation of robustness, we further proposed an algorithm for finding most robust plans. Preliminary empirical results confirm the intuitively possible exponential savings, due to the exploitation of the planning problem's relevance and dependence structure.

The problem of computing robust plans has received comparatively little attention in the AI community. Conceptually, the problem explores the spectrum between decision-theoretic and conformant planning. In fact, previous approaches to the latter have made similar use of the relevance structure of problems (e.g., (Son and Tu 2006)).

Lots needs to be done in terms of future work. For lack of space we opt to merely enumerate possible directions: run more experiments; integrate the technique with a state-of-the-art planner; develop heuristics for robustness search, e.g., by extracting maximally robust relaxed plans from plan graphs, or compute probability of reaching identified landmarks; incrementally build conditional plans by considering most likely points of failure.

Acknowledgments: We gratefully acknowledge funding from NSERC and ERA.

References

Bacchus, F.; Halpern, J. Y.; and Levesque, H. J. 1999. Reasoning about noisy sensors and effectors in the situation calculus. *Artificial Intelligence* 111(1-2):171–208.

Blackmore, L.; Bektassov, A.; Ono, M.; and Williams, B. C. 2007. Robust, optimal predictive control of jump markov linear systems using particles. In *Proc. of the 10th International Workshop on Hybrid Systems: Computation and Control*, 104–117.

Fox, M.; Howey, R.; and Long, D. 2006. Exploration of the robustness of plans. In *Proc. of the 21st National Conference on Artificial Intelligence*, 834–839.

Mateus, P.; Pacheco, A.; Pinto, J.; Sernadas, A.; and Sernadas, C. 2001. Probabilistic situation calculus. *Annals of Mathematics and Artificial Intelligence* 32(1-4):393–431.

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

Son, T. C., and Tu, P. H. 2006. On the completeness of approximation based reasoning and planning in action theories with incomplete information. In *Proc. of the 10th International Conference on Knowledge Representation and Reasoning*, 481–491.

Waldinger, R. 1977. Achieving several goals simultaneously. *Machine Intelligence* 8:94–136.