Representing General Numeric Uncertainty in Non-Deterministic Forwards Planning

Liana Marinescu and **Andrew Coles**
Department of Informatics,
King's College London, UK
firstname.lastname@kcl.ac.uk

Abstract

Many interesting applications of planning exhibit numeric uncertainty. Prior work in forwards planning approximates uncertain values as Gaussian distributions, but this is not always accurate. We explore a novel way to represent numeric uncertainty more generally. Our approach allows us to sample non-deterministic action effects from any probability distribution without sacrificing computational time. We integrate our approach into an existing policy-building setting, and use it to improve how well the states expanded by search reflect reality. This is part of a work in progress, and will provide new insights into the amount of detail about uncertainty necessary to obtain robust plans.

1 Introduction

1.1 Context

Planning under uncertainty is a compelling research area due to its role in broadening the range of problems that automated planners can tackle. Common situations where uncertainty arises include noisy sensors, unpredictable environments, and limited domain knowledge. For example, after each stretch of driving on rough terrain, a car may or may not have suffered a flat tyre - this is a case of propositional uncertainty. Or, after navigating through more or less favourable currents, a submarine may have used a non-deterministic amount of fuel - this is a case of numeric uncertainty. In this paper we focus on the latter, and in particular on the representation of non-deterministic numeric effects as probability distributions.

There is no question that plan robustness benefits from taking uncertainty into account. While it is possible to ignore uncertainty and assume all non-deterministic numeric effects take the median value every time, this simplification can have serious consequences for plan success. For example, a mission-critical action might cost 10 units of fuel; the planner might use the median value of all effects so far to calculate that the fuel remaining is exactly 10; it will then consider it safe to take the mission-critical action. In reality though, there might be 8.5 (or 9, or 10.7) fuel remaining, so there is a risk that the mission-critical action will fail. Planning under uncertainty aims to address this problem.

1.2 Prior Work

There is a rich body of prior work which tackles uncertainty from several angles.

Providing an excellent starting point for our contributions, work by (Beaudry, Kabanza, and Michaud 2010) uses a Bayesian network to model resources and time based on continuous random variables. They introduce the idea of querying the Bayesian network to check the likelihood of the variables remaining in a valid state. We base our planning kernel on their approach, as described further in this paper.

(Coles 2012) adapted the work of (Beaudry, Kabanza, and Michaud 2010) to assuming for heuristic purposes that variables take their median value. They proposed a method to first generate plans that are conservative about resource usage, and then to create branches that can exploit situations where resource usage is less than pessimistically expected. This approach to branching inspired part of our work as well.

Building on top of (Coles 2012) is the paper by (Marinescu and Coles 2016a), which employs the median in the calculation of the heuristic, and additionally introduces the concept of an offset – a safety margin by which preconditions must be met some given percentage of the time. They compute this margin based on Gaussian uncertainty in the problem model, and enable the planner to consider actions that reduce uncertainty.

For propositional uncertainty (where actions have many discrete outcomes), work by (Muise, McIlraith, and Beck 2012) and (Muise, Belle, and McIlraith 2014) on the planner PRP builds a policy by making repeated calls to a deterministic planning kernel. This kernel finds weak plans, which assume the non-deterministic action outcome can be chosen. They incrementally build a policy to cover the outcomes that were not chosen, and recurse. Their approach scales remarkably well due to the use of regression to keep only the relevant parts of a state, leading to a compact policy representation.

Building on the work of (Muise, McIlraith, and Beck 2012) on propositional uncertainty, (Marinescu and Coles 2016b) extend the policy-building process to numeric uncertainty. They achieve this by defining the process of regression through non-deterministic numeric effects. This offers them the additional benefit of generalising numeric deadends in order to prune them more efficiently. The limitation

here is that the approach only works if the effects are represented as Gaussian probability distributions. We aim to address this limitation with our current work.

There are many other compelling approaches to planning with numeric uncertainty. For example, work by (Beck and Wilson 2007) solves the job shop scheduling problem in the case where durations are drawn from Gaussian distributions. (Babaki, Guns, and Raedt 2017) integrate a probabilistic engine with constraint programming in order to acommodate uncertain demand or processing times in decision-making problems. Optimal planning in stochastic domains with resource constraints is addressed with a novel algorithm by (Meuleau et al. 2009). POMDPs are used to model the problem of maximising performance while bounding risk with a safety threshold by (Santana, Thiebaux, and Williams 2016).

1.3 Our work

We aim to tackle one of the main drawbacks of prior work on policy-bulding for non-deterministic numeric planning – the fact that it is limited in scope when it comes to the types of probability distributions it can accommodate. General distributions are acknowledged but they are not focused on, with approaches so far concentrating only on Gaussian distributions.

In (Marinescu and Coles 2016b) the outcome of a non-deterministic action can be chosen when starting to build the policy. For example, if an action has three modes (lucky, normal, and unlucky) the weak plan would always choose the lucky outcome and leave the other outcomes to be filled in later.

In our approach however, the user should not be telling the planner control knowledge, so we don't require the outcome modes to be specified in the domain file. In fact we expend no computational effort on discovering what the modes of a certain action might be – especially since the most suitable modes to branch on may differ at different points in the plan for the same action. We instead rely on branching as necessary in order to meet the certainty requirements, as we explain further below.

One novel element in the context of general probability distributions is enabling the planner to be proactive about improving on its initial solution. It can start by finding a policy which meets a given certainty threshold, and continue by incrementally refining the solution in order to nudge up the certainty.

An obvious question to pre-empt would be the following – won't we sacrifice computational time in order to accommodate any probability distributions in the policy-building process? We explore this concern later in the paper through the use of parallel computation in order to speed up our extended regression algorithm.

In the following, we first define the non-deterministic planning formalism we use, and the policy-building process on top of which we build our work. We then present our contribution and discuss our implementation and our extension to prior work.

As this is a work in progress, we can only provide our best estimates regarding experimental performance, and we put forth our ideas on both ways our approach could perform – faster or slower than the Gaussian-only approach.

2 Background

2.1 Planning with Numeric Uncertainty

The formalism we use in this work is based on that of (Beaudry, Kabanza, and Michaud 2010), adapted so that actions can have multiple outcomes, to support the policy-building mechanics we will detail in Section 2.2. A planning problem is thus a tuple $\langle F, \mathbf{v}, I, G, A, \theta \rangle$ where:

- F is a set of propositional facts.
- $\mathbf{v}$ is a vector of numeric variables.
- I is the initial state: a subset of F and values of variables in $\mathbf{v}$.
- Conditions are conjunctions of facts from F and Linear Normal Form constraints on $\mathbf{v}$, each written: $(\mathbf{w}.\mathbf{v} \geq c)$, where $c \in \Re$, and $\mathbf{w}$ is a vector of real values.
- G describes the goals: a set of conditions.
- A is a set of actions, with each $a \in A$ having:
  - $Pre(a)$: a (pre)condition on its execution;
  - $\mathit{Eff}(a)$: a list of outcomes. Each $o \in \mathit{Eff}(a)$ is a tuple $\langle \mathit{Eff}^+, \mathit{Eff}^-, \mathit{Eff}^{num} \rangle$ where:
    - Eff^+, Eff^-: a set of facts added (deleted) by that outcome;
    - Eff^{num}: a set of numeric variable updates triggered by that outcome, each of the form $\langle v \ op \ D(\mathbf{v}) \rangle$ where $op \in \{+=, =\}$ and D is a (possibly deterministic) probability distribution that governs the range of the numeric effect. For instance, $\langle battery \mathrel{+}= \mathcal{N}(-10, 2^2) \rangle$ means 'decrease $battery$ by an amount with mean 10 and standard deviation 2'.
- $\theta \in [0.5, 1)$ is a confidence level that $Pre(a)$ must meet to be considered true (this is necessary due to the uncertainty in effects).

Because there is uncertainty on numeric variables (due to the distributions D in Eff^{num}), it is not possible to be absolutely certain that numeric conditions are satisfied. Thus, (Beaudry, Kabanza, and Michaud 2010) uses a Bayesian Network (BN) to model this uncertainty, and check that numeric conditions are satisfied with the prescribed confidence level θ. When each action has only a single outcome, the task of planning is to find a sequence of steps $[a_0, .., a_n]$, giving a state trajectory $[I, S_0, .., S_n]$; with the BN ensuring that, with confidence θ, each action's preconditions are true and S_n satisfies the goals G.

Work by (Marinescu and Coles 2016a) looks at Gaussian vs non-Gaussian distributions in the context of heuristics. In particular, they intoduce a heuristic which is admissible for monotonically worsening uncertainty, based on the difference between the median and the θ'th percentile of a distribution. Using this difference (offset) they can evaluate whether numeric preconditions are true. In the case where an effect would have influence the uncertainty of a variable non-monotonically (e.g. assigns it a value rather than

Algorithm 1: Generating a Strong-Cyclic Plan in PRP (Muise *et al.* 2012)

Data: A planning task, with initial state I and goals G
Result: A policy P

```
1  P ← {}; Open ← [I]; Seen ← {};
2  while Open is not empty do
3      S ← Open.pop();
4      if (S ∈ Seen) ∨ (S ⊨ G) then continue;
5      Seen ← Seen ∪ S;
6      if ∃⟨ps, a⟩ ∈ P such that S ⊨ ps then
7          for S′ ∈ apply_outcomes(S, a) do
8              Open.push(S′);
9      else
10         (weak_plan, G′) ← run planning kernel from
           S;
11         if planning kernel could not solve problem then
12             ps_dead ← generalise_dead_end(S);
13             generate forbidden state–action pairs from
               ps_dead;
14             P ← {}; Open ← [I]; Seen ← {};
15         else
16             PS ← regress G′ through weak_plan to
               generate partial-state–action pairs;
17             P ← P ∪ PS;
18             for S′ ∈ apply_outcomes(S, weak_plan_0)
               do
19                 Open.push(S′);
20 return P
```

increases it by a value), then then in the heuristic the offset is reset back to zero.

2.2 Policy-Building for Uncertainty

As noted in the formalism above, actions can have multiple outcomes, and each outcome has a set of associated effects. A solution to problems containing such actions can be represented by using a *policy* – a set of rules that dictates what should be done in each state. For our policies, we assume states are fully observable, i.e. we know which action outcome occurred at any point.

In the presence of multiple outcomes, a *weak plan* corresponds to a single trajectory of actions that leads from the initial state to a goal state, assuming it is possible to choose which action outcome occurs at each point (i.e. to be optimistic). In the propositional case, weak plans can be found using a deterministic planner which is given as input the *all outcomes determinisation*. This means that each action with preconditions $Pre(a)$ and effects $\mathit{Eff}(a)$ is replaced by several actions, one for each $o \in \mathit{Eff}(a)$, whose preconditions are $Pre(a)$ and whose effects are just those corresponding to o.

(Muise, McIlraith, and Beck 2012) present an approach where, by repeatedly invoking a deterministic planner to find weak plans, it is possible to incrementally build a policy. The core of this approach is set out in Algorithm 1. Key to the success of their approach is exploiting *relevance* – by regressing the goal through a weak plan step-by-step, they determine which facts at each point are relevant to plan success.

Regression takes as input a partial state ps – here, a set of literals. It then applies an action 'backwards' to it, yielding a new partial state ps' that has to be satisfied prior to the action being applied. That is, applying the action in ps' returns us to ps.

The process begins from the goals, i.e. initially $ps = G$. Regressing ps through a step a of a weak plan, with preconditions $Pre(a)$ and a single outcome with add effects $\mathit{Eff}^+(a)$ yields a partial state ps' where:

$$ps' = (ps \setminus \mathit{Eff}^+(a)) \cup Pre(a)$$

Each of these pairs $\langle ps', a\rangle$ is added to the policy (line 17). If policy-building reaches a state S that matches some known partial-state–action pair (line 6), then all the outcomes of the corresponding action are applied. Otherwise, if there is no such match, the planning kernel is invoked from S. Ideally, this produces a weak plan, either to the goals G or to some other partial state which is already in the policy (G'), in which case the partial-state–action pairs from this weak plan are added to the policy.

Policy-building terminates when the open list is empty, hence $\exists\langle ps, a\rangle$ such that $S \vDash ps$ for all states S reachable from the initial state via the policy. Alternatively, if no strong cyclic plan exists, all actions that could be applied in the initial state are forbidden, and planning terminates with failure.

The numeric extension to this algorithm has been proposed by (Marinescu and Coles 2016b), where regression is performed through successive Gaussian effects by taking advantage of these distributions' analytical form.

3 Approach

3.1 Focus of the Paper

Our goal is to introduce a general representation of uncertain numeric effects in non-deterministic planning. We aim to allow effects to be sampled from any probability distribution, without incurring a high computational cost. Our representation allows the planner to find robust solutions which meet a given certainty threshold.

Below we present how the policy-building process described in Section 2.2 can be extended to incorporate a wider range of uncertain action effects. Specifically, our novel representation of uncertainty allows regression to be done through a sequence of actions with non-Gaussian effects.

The propositional elements of regression stay the same as in prior work by (Muise, McIlraith, and Beck 2012). We leverage, among other things, their approach to propositional uncertainty, for its notable speed when triaging possible matches to a given partial state.

The numeric elements of regression are the focus of our work. We show that it is computationally feasible to perform regression through non-Gaussian effects, and provide an efficient implementation to do so.

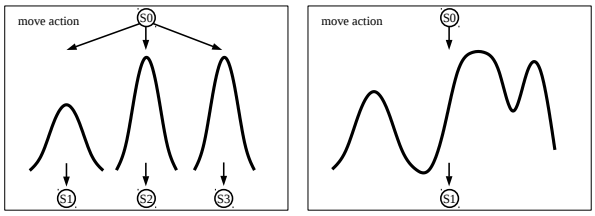

Figure 1: Multiple modes (left) vs single mode (right).

3.2 Multiple Modes vs Single Mode

One of the factors contributing to the success of prior work by (Marinescu and Coles 2016b) was the existence of explicit modes in non-deterministic action effects. For example, the `move` action had three different outcomes hard-coded in the PDDL domain file. One lucky mode (using less energy than nominally), one nominal mode, and one unlucky mode (using more energy than nominally). The existence of these modes allowed the planner to do two things. First, to perform an all-outcomes determinisation on the non-deterministic effects (cf. (Muise, McIlraith, and Beck 2012)), which resulted in a favourable weak plan (as it could choose all the lucky action outcomes). Second, to recursively branch off from the weak plan at those lucky points in order to solve for all the unfavourable outcomes as well.

However, there is a problem with these hard-coded modes – the user needs to specify them in the beginning. This is often impractical, as it forces the user to guess how an uncertain environment might react. It also implies giving the planner additional control knowledge, essentially offering it hints without being certain these hints are correct. We expect better performance when allowing the planner to decide by itself when branching an outcome into several modes is necessary.

The question then becomes, if we don't want to rely on the user to specify modes, how can we still leverage the prior work and its fast policy-building process? We need to infer the modes automatically, without extra information from the user. We also need to infer them efficiently – it would be inefficient for example to always branch 3 ways (or some other arbitrarily chosen number); it might not always be necessary to branch, as we explain below.

We propose to dynamically generate branches as needed. We do this by successively bisecting the probability distribution of the non-deterministic effect. Whether we generate a branch or not will depend on the success or failure of the weak plan from that branch outcome to the goal. This success or failure is dictated by θ as described in Section 2.

3.3 Representing the Single Mode

The core motivation of our work is to allow the representation of any probability distribution – not just a set of Gaussians – in non-deterministic action effects. We demonstrate that our ambition is computationally feasible and effective by using it to improve the policy-building process of prior work. Thus, we face the question – how do we represent a single, general probability distribution such that policy-building still works as efficiently as it did with Gaussian distributions?

We introduce the concept of a Bayesian Plan Network (BPN) as a representation of uncertainty at any point in the reachable search space. Its purpose is to check whether an action precondition holds given the uncertainty at the point of application.

The answer to the computation done by the BPN (a boolean – precondition holds or does not hold) is used to better inform state expansion about the uncertain environmental conditions. This is the case both in search and in RPG building. As part of this computation we use the certainty cutoff value θ explained in Section 2. We use the value $\theta = 0.9$ throughout the following to illustrate our approach. This is for demonstration purposes – the concepts we introduce work the same regardless of the value of θ.

3.4 Building a Bayesian Plan Network

The BPN can be described as a directed graph of nodes, where each node represents either a probability distribution d or a variable v. A BPN that corresponds to a given plan contains all the variables affected by that plan as they go through sequential changes (actions affecting them), together with their corresponding distributions (if an effect is not uncertain, its distribution is degenerate). The graph is weighted – coefficients from action effects are used to indicate how a variable depends on previous variables multiplied by constants.

The distribution nodes can be described as source nodes – they have no parent nods, as their value does not depend on other variables in the plan. They are akin to buckets of samples (whether described analytically, like the shape of a Gamma function, or empirically, like a collection of sensor measurements).

The variable nodes are essentially addition nodes. They have at least one parent node (which can be a distribution, or another variable). If a variable node is queried to obtain a sample of its value, it will in turn query its parents – this operation recursively all nodes in the PN once.

The steps for building a BPN are as follows:

1. Input a problem description and a weak plan found by the planning kernel.
2. For each variable set in the initial state, create one variable node and one distribution node for each one. Each distribution node is the parent of its corresponding variable node, and represents a degenerate distribution (only contains one sample, the value set by the initial state).
3. For each action in the weak plan, loop through its effects.
4. For each effect, create one new variable node for the affected variable. Then create parent links between the new node and all the variable nodes whose values are used by that effect (with their respective weights assigned to the parent link). Note that we keep track of the latest variable nodes at all times, to ensure the sequential changes to the variables are accurate.
5. If the effect above is non-deterministic, then create one

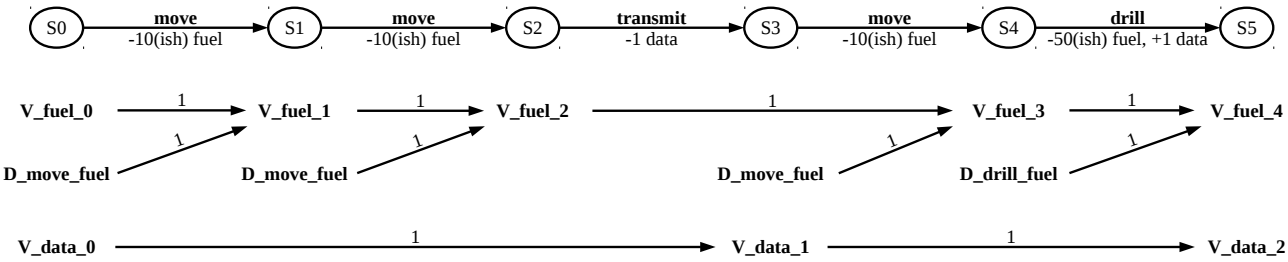

Figure 2: Weak plan and its corresponding BPN.

new distribution node containing the samples corresponding to that effect.

6. If the effect above uses a constant, then create one new distribution node containing a degenerate distribution (only contains one sample, the constant itself).
7. Each time a new node is created, sample its value a given N number of times and compute the median value. This median will be necessary when defining the regression operation in Section 3.5.
8. Each time an action with two modes (outcomes) is encountered, apply the operations above for each mode, then create one new variable node whose parents are the variable nodes of each mode, weighted equally at 0.5 each. This situation can arise when our algorithm cannot find a solution by using a single mode, and bisects the probability distribution at its median.

3.5 Integrating a Bayesian Plan Network with Prior Work

As the purpose of our work here is to allow the prior policy-building work to deal with general probability distributions, we present below how this can be achieved. Specifically, we introduce a novel way to perform regression.

The core idea of our contribution is to build a BPN and sample it to check whether an action's preconditions hold at any given stage.

The steps for building a policy by using the BPN to include general probability distributions are as follows:

1. Use the planning kernel to find a weak plan.
2. Use the weak plan to build a BPN as described in Section 3.4.
3. Use the BPN in the regression stepo to build a policy. This is where we use the BPN to check if preconditions hold, not with a Gaussian offset as in prior work, but with an offset obtained through repeated sampling of the BPN. We will expand on this step below.
4. Use the policy to decide which actions the planner should take.

Instead of the (partial state, action) pairs described in Section 2.2, the policy will now contain (partial state, list of actions) pairs. We obtain this list by storing the steps in the weak plan created when the partial state is expanded.

When checking if the policy knows what to do in a particular state, we first do the propositional triage. We look for all the partial states in the policy which match the facts in our particular state, and thus obtain a list of candidate matches.

To choose a candidate match, we need to recreate the functionality of regression, as we no longer have the option of computing regression analytically based on all uncertainty being Gaussian. We build a BPN from the actions in the plan-so-far up to our particular state, concatenated with the list of actions from our particular state to the goal (as mentioned above, the list of actions is found in the candidate pair). We then sample the BPN a certain amount of times – in our preliminary experiments, an amount of 1000 was suitable. By sampling the BPN we refer to sampling the goal state in the BPN, which will propagate backwards and eventually sample all nodes in the network. After each sampling, we loop through the plan that generated the BPN and check if at each step the preconditions hold, keeping track of each precondition's status with a counter. At the end of the 1000 sampling steps, we check if all the preconditions are satisfied with the required degree of certainty θ, e.g. if they are satisfied at least 90% of the time during those 1000 samples, based on our counters.

To improve the process of choosing a candidate match, we also order the list of candidates by heuristic value (Metric-RPG).

3.6 Representing a Plan Network Efficiently

The graphical representation of the BPN described in the previous section is useful to intuitively understand how the network functions. However, the sequential computations based on this representation slow down our approach and make it less competitive with prior work.

We thus propose a matrix representation in order to efficiently compute the answer to the central question in the section above – are all the preconditions in a given plan satisfied with certainty θ? Our method will allow each sample run to happen concurrently rather than sequentially, significantly reducing the time taken to compute the final answer.

The structure of the matrix stems from the topological order traversal of the BPN. Each row is a node, and each column is a sample run. For each distribution node we have a value of 1 in the column that corresponds to a sample from that distribution, and a value of 0 everywhere else. For each

variable node we have non-zero values in the columns that define that node's value in relation to the edges coming into it.

Then, to check if that BPN's corresponding plan is satisfied with certainty θ, we multiply our matrix representation with a matrix containing all the sampled values for all uncertain variables. We then use the result to count the number of sample runs in which all the preconditions were satisfied, as in Section 3.5.

3.7 Example

Consider a simple robot-waiter domain with two actions:

- The robot can move from the customer to the kitchen. This has deterministic propositional effects.
- The robot can move from kitchen to the customer, and pass the butter. The amount of butter passed is non-deterministic, according to some distribution, due to the accuracy of the robot's actuators.

From a modelling point of view, there is a single outcome mode for second of these actions: that some amount of butter is passed. In prior work (Marinescu and Coles 2016b) one would represent that as a single Gaussian-distributed outcome – there is no reason per se to use multiple Gaussian outcomes in the effect list of passing butter.

To ensure that enough butter is passed, with sufficient confidence, a strong plan could then be [move, pass-butter, move, pass-butter]. In the absence of multiple outcomes on passing butter (because there is no need to hand-prescribe multiple outcomes, from a user point-of-view), there would be no branching. As such, regardless of how much butter was passed, the expected solution length is four actions. A more efficient outcome would be to branch on the outcome of butter passing within the planner, rather than expecting this in the model. For instance, if with P(0.5) enough butter is passed, then a branch to execute the second round of moving and passing butter would reduce the expected solution length to $(0.5 \times 2) + (0.5 \times 4) = 3$ actions.

4 Evaluation

4.1 Evaluation Plan

As our work is still in progress, we will confine this section to presenting our evaluation plan.

First, we will compare our planner with the one used by (Marinescu and Coles 2016a). We will use the same numeric planning domains (such as `rovers`) for both planners, while taking into account that information about uncertainty is conveyed differently to these two planners. In prior work, the PDDL domain file contains the parameters of the Gaussian distribution associated with each non-deterministic effect. In our work, an additional data file contains a set of samples taken from a Gaussian distribution in a preprocessing phase.

This first comparison will establish whether our planner performs as well as prior work or better in problems where the uncertainty is genuinely a Gaussian (rather than poorly approximated as one). We expect these tests to confirm the computational feasibility of our work – even facing off against the fast analytical mathematics that are possible with Gaussians.

Second, we will take the prior work from (Marinescu and Coles 2016b) and compare it with our work in order to measure the impact of multi-mode versus single-mode action outcomes. As before, information about uncertainty is conveyed differently. The prior work once again hard-codes both the Gaussian parameters and the outcome modes into the PDDL domain file. For our work, we take in a data file containing samples from all the outcome modes (assuming that, for N modes, each mode has a $1/N$ likelihood to occur).

We make this second comparison in order to check how the removal of the hard-coded clues (the modes) impacts planner performance and solution certainty. One interesting metric to look at will be the expected probabilistic cost of the plan, as mentioned in Section 3.7.

Another useful indicator of performance will be the computational cost of planning. We are interested in comparing the process of finding a plan for the Gaussian case versus finding a plan for the general case. This will indicate whether the extra detail present in the general probability distribution impacts planner running time. We expect this not to be the case, due to the parallel computation approach we described in Section 3.6.

4.2 Potential Domains

We initially aim to test our planner on the same domains as prior work, namely:

1. `Rovers` from (Coles 2012), where the `move` action exhibits Gaussian uncertainty due to soil characteristics and potentially incomplete data on obstacles.
2. `AUV` from (Coles 2012) (modified to no longer be an over-subscription problem), where the `move` action outcome is drawn from a general probability distribution reflecting the influence of stochastic ocean currents.
3. `TPP` from (Gerevini et al. 2009), where the `purchase` action is drawn from a general probability distribution simulating the honesty of the merchants.

We are also actively seeking out domains from the planning applications community, in order to best illustrate the types of problems our approach is suitable for. Of particular interest are situations where probability distributions are skewed to either side of the median line, or exhibit modes (distinct areas on the graph) that are not easily distinguishable when modelling the problem. Intuitively, in these cases Gaussian approximations are not adequate.

4.3 Potential Outcomes

Extrapolating from the improvements obtained in (Marinescu and Coles 2016a) by adding approximate information about uncertainty into the heuristic, we expect that adding more accurate information about uncertainty into the heuristic will enhance the already-existing benefits. For example, we expect a more informed heuristic to discover dead ends faster (as it is not likely to lead search down risky paths). We also expect that, if placed side-by-side in a simulator,

our work would find more reliable solutions than (Marinescu and Coles 2016a) when measured by how frequently the solution obeyed the certainty threshold θ in the simulator.

Compared to the work in (Marinescu and Coles 2016b), on top of more accurate information about uncertainty, we expect our single-mode outcomes to lower the expected probabilistic cost of solution plans. We believe this is the case due to the planner only branching as-needed depending on the certainty threshold, rather than always branching on multiple pre-specified modes.

If the improvements outlined above do not occur, then our work provides evidence that the Gaussian approach taken in (Marinescu and Coles 2016b) is a good enough approximation of uncertainty – with the drawback of being reliant on explicit information on modes from the user.

Additionally, we expect the computational cost to be lower due to the efficient matrix implementation of the Plan Network outlined in Section 3.6.

If instead this cost turns out to be higher, it would mean that our implementation, in spite of parallel computing, cannot surpass the advantages offered by using analytic Gaussian mathematics.

5 Conclusions

5.1 Summary

In this paper, we introduced a novel way to represent numeric uncertainty at any point in the reachable search space. Our representation allows non-deterministic numeric effects to be drawn from any probability distribution, specified either in analytic form or as a collection of data samples. We described an efficient way to implement this representation that uses parallel computation and can make the most of GPU hardware. We integrated our approach with prior work on policy-building for non-deterministic planning, defining the regression operation through non-Gaussian effects.

While this is a work in progress and experimental evaluation is still pending, our research is an excellent opportunity to examine the precision / speed trade-off when it comes to generality. Accommodating general probability distributions requires less effort in terms of domain modelling and user input. Our work is able to take the most accurate probability distribution available (perhaps obtained by sampling data from previous runs), and make the most of that information, efficiently creating branches in non-deterministic outcomes as necessary to meet the certainty requirements.

According to our research, if information about uncertain probability distributions is available when writing the domain model, the planner should use it in its entirety rather than abstract it into a Gaussian distribution, or to a single median. With our implementation, it'll be tractable to do so, taking advantage of all available information.

If on the other hand information is not available to begin with, our approach is able to start out with a uniform or degenerate probability distribution, and refine it with time as more information is obtained (typically at plan running time).

5.2 Future Work

While at present the main application of our contribution is the policy-building process outlined in Section 3.5, our work can be used in the future to generate strong plans, in a similar vein to (Coles 2012) and (Marinescu and Coles 2016a).

In addition, our architecture allows the planner to not only generate a plan with θ certainty, but also to bump up θ to the highest value it can take under the given uncertain numeric effects. This is fairly straightforward to achieve – the query to the Plan Network that indicates success or failure can be modified to instead return the number of successful sample runs out of the total ones attempted.

The opposite of the above is also achievable in a similar fashion. If a solution with certainty θ is not found, our planner can be modified to return an alternative, lower value of θ for which a solution is found.

Another highly promising avenue for future work is learning probability distributions during execution. We can start out with a rough idea of a probability distribution – perhaps a Gaussian or a degenerate one if we lack any insight about the problem. We can then refine it to a more accurate distribution in a live feedback loop during plan execution. Our architecture allows changing distribution samples and parameters easily, so we expect the learning aspect to be a major selling point of our future work.

Acknowledgements

We would like to thank Amanda Coles for her insight into branching in the presence of uncertainty, and also for her critical analysis of our work.

Liana Marinescu's research is funded by a scholarship awarded by the Department of Informatics at King's College London.

References

Babaki, B.; Guns, T.; and Raedt, L. D. 2017. Stochastic constraint programming with and-or branch-and-bound. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*.

Beaudry, E.; Kabanza, F.; and Michaud, F. 2010. Planning with concurrency under resources and time uncertainty. In *Proceedings of the Nineteenth European Conference on Artificial Intelligence*.

Beck, J. C., and Wilson, N. 2007. Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research*.

Coles, A. J. 2012. Opportunistic branched plans to maximise utility in the presence of resource uncertainty. In *Proceedings of the Twentieth European Conference on Artificial Intelligence*.

Gerevini, A.; Long, D.; Haslum, P.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth International Planning Competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*.

Hoffmann, J. 2003. The Metric-FF planning system: Translating ignoring delete lists to numeric state variables. *Journal of Artificial Intelligence Research*.

Marinescu, L., and Coles, A. I. 2016a. Heuristic guidance for forward-chaining planning with numeric uncertainty. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling*.

Marinescu, L., and Coles, A. I. 2016b. Non-deterministic planning with numeric uncertainty. In *Proceedings of the Twenty-Second European Conference on Artificial Intelligence*.

Marinescu, L., and Coles, A. I. 2016c. Non-deterministic planning with numeric uncertainty. Technical report, King's College London.

Meuleau, N.; Benazera, E.; Brafman, R. I.; Hansen, E. A.; and Mausam, M. 2009. A heuristic search approach to planning with continuous resources in stochastic domains. *Journal of Artificial Intelligence Research*.

Muise, C. J.; Belle, V.; and McIlraith, S. A. 2014. Computing contingent plans via fully observable non-deterministic planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*.

Muise, C. J.; McIlraith, S. A.; and Beck, C. J. 2012. Improved non-deterministic planning by exploiting state relevance. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*.

Santana, P.; Thiebaux, S.; and Williams, B. 2016. RAO*: an algorithm for chance constrained POMDPs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*.