Forward Policy Building For Planning With Numeric Uncertainty

Marinescu, Liana

Awarding institution:
King's College London

Forward Policy Building For Planning With Numeric Uncertainty

Liana-Eleonora Marinescu

Supervisor: Andrew Coles
Department: Informatics
Institution: King's College London

December 16, 2022

Acknowledgements

I am grateful to:

- my supervisor Andrew Coles for seeing this entire adventure through to the end, for helping me express my ideas both in code and on paper, and for being endlessly patient while I found my footing in the world of AI research

- my second supervisor Amanda Coles for helping me refine the focus of my work early on by sharing her valuable insight on branched planning

- the Department of Informatics at King's College London for funding my studies and providing me with a workspace

- planning researchers who shared their ideas with me over the years, in particular Christian Muise for passing on to me his excitement about non-deterministic problems, Sara Bernardini for supervising my MSc and introducing me to my greatest motivator, planning for robotics, and Malte Helmert for reassuring me that my ideas on environmental robotic exploration were interesting and worth pursuing

- all the former and current members of the Planning research group at King's College London – Maria Fox, Derek Long, Andrew Coles, Amanda Coles, Daniele Magazzeni, Bram Ridder, Michael Cashmore, Josef Bajada, Chiara Piacetini, Wiktor Piotrowski, Emre Savas, Atif Talukdar, Elizabeth Black, Christopher Hampson, Tanja Daub, Stefan Edelkamp, Elad Denenberg, Moises Martinez, Senka Krivic, Gerard Canal, Ionut Moraru, Parisa Zehtabi, Rares Buksz, Fares Alaboud, Adam Green, Anna Collins, Ben Krarup, Yassin Warsame – for listening to my talks and giving me useful feedback on my work

- my mum and dad for always encouraging me to pursue what makes me happy even if it means living a thousand miles away
- my partner Emil for personally making sure I have a happy and healthy life outside of my research, and for helping me see the big picture instead of overthinking all the things
- my dear friend Tanja for gently talking me out of impostor syndrome and reassuring me that both my work and I matter and have the potential to make a difference
- my dear friend Konrad for listening to my rants and reminding me to enjoy this weird and wonderful time being a PhD student
- my dear friend Yani for keeping me accountable to my research goals and exerting the most well-intentioned variety of peer pressure
- my dear friends in Zürich, London, and the world over – Cami, Anca, Ana, Everest, Polina, Aldo, Mai, Carol, Michał, Alessio, Aleks, Maggie, Emil, Eric, Fayyaz, Stefan, Exie, Richard, Mwachala, Manu, Silvia, Julian, David, Joris – for being awesome and supportive
- my hosts, mentors and managers at Google – Ant, Mike, Simon, Arvind, Gabriel, Aurélien – for teaching me better ways of working that helped my research in unexpected ways
- Carl Friedrich Gauss for introducing my favourite probability distribution in 1809 in order to rationalise the least squares method

Abstract

Uncertainty hinders many applications of AI – algorithms are often faced with noisy sensors, unpredictable environments, or known limitations in world models.

Automated planning with numeric uncertainty is an active research field that can tackle some of these challenges already. However, areas such as computing heuristics and building policies still pose open questions.

In this thesis, we aim to improve planner performance on domains with numeric uncertainty. To this end, we implement several techniques into a novel planner based on the existing Optic+. Our contributions are connected by an over-arching theme: tracking uncertainty and integrating it with aspects of planning that previously ignored it.

We begin by improving heuristic guidance for forwards planning with continuous random variables, by ensuring preconditions are met with a given degree of confidence. With this new information, the planner can also consider acting to reduce accumulated error. We then go on to define regression semantics for the case where uncertainty is Gaussian, allowing forward-policy building to be extended to handle not just propositional uncertainty but also numeric uncertainty. We also propose an internal representation that allows the planner to sample action effects from any probability distribution. As a consequence, the world model can be refined without sacrificing computational time by adding more samples during execution.

While our original motivation – and our running example throughout this thesis – is robotic exploration, our contributions are general and not limited to one type of problem. Ultimately, we reduce the effort needed to describe uncertain domains and show how planners can tackle a changing environment while meeting given certainty requirements.

Contents

Chapter 1

Introduction

In this chapter we explain the reasons for choosing automated planning under uncertainty as the topic of our research. We state the purpose of this thesis, and the scientific contributions it contains. We also give a brief chapter-by-chapter overview.

1.1 Motivation

Automated planning is a rich and active research field within artificial intelligence. It is a fast moving landscape with many open problems and frequent incremental progress.

Planning is arguably one of the strongest fields of AI when it comes to choosing, ordering and scheduling high level tasks. Its success is due in part to the powerful guarantees it can offer – solution existence, correctness, optimality – within reasonable time bounds. This makes it appeal to areas like robotic systems, where such guarantees are often what stands between design and deployment. However, planning is more difficult to apply here, due to the uncertainty inherent in some robotics domains. Physical systems often face sensor noise, operate in unpredictable environments, or have known limitations in their world models. Planning under uncertainty is the area that aims to tackle these difficulties.

We believe the subfield of planning under numeric uncertainty poses a set of research questions which are worth exploring. In particular, we are interested in how uncertainty can be represented and tracked, and how it impacts the construction of plans and policies.

As mentioned above, we also believe planning under numeric uncertainty is relevant to tangible issues that robotic systems grapple with today. Applications such as autonomous driving, drone delivery, and planetary exploration could benefit from our research, as they are strongly affected by uncertainty. For example, after a stretch of rough terrain, a truck may have lost some of its payload. Or, after mapping an unknown crater, a rover may have used a higher amount of battery than expected. Our work targets issues like these in order to broaden the range of problems that automated planners are able to tackle.

1.2 Statement of Thesis

For the reasons outlined above, we chose to work on planning with numeric uncertainty. In broad terms, the aim of our research is to allow AI planning software to account for accumulated error due to numeric measurements. We offer the planner as much information about uncertainty as we have available, and then implement and explore means to make the most of that information.

Our work targets fully observable, non-deterministic planning problems. Uncertain numeric effects are drawn from known probability distributions, either Gaussian or arbitrary. We search for a plan or a policy depending on whether actions have a single outcome or multiple outcomes. The following table indicates which cases we explore in which chapters:

	Gaussian distribution	Arbitrary distribution
Single outcome	Chapter 3	Chapter 5
Multiple outcomes	Chapter 4	Chapter 6

The question this thesis answers is the following:

Can we improve planner performance by tracking numeric uncertainty and taking it into account at different stages of the planning process?

We show how to track uncertainty in a computationally feasible manner, regardless of the probability distribution from which it is drawn. We integrate uncertainty into elements of planning that did not previously take it into account, and present experimental results that show planner performance is indeed improved. The specific elements we contribute to are detailed in the following subsection.

Initially we introduce tracking techniques which assume the distribution has a Gaussian shape. Later we introduce a richer representation that allows us to track uncertainty no matter what shape the distribution has. In both cases we adapt existing elements of the planning process or introduce novel ones in order to make use of the newly tracked information, as detailed below.

1.3 Contributions

For each of our contributions we enumerate the following information:
– a short summary
– the corresponding chapter
– the corresponding publication
– when an uncertain numeric effect occurs, what probability distribution it is drawn from (Gaussian or arbitrary)
– when an uncertain action occurs, how many discrete outcomes it has (single or multiple)

- **Contribution 1: Guiding Search Under Fully Observable Gaussian Uncertainty**
  – Improving heuristic accuracy
  – Allowing the use of uncertainty-reducing actions
  – We introduce the concept of offset to express how well preconditions hold when faced with numeric uncertainty from a known probability distribution. We use this offset to compute a novel heuristic based on prior work; our heuristic is aware of how accumulated uncertainty affects preconditions further down the line. Our approach also allows the planner to strategically use actions that reduce accumulated uncertainty, if such actions exist and uncertainty is Gaussian. Results show that our more informed heuristic performs better than the unmodified heuristic.
  – Chapter: 3
  – Publication: Marinescu, L., & Coles, A. I. (2016). Heuristic guidance for forward-chaining planning with numeric uncertainty. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling*

– Distribution: Gaussian
– Outcomes: single

- **Contribution 2: Building Policies Under Fully Observable Gaussian Uncertainty**
  – Defining numeric regression
  – Defining numeric dead end generalisation
  – We define the semantics of regression through numeric effects that exhibit Gaussian uncertainty. This process of regression is the core of policy building – we are therefore able to extend prior work on policy building from the propositional case to the numeric case. Our approach successfully leverages state relevance, the crucial element that makes policy building tractable by keeping the branching factor low. Results show that our implementation builds numeric policies faster than a baseline that only uses propositional regression.
  – Chapter: 4
  – Publication: Marinescu, L., & Coles, A. I. (2016). Non-deterministic planning with numeric uncertainty. In *Proceedings of the Twenty-Second European Conference on Artificial Intelligence*
  – Distribution: Gaussian
  – Outcomes: multiple

- **Contribution 3: Guiding Search Under Fully Observable, Arbitrarily Distributed Uncertainty**
  – Tracking arbitrarily distributed effects
  – We propose a representation that allows us to sample uncertain action effects from any probability distribution. We use a directed graph to capture how the accumulated uncertainty at any point in a plan depends on previous action effects. We apply our approach to find a strong plan that meets given certainty requirements. Results show that a strong plan that represents uncertainty accurately can be found comparatively fast, and sometimes faster than a policy that approximates uncertainty.
  – Chapter: 5
  – Publication: Marinescu, L., & Coles, A. I. (2018). Representing general numeric uncertainty in non-deterministic forwards planning. In *Proceedings of the Workshop on Heuristics and Search for Domain-Independent Planning at the Twenty-Eighth International Conference*

on Automated Planning and Scheduling
– Distribution: arbitrary
– Outcomes: single

- **Contribution 4: Building Policies Under Fully Observable, Arbitrarily Distributed Uncertainty**
  – Allowing policy building to function in the absence of an analytic form of uncertainty
  – We implement an alternative to our prior policy building approach, for cases when uncertainty is not Gaussian. The technique of numeric regression defined previously cannot be applied, as it relies on having an analytic representation of uncertainty. Instead we use a sampling-based approach in order to check whether or not a given state matches any existing policy entry. Results show that policy building performance is generally better when uncertainty can be represented analytically.
  – Chapter: 6
  – Distribution: arbitrary
  – Outcomes: multiple

1.4 Reader's Guide

Chapter 1 explains why this thesis exists. It presents the motivation behind our work in the broad terms of applied AI. It then states the goal of the thesis, and enumerates the novel contributions we made in pursuit of the goal.

Chapter 2 begins with a brief history of automated planning. It then narrows the focus down to planning with uncertainty, introducing the core concepts necessary for the rest of this thesis. It also surveys prior work done in our areas of focus, and elaborates on the work we build upon in the following chapters.

Chapters 3, 4, 5 and 6 each present one of the thesis contributions described in the previous section. They begin by going further into detail on the prior work that forms the basis of each approach. They then describe our proposed ideas and their implementation, and afterwards present the experimental results.

Chapter 7 circles back to our original motivation and sums up the results and takeaways of our work. It wraps up by laying out what may be promising future research directions based on our findings.

Chapter 2

Background

We begin this chapter with a look at the challenges intelligent agents face due to their environments. We then give a high-level overview and a very brief history of seminal work on classical planning. We gradually narrow the focus down to planning with uncertainty, introducing the core concepts necessary for the rest of the thesis. We also conduct a broad survey of prior work done in our research area, and elaborate more deeply on the particular work we build upon in the following chapters.

2.1 Artificial Intelligence

The field of artificial intelligence has a fruitful history of enabling machines to solve problems more efficiently and safely than humans. At its core, AI relies on computers being able to sift through more incoming data, more possible choices, and more hypothetical outcomes than a human brain ever could. Intelligent algorithms have been successfully used in applications ranging from human-like game agents and clever marketing recommendations, to mission-critical spacecraft control and disaster recovery. We believe that some of AI's most interesting challenges lie in risky and dynamic real-world situations such as the latter.

Currently, application-specific AI is the most prevalent – for instance, a robot might navigate a minefield, but not a flight of stairs, or a drone might survey a vast woodland, but not an indoor maze. In order to progress toward more general intelligent agents, it is paramount to understand and tackle the issues arising from the non-ideal, ever-changing, risky environments in which agents may operate. To this end, we detail below the formal way to describe an environment in the context of AI.

According to the work of Russell and Norvig (Russell & Norvig, 2010), whose focus is the intelligent agent in a sense-act loop, there are seven ways to categorise an environment:

- **Deterministic / Non-deterministic** - A deterministic environment responds predictably to the agent's decisions: given the current state and the agent's chosen action, it is possible to infer the next state with absolute certainty. For example, completing a jigsaw puzzle can be considered deterministic. In contrast, when acting upon a non-deterministic environment, there may be several possible outcomes. If these outcomes have probabilities associated with them, the environment may be referred to as **stochastic**. For instance, the game of backgammon and the stock exchange are both non-deterministic. In particular, backgammon is stochastic, because rolling the dice has a fixed set of outcomes with known probabilities (unless of course the dice are loaded).

- **Static / Dynamic** - A static environment maintains a constant state if the agent takes no action. In other words, the agent can count on the environment not to change while it spends time considering what to do next. Playing solo chess or solitaire are both examples of this. On the other hand, a dynamic environment might change independently of the agent, in the time between two of its consecutive actions. Tasks such as checking items on an assembly line, or vacuuming a room with people in it, are dynamic.

- **Discrete / Continuous** - A discrete environment has a finite number of possible states, observations, and actions. For example, the finite number of states might be positions a grid; the observations might be limited to `goal` / `not-goal`, or to `colour-red` / `colour-green` / `colour-blue`; and the actions might be `move-north` / `move-south` / `move-east` / `move-west`. Time steps can also be viewed as discrete, occurring once every period T. On the other hand, in a continuous environment, these possibilities are not finite. States, observations, actions, and time can all be expressed in terms of real numbers, on a scale as finely-grained as necessary. For example, an agent might observe a luminosity value anywhere between 0 and 1, and might be able to turn at an angle between 0 and π.

- **Fully Observable / Partially Observable** - In a fully observable environment, the agent is able to acquire all the information relevant to its task. In other words, at any given time, the state of the environment is completely known to the agent. This might be the case in an image analyser, or a top-down maze game. The other possibility is that the environment is partially observable. In this case, the agent's sensors might be unreliable, or they might only be suited to gathering part of the relevant information. For example, a game agent might only see opponents if they are within a certain range, and a vacuuming robot might be unable to sense dirt if it is smaller than its sensors' resolution.

- **Episodic / Sequential** - In an episodic environment, the outcome of an action only depends on factors within the current episode. Episodes are typically sense-act pairs, which means that there is no need for planning ahead in this type of environment. An example would be the task of separating coloured balls into boxes, or deciding whether text is spelled correctly. In contrast, in a sequential environment, the current outcome might depend on all the agent's past actions. Deciding which roads to take to a waypoint, or in what order to place domino pieces, are both examples of sequential environments.

- **Known / Unknown** - For an environment to be known, the agent needs to be aware of what the possible outcomes of its actions are. This can be paraphrased as the agent knowing the rules of the game. For example, if an agent knows how Tetris is played, it will be certain that pressing "right" will move the current piece right, and that pressing "turn" twice will turn the piece upside-down. On the other hand, if an agent is placed in an environment with only a vague idea of its laws, it will likely have to experiment with each action in order to observe some possible outcomes.

- **Single-agent / Multi-agent** - In a single-agent environment, there is only one entity that strives to achieve a goal. Some straightforward examples might be a robot picking up items from the floor, or a robot building a stack of blocks. A multi-agent environment is the obvious alternative: there are multiple intelligent agents, each with their own goals. If the goals overlap, the environment is cooperative, and if they clash, it is competitive. Coordinated drone flight is an example of the former, and a multiplayer board game illustrates the latter.

The focus of this thesis is on non-deterministic environments (which we also refer to as uncertain environments). As for the other characteristics, in general, the domains we use for experiments are:

- Static
- Continuous
- Fully Observable
- Sequential
- Known
- Single-agent

2.2 Automated Planning

Automated planning is a field of AI that allows an intelligent agent to generate a sequence of actions leading to a goal. The use of planning renders a system fully autonomous; the only instance of human intervention occurs in the very beginning, when devising a model of the problem in a manner understandable by the planner. As described by Geffner *et al.*, such a model includes information about the state space an agent is in; the initial state and goal state; all possible actions the agent can take, together with their costs; and a transition function which dictates how an action leads from one state to another (Geffner & Bonet, 2013). Formally, a planning problem is a tuple $\langle F, I, G, A \rangle$ where:

- F is a set of propositional facts.
- I is a known initial state: a subset of facts from F.
- A *condition* is a conjunction of a subset of facts from F.
- G is a set of goal conditions.

- A is a set of actions, each $a \in A$ with:
  - $\mathit{Pre}(a)$: a set of conditions on the execution of action a.
  - $\mathit{Eff}^+(a)$, $\mathit{Eff}^-(a)$: sets of facts from F added and deleted by action a, respectively.

One of the earliest instances of what is now known as automated planning was introduced by Fikes and Nilsson (Fikes & Nilsson, 1971). Their software, STRIPS (Stanford Research Institute Problem Solver), used theorem proving and means-end analysis to solve problems, and described the world model using first-order logic. Uncertainty was not yet taken into consideration, however Schoppers *et al.* later described the novel idea of a universal plan that can guide behaviour in an unpredictable environment (Schoppers, 1997). Their work came at a time where the state of the art consisted largely of manual robot programming. The crucial difference to the STRIPS approach was that universal planning problems have no initial state.

Another early innovation, albeit not strictly to classical planning, was made by Littman *et al.* in the context of game agents (Littman, 1994). They proposed a framework for Markov games in which, instead of one agent interacting with an environment (that might contain another agent), two agents whose goals might be opposed share the same environment.

A large variety of planning software continued to be developed, the vast majority predicated on search, as surveyed in an extensive state-of-the-art study (Hendler, Tate, & Drummond, 1990). The prevailing strategy was to search the state space for a goal state, then extract the sequence of nodes from the initial state to the goal state. The state explosion problem was (and still is) one of the major challenges of planning, and it is typically addressed by pruning the state space with the appropriate heuristics. One innovative breakthrough that avoided the need to explore the state space fully was Graphplan (Blum & Furst, 1997). Their introduction of the planning graph (a compact structure alternating fact layers and action layers) allowed them to strongly outperform prior work.

After succeeding at the level of propositional planning, techniques for planning under uncertainty were developed as well. Anderson *et al.* introduced the concept of factored expansion to split actions with conditional effects into several components. This allowed Graphplan to solve problems with conditional effects faster than if it had used the default full expansion (Anderson, Smith, & Weld, 1998). Smith *et al.* introduced Conformant

Graphplan to handle types of problems that could not previously be solved due to lack of sensory information (Smith & Weld, 1998). Because of uncertainty in the initial state, during solution extraction, Conformant Graphplan considered the possible side effects of an action in other possible worlds.

Around the same time, sensing actions were investigated by Weld *et al.*. They introduced SGP, an extension of Graphplan that solved contingent problems (Weld, Anderson, & Smith, 1998). The crucial contribution of SGP was that it could distinguish between actions that sense a propositional value without changing it, and actions that change the value.

Probabilistic planning was then introduced to Graphplan by Blum *et al.* with two different approaches (Blum & Langford, 1999). PGraphplan produced an optimal contingent plan and TGraphplan produced a policy not guaranteed to be optimal. PGraphplan was slower than the original and TGraphplan was on par.

Soon after, Hoffman and Nebel introduced the highly successful planner FF (Hoffmann & Nebel, 2001). Similarly to prior work, they also used a heuristic based on ignoring delete effects. However, FF did not assume facts to be independent when computing the heuristic. Additionally, FF incorporated a combination of hill climbing and systematic search into its algorithm, which contributed to its winning performance at the IPC (International Planning Competition) 2000 competition (Bacchus, 2001).

2.3 PDDL

The standard language for representing problems in the formal model described above is the Planning Domain Definition Language (PDDL) (McDermott, Ghallab, Howe, Knoblock, Ram, Veloso, Weld, & Wilkins, 1998). The main PDDL constructs are variables, actions and predicates, which initially allowed classical planning environments to be described. The language was used for the first time at the AIPS (Artificial Intelligence Planning and Scheduling) competition in 1998. McDermott *et al.* wrote about the syntax and capabilities of the language, its expressive advantages compared to previous languages, and its potential to be useful beyond just the competition. Indeed, PDDL is currently used as input to most planning software, including the one presented in this thesis.

PDDL2.1 evolved in tandem with the AIPS competition (subsequently renamed IPC), and went on to support time and numeric resources. This broadened the scope of planning significantly, and opened the door to our current research for the first time.

Fox and Long wrote about the extended language PDDL2.1 in great depth, and reflected on the path taken by the planning community towards the development of the language (Fox & Long, 2002). They pointed out how the competitions (the third and most recent one in particular) were a strong driver for the additions introduced in PDDL 2.1. In further support of extending the language, they later analysed the IPC 2002 results in light of different levels of expressive power (and hence problem difficulty) offered by PDDL 2.1 (Long & Fox, 2003).

In a similar vein, Gerevini *et al.* presented a version of the new PDDL3 used in IPC 2006, which could additionally express hard and soft constraints (Gerevini, Long, Haslum, Saetti, & Dimopoulos, 2009). They helped establish the extended language by comparing the competing planners side-by-side in terms of how successfully they made use of the new PDDL3 features. The IPC continues to drive PDDL extensions by posing increasingly realistic and diverse problems.

2.4 Numeric Planning

A major development that allowed early automated planning paradigms to be applied to a vast number of real-world problems was the extension from propositional variables to numeric variables. Building on top of the formalism described in Section 2.2, a numeric planning problem as can be expressed in PDDL 2.1 is a tuple $\langle F, \mathbf{v}, I, G, A \rangle$ where:

- F is a set of propositional facts.
- $\mathbf{v}$ is a vector of numeric variables.
- I is a known initial state: a subset of facts from F and a value for each variable in $\mathbf{v}$.
- A *condition* is a conjunction of a subset of facts from F and constraints on $\mathbf{v}$. Each constraint is written as $(\mathbf{w}.\mathbf{v} \geq c)$, where c is a real value and $\mathbf{w}$ is a vector of real values.
- G is a set of goal conditions.

- A is a set of actions, each $a \in A$ with:
    - $Pre(a)$: a set of conditions on the execution of a.
    - $Eff^{+}(a)$, $Eff^{-}(a)$: sets of facts from F added and deleted by a, respectively.
    - $Eff^{num}(a)$: a set of numeric variable updates that occur when applying a. Each is of the form $\langle v\ op\ e \rangle$ where $op \in \{+=, =\}$ and e is a real value.

A notable planner that leveraged this upgrade was Metric-FF (Hoffmann, 2003). Its authors introduced an extension to the STRIPS heuristic approach (which relaxed the problem by ignoring delete effects). This extension allowed the existing heuristic to be applied to numeric planning problems by computing a relaxed reachability analysis that ignored decrease effects on an upper bound of each variable, and ignored increase effects on a lower bound of each variable. Their technique applied as long as the numeric effects were monotonic, which was most often the case. Further, they showed how linear numeric effects can be processed so they can be treated as monotonic. This approach made Metric-FF succeed at the IPC 2002 and become one of the two most efficient numeric planners at the time. We will expand more on the functionality of the Metric RPG heuristic in Chapter 3.

The other numeric planner that saw great results at IPC 2002 was LPG (Gerevini, Saetti, & Serina, 2003). LPG was based on stochastic local search and could handle temporal as well as numeric constraints. Gerevini *et al.* first introduced Temporal Action Graphs to represent time constraints and performed search in that graph representation, and later introduced Numeric Action Graphs (Gerevini, Saetti, & Serina, 2004) to leverage a similar technique for numeric expressions.

Another high performing forward search planner is Fast Downward, which won the IPC in 2004 (Helmert, 2006). Instead of the PDDL representation of the planning task, their approach uses an alternative SAS+ representation (Bäckström & Nebel, 1995) which allows them to use the novel causal graph heuristic (which, interestingly, is not based on ignoring delete effects).

2.5 Planning With Uncertainty

As we have discussed so far, there are certain classes of problems within AI that planning excels at. One of these classes is that in which the order of actions is critical, for example playing a strategy game, or assembling a Lego structure. Another class is that of problems with many timing and scheduling constraints – actions must be taken at exactly the right time in order to be successful. An example would be underwater navigation: an autonomous vehicle needs to plan which actions to use and when to use them, while at the same time accounting for ocean currents, low visibility, or fuel constraints. Another example might be a robot that navigates through areas of sunlight and shade, and must time its recharging actions just right to the moments of sun exposure in order to maximise its battery life.

There are, however, other areas of AI that planners are increasingly able to tackle, though they are not as straightforward to model as the examples above. They require the development of more complex techniques such as abstracting parts of the world in a way that preserves information about uncertainty; or coming up with new heuristics that more accurately estimate future uncertain effects. Such problems might be, for example, failure-prone navigation, or unguided exploration of large unknown areas. More generally, they might be situations where, in the face of incomplete world dynamics, something needs to be done towards acquiring and leveraging information about uncertainty.

2.5.1 Early Approaches

There is a long history of work setting out to tackle uncertainty in planning with a range of approaches. One seminal piece of work was that of Kaelbling *et al.*, whose approach was rooted in operations research techniques (Kaelbling, Littman, & Cassandra, 1998). The authors introduced novel ideas to the then-young field of planning in stochastic, partially observable environments. They opted for a probabilistic representation of the environment, which is perfectly suited for scenarios where the current state or the action outcomes lie in a probability distribution. The problem representation they chose is an MDP (or a POMDP for the partially observable case). Their work has served as the basis for many subsequent efforts using MDPs to keep track of the world.

In the same year as Kaelbling *et al.*, Cimatti *et al.* presented a novel algorithm for non-deterministic domains (Cimatti, Roveri, & Traverso, 1998). Their algorithm produced universal plans; it generated solutions guaranteed to reach the goal, if they existed; and it generated iterative trial-and-error strategies otherwise. The latter were guaranteed to reach the goal if the iterative loops eventually terminated.

Similarly to universal plans, strong cyclic plans as defined by Daniele *et al.* are iterative plans that have a possibility to terminate, and when they do they are guaranteed to succeed (Daniele, Traverso, & Vardi, 1999). Such plans are necessary in non-deterministic planning domains where an effect is not guaranteed apriori (meaning that a plan might end up in an infinite loop). Daniele *et al.* introduced an algorithm to generate such plans based on model checking, and proved their algorithm always found a strong cyclic plan if one existed. They also introduced the notions of strong plan (non-iterative plan guaranteed to succeed), and weak plan (non-iterative plan with a possibility to succeed), which we will refer back to later in this thesis. Cimatti *et al.* also implemented means to generate weak, strong, and strong cyclic plans using model checking techniques (Cimatti, Pistore, Roveri, & Traverso, 2001).

2.5.2 Conformant Planning

Another major research direction within planning under uncertainty is conformant planning – planning in the case when the initial state and action effects are uncertain, and no sensing possibilities are available. Brafman and Hoffmann introduced conformant planning capabilities in the existing FF planner mentioned in Section 2.2 (Brafman & Hoffmann, 2004) (Hoffmann & Brafman, 2006). They presented a new representation of belief space, which allowed them to integrate new information into the FF heuristic. Instead of storing sets of possible worlds in the search space, they reasoned about the effects of action sequences. Conformant-FF outperformed all other existing conformant planners at the time.

Further developments to conformant planning were made soon after. One introduced an alternative way to prune the search space by checking the validity of a partial plan (checking if it is consistent with the theory and with each possible initial state) (Palacios, Bonet, Darwiche, & Geffner, 2005). Another extended prior work on translations to account for the fact that all classical plans are conformant but not vice-versa (Palacios & Geffner, 2007). A translation approach also formed the basis of the planner T0 which won the IPC in 2006 (Palacios & Geffner, 2009). Instead of solving the problem by searching in belief space, Palacios *et al.* introduced translations that map literals and assumptions together in a "literal true if assumption initially true" layout.

2.5.3 Probabilistic Planning

Another large subfield of planning under uncertainty is probabilistic planning – when the planner has information available about the probabilities of uncertain action effects or events. Younes *et al.* introduced the language PPDDL (Probabilistic PDDL), which is a variant of PDDL that allows the modelling of probabilistic problems with rewards (Younes & Littman, 2004). This language was used at the IPC in 2004.

A short example of a probabilistic action with two possible outcomes in PPDDL might look like the following, based on the Bomb and Toilet problem:

```
(:action dunk-package
   :parameters (?pkg)
   :effect (and (when (bomb-in-package ?pkg) (bomb-defused))
                (probabilistic 0.05 (toilet-clogged))))
```

Little *et al.* tackled the still-emerging field of probabilistic temporal planning, where problems contain durative actions, concurrency, and probabilistic effects (Little, Aberdeen, & Thiébaux, 2005). In particular, they targeted the gap in existing research around problems where time and uncertainty interact – effects are probabilistic, their start time is probabilistic, and their duration is probabilistic.

Other successful probabilistic planners were mGPT (Bonet & Geffner, 2005), which solved MDPs by extracting bounds from deterministic problem relaxations; and ProbabilisticFF (Domshlak & Hoffmann, 2006) (Domshlak & Hoffmann, 2007), which extended Conformant-FF to probabilistic domains with uncertain initial state and action effects.

There was a surprising victory at the 2004 and 2006 IPC probabilistic tracks by the planner FF-Replan (Yoon, Fern, & Givan, 2007), compared against the major probabilistic planners mentioned previously. FF-Replan first constructed a deterministic version of the probabilistic problem, then called the planner FF on this version until an unexpected effect took place – and then replanned. As a short example, the all-outcomes determinisation of the probabilistic `dunk-package` action above would produce the following two actions:

```
(:action dunk-package-unclogged
   :parameters (?pkg)
   :effect (and (when (bomb-in-package ?pkg) (bomb-defused))))

(:action dunk-package-clogged
   :parameters (?pkg)
   :effect (and (when (bomb-in-package ?pkg) (bomb-defused))
                (toilet-clogged)))
```

The authors of FF themselves noted its simplicity and its unexpectedly good performance on the competition benchmarks. This prompted Little *et al.* to conduct a comprehensive theoretical comparison of probabilistic planning and replanning in the context of the IPC. They introduced a way to measure the probabilistic interestingness of a problem, and suggested improvements that could be made to the competition in the future (Little & Thiébaux, 2007).

Other notable work on probabilistic planning was the planner HMDP (Keyder & Geffner, 2008) which introduced two novel heuristics, and the planner FF-Hindsight (Yoon, Ruml, Benton, & Do, 2010) which reused previously generated plans in order to scale well in an online setting. Here, *online* refers to planning while acting, e.g. updating the plan in response to information that was unavailable at the start. This is in contrast to *offline* planning, where the plan is generated before acting and cannot be changed.

PROST (Keller & Eyerich, 2012) was also a successful probabilistic planner based on the prior UCT (Kocsis & Szepesvári, 2006), which used bandit-based techniques to plan even in the absence of a world model.

The probabilistic track of the IPC from 2010 onwards has gone on to use the language RDDL (Sanner, 2010), or subsets or compilations of RDDL. This language describes fully or partially observable processes (discrete or continuous, with possible concurrency) in a way which resembles dynamic Bayes nets, and constitutes an effort to standardise probabilistic domain languages. The transition of the planning competition from PPDDL to RDDL allowed the introduction of domains with concurrency and stochastic events, e.g. traffic control or wildfire management (Vallati, Chrpa, Grześ, McCluskey, Roberts, & Sanner, 2015).

2.6 Planning With Numeric Uncertainty

The central focus of this thesis is non-deterministic numeric planning – when there is uncertainty over the amount of resources used by actions. One planner whose approach inspired us was by Meuleau *et al.*, who introduced the novel algorithm HAO* as a generalisation of prior work on continuous stochastic resources (Meuleau, Benazera, Brafman, Hansen, & Mausam, 2009). They followed the classic heuristic search approach and noticed that taking continuous resources into account helps limit the search to only states that are likely reachable given the stochasticity of the resources. To show the success of their approach they applied HAO* to a planetary rovers problem, which shares many similarities with the rovers domain we later use for our own experiments.

Below we summarise two particular approaches that formed the basis of our research on planning with numeric uncertainty. In Chapter 3 we go further into detail on the specific concepts from these approaches that we use in our work.

2.6.1 The Bayesian Network Approach

One piece of prior work we draw heavily on is by Beaudry *et al.* - their core contribution was introducing time and resource uncertainty in the form of a Bayesian network into the planner RTU (Beaudry, Kabanza, & Michaud, 2010b) (Beaudry, Kabanza, & Michaud, 2010a). Their work was motivated by the fact that, thus far, planning could deal with problems with concurrent actions and resource uncertainty, but in the case of time uncertainty, the norm was to discretise it. The issue with time discretisation was the unmanageably large search space, once the uncertain durations and concurrent actions were taken into account.

This Bayesian network approach that Beaudry *et al.* introduced is one we later use ourselves in order to represent the influence of previous uncertain effects on the current state (we thus inherit the "Bayesian" terminology from Beaudry *et al*).

In general, a Bayesian network (or "belief network") is a directed acyclic graph where each node is a random variable, and each edge is a dependency between two random variables. A Bayesian network can be used to answer probabilistic questions about the random variables it contains. For instance, later in this thesis, we use a Bayesian network to answer the question "is this precondition satisfied with more than X% probability, given the uncertainty in the plan so far?". We will be explaining how to build a Bayesian network of this form in Section 5.3.3.

In the work of Beaudry *et al.*, random variables were used to represent the current belief of resources and time, and a Bayesian network was used to model the relationships between these variables. This Bayesian network was built dynamically during search, by applying states to actions. The Bayesian network was then queried to estimate the probability of plan success. The authors applied their approach on a truck delivery domain where driving time and fuel usage were stochastic. In our later experiments we use a non-temporal version of this domain, as our contributions focus on numeric rather than temporal uncertainty.

2.6.2 The Pessimistic Planning Approach

A further piece of work we base many of our ideas upon is that of Coles, who introduced support for uncertain numeric action effects in an opportunistic manner (Coles, 2012). They aimed to balance the competing requirements of exploration (high resource usage, high rewards, risky) and exploitation (low resource usage, low rewards, safe) in the context of probabilistic numeric planning.

The approach of Coles was based on a novel combination of a pessimistic core plan with optimistic branches. The key technique was generating the pessimistic plan, and augmenting it with branches that account for better outcomes.

The interesting thing about uncertain numeric effects in the core plan is that a statistical guarantee can be given for a sequence of actions to execute successfully (given a certain percentile threshold). This was achieved by taking into account not only the value itself but also the potential Gaussian uncertainty around it. As we later detail, we also adopted a similar percentile-based approach to generating weak plans.

Coles proposed their approach in the context of over-subscription planning problems. This is a class of problems with several soft goals, where each soft goal carries a violation cost if not reached. In this setting, the task of planning is to find a plan that minimises the sum of violation costs for non-achieved goals. The core pessimistic plan introduced in Coles' work assumed that the full distribution of possible outcomes could happen and, if found, guaranteed success with a prescribed confidence level. Then, for each soft goal, a branch was added to the core plan with a corresponding probability for the case when the outcome was better (i.e. had a lower cost) than pessimistically expected. These branches were followed at execution time depending on the current values of uncertain resources. This approach obeyed strict safety constraints while dramatically increasing utility compared to previous approaches, due to the added branches.

2.7 Building Policies With Propositional Uncertainty

Also highly relevant to this thesis is prior work on contingent planning, or policy building – this involves finding a plan that handles all possible eventualities (also known as a policy). At execution time, the course of action varies based on sensed information about the current state of the world. Often this setting assumes that the world is fully observable - hence the acronym FOND, or fully observable non-deterministic planning.

A contingent planner based on FF was developed by Hoffmann and Brafman, who approached contingent problems as And-Or search problems in belief space (Hoffmann & Brafman, 2005). They based their work on the successful Conformant-FF planner mentioned earlier. The search space representation was still based on propositional formulas; the relaxed planning problem however was more difficult to adapt to contingent planning. The authors introduced an additional relaxation allowing them to map a relaxed contingent problem to a relaxed conformant problem, in order to leverage Conformant-FF.

Soon after, Albore *et al.* introduced a novel way of translating contingent planning problems into And-Or search problems, this time in search space (Albore, Palacios, & Geffner, 2009). Their idea contrasted with the work above, which treats contingent planning with sensing as a problem in belief space.

Fu *et al.* looked at improving strong cyclic planning for FOND problems (Fu, Ng, Bastani, & Yen, 2011). As mentioned earlier, a strong cyclic plan is an iterative plan that might terminate, and if it terminates it is guaranteed to succeed. Fu *et al.* pointed out the main reason why FOND problems with strong cyclic solutions are difficult to tackle - the unmanageable size of the search space. They first developed their own strong cyclic planning algorithm based on the prior planner NDP, and then introduced a novel heuristic that addresses the search space size issue.

Policy building was also introduced (together with failure analysis and plan repair) to continuous-time stochastic domains with concurrency by the planner Tempastic (Younes & Simmons, 2004). This planner developed policies incrementally by making use of a deterministic kernel, similarly to the approaches described in the following section.

2.7.1 The State Relevance Approach

Of particular interest to us, Muise *et al.* introduced a collection of novel approaches in the area of strong cyclic plans (or policies) for FOND problems. They successfully built policies orders of magnitude more compact than those of previous approaches (Muise, McIlraith, & Beck, 2012). They introduced the novel concept of partial states in the context of FOND problems, and used it to collapse similar states together based on fact relevance (Muise, 2014). They also introduced forbidden states as a way of extending the no-go area of the search space surrounding dead ends. We will build upon the concepts of both partial states and forbidden states in Chapter 4 of this thesis.

The core mechanic Muise *et al.* used for building policies with partial states was goal regression. Regression is a technique they had already used successfully in the context of partial order plan execution monitoring (Muise, McIlraith, & Beck, 2011). In that setting the policy allowed them to switch between one plan linearisation and another as needed during execution in an uncertain environment. We will extend regression from propositional domains to numeric domains later in this thesis.

Using policy building they also targeted FOND problems with conditional effects, which were not addressed in the work on non-deterministic planning to date. They proposed an extension to the current state of the art planner PRP (Planner for Relevant Policies), also developed by Muise *et al.*. This extension enabled PRP to support conditional effects (Muise, McIlraith, & Belle, 2014).

The authors also investigated partially observable environments – they noted how the predominant way to solve problems in partially observable domains was online. They set out to show how to do so offline in a scalable way (Muise, Belle, & McIlraith, 2014). In the context of planning under partial observability with sensing (PPOS), sensing actions are necessarily separate from regular actions. The authors chose to solve PPOS with offline planning, focusing on problems where the number of unobserved entities (true/false facts) decreases monotonically. They effectively handled partial observability by converting the problem to FOND planning. The main idea behind this conversion was replacing every sensing action with its two possible outcomes – thus, a previously unknown entity becomes either known to be true or known to be false.

Later, Camacho *et al.* introduced the ProbPRP planner, which computed solutions to probabilistic problems offline, also in the form of policies (Camacho, Muise, & McIlraith, 2016). The current state of the art did not scale to very large problem instances due to the complexity of planning for maximising the probability of reaching a goal. The authors made ingenious use of the similarities between probabilistic planning and FOND planning, and built upon the existing FOND planner PRP mentioned above. This allowed them to compute very compact policies that also guaranteed no avoidable dead ends would be reached.

In Chapter 4 we go further into detail on the techniques belonging to Muise *et al.* that we build upon.

2.8 Other Related Work

There are many other compelling approaches to planning with numeric uncertainty. For example, constraint programming (CP) was used to solve the job shop scheduling problem when durations are drawn from Gaussian distributions (Beck & Wilson, 2007). CP was also integrated with a probabilistic engine to accommodate uncertain demand in decision-making problems (Babaki, Guns, & Raedt, 2017).

Partitioning techniques based on POMDPs were used to solve problems with continuous state variables (Feng, Dearden, Meuleau, & Washington, 2004). POMDPs were also used to model the problem of maximising performance while bounding risk with a safety threshold (Santana, Thiébaux, & Williams, 2016). More recently, importance sampling was applied to an existing POMDP algorithm, improving performance critical but rare (and therefore difficult to sample) actions (Luo, Bai, Hsu, & Lee, 2019).

The FOND planner Gamer, which won its IPC track in 2008, approached planning in the form of a turn-taking game where one player is the solver and the other player is the environment (Kissmann & Edelkamp, 2009).

Finally, a compelling practical application of nondeterministic planning was described by Thiébaux *et al.*: power distribution in the presence of faults, where uncertainty over the sensors and actuators made the fault locations and network configuration only partially observable (Thiébaux & Cordier, 2001).

Chapter 3

Guiding Search Under Gaussian Uncertainty

This chapter contains the first major contribution of the thesis: informing the heuristic about numeric uncertainty. The contents of this chapter have been published in the Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (Marinescu & Coles, 2016a).

In what follows we explore heuristic guidance for forward-chaining planning with continuous random variables, while ensuring a probability of plan success. We extend the Metric Relaxed Planning Graph heuristic to capture a model of uncertainty, providing better guidance in terms of heuristic estimates and dead-end detection. By tracking the accumulated error on numeric values, our heuristic is able to check if preconditions in the planning graph are achievable with a sufficient degree of confidence; it is also able to consider acting to reduce the accumulated error. Results show that our approach offers significant improvements in several areas, compared to prior work where a less-informed relaxation was used. Overall, problems took less time and memory to solve; previously unsolved problems were solved; and unsolvable problems were proven so in less time.

3.1 Introduction

Our focus is on finding plans for domains with uncertainty over the numeric effects of actions, where each uncertain numeric effect is governed by a continuous distribution. Here, the task is to find a plan where all the preconditions are met, and the goals are reached, with some level of confidence θ. This paradigm has been explored by previous work (Beaudry et al., 2010b; Coles, 2012), but heuristic guidance is still an open challenge. A planning model without uncertainty cannot always provide reliable plans – similarly, a heuristic without a model of uncertainty cannot always provide useful guidance. A good heuristic would be better able to indicate which actions are suitable, and offer better state pruning by recognizing dead ends sooner.

We present an extension to the Metric Relaxed Planning Graph heuristic (Hoffmann, 2003) that incorporates a model of uncertainty for two purposes:

- First, basic information about uncertainty on variables thus far is used to determine which preconditions are true in the planning graph.
- Second, if remedial actions are available to reduce uncertainty, the heuristic is able to include these actions in the relaxed plan.

To demonstrate the efficacy of this new heuristic, we present empirical results that indicate it is effective in a number of interesting planning domains, reducing the search effort that is needed to find acceptable solution plans.

3.2 Background

Below we reiterate the core numeric planning formalism laid out in Chapter 2, with two important additions based on the non-deterministic state progression semantics of the RTU planner (Beaudry et al., 2010b). First, numeric effects are not necessarily constant values – they can be drawn out of specified probability distributions instead. Second, there is a given confidence threshold θ applied to action preconditions. Both of these additions are accompanied by examples in the bullet points below. A planning problem is therefore 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 a known initial state: a subset of facts from F and a value for each variable in $\mathbf{v}$.

- A *condition* is a conjunction of a subset of facts from F and constraints on $\mathbf{v}$. Each constraint is written as $(\mathbf{w}.\mathbf{v} \geq c)$, where c is a real value and $\mathbf{w}$ is a vector of real values.

- G is a set of goal conditions. For over-subscription planning problems, each $g \in G$ has an associated cost $c(g) \in \Re^{+}$ if g is not true at the end of the plan. For hard goals, $c(g) = \infty$.

- A is a set of actions, each $a \in A$ with:

  – $Pre(a)$: a set of conditions on the execution of a.

  – $Eff^{+}(a)$, $Eff^{-}(a)$: sets of facts from F added and deleted by a, respectively.

  – $Eff^{num}(a)$: a set of numeric variable updates that occur when applying a. Each is 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 outcomes of the 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 given at the start and applied to action preconditions. For instance: precondition $v \geq 15$ is considered true with $\theta = 0.9$ if the inequality holds 90% of the time. This is necessary due to the uncertainty in the effects on v.

According to Beaudry *et al.*, a Bayesian network (BN) is used to define the belief of each variable in $\mathbf{v}$. As actions are applied, the network is updated with additional variables. In a state S_i, for each $v^j \in \mathbf{v}$, a variable v_i^j is associated with the belief of v. If an action a is applied, leading to a state S_{i+1}, then for each numeric effect $\langle v^j \ op \ D(\mathbf{v})\rangle$, two random variables are added to the network. The first of these, D_{i+1}^j, represents $D(\mathbf{v})$. The second, v_{i+1}^j, is associated with the belief of v in S_{i+1}, and it is determined by either:

- $v_{i+1}^j = v_i^j + D_{i+1}^j$, if op is $+\!=$

- $v_{i+1}^j = D_{i+1}^j$, if op is $=$

The BN is key to determining whether a plan meets the required confidence level θ. An action a is applicable in a state S_i if $Pre(a)$ is satisfied. A sequential (linear) solution is a sequence of steps $[a_0, .., a_n]$, implying a state trajectory $[I, S_0, .., S_n]$. We use the BN to ensure that with probability $P \geq \theta$, in a given execution of the plan, each action's preconditions are met and S_n satisfies any hard goals.

The above state progression formalism by Beaudry *et al.* was adopted and extended by Coles as the basis of an over-subscription planning approach (Coles, 2012). A forward-chaining planner following these semantics was used to find a single plan, onto which branches were added afterwards by making additional calls to the planner.

As mentioned in Chapter 2, a range of other approaches have been adopted for planning under uncertainty, such as those based on the use of Markov Decision Processes (Meuleau et al., 2009; Mausam & Weld, 2008; Rachelson, Quesnel, Garcia, & Fabiani, 2008); these approaches are particularly useful when searching for a policy. As the contribution described in this chapter is on the heuristic inside a forward chaining planner, our focus will be on planning under the semantics of RTU described above. We will discuss policy building in further chapters.

3.3 Ensuring the Heuristic Remains a Relaxation

In deterministic forward-chaining numeric planning, one way to guide search is the Metric Relaxed Planning Graph (Metric RPG) heuristic (Hoffmann, 2003). This performs a forward reachability analysis that estimates the number of actions needed to reach goals by relaxing the effects of actions. For numeric state variables, this amounts to estimating reachable bounds on the values of variables. This is done by optimistically assuming that increase effects only increase the upper bound, and decrease effects only decrease the lower bound.

When working with RTU's semantics, Coles adapted the above to assume that, for heuristic purposes, each variable takes its median value. They point out that this is guaranteed to be a relaxation for $\theta \geq 0.5$, based on Jensen's inequality. In other words, adding more uncertainty would not contribute to achieving the goals.

As θ becomes large however, this median-based heuristic becomes increasingly unrealistic. A numeric condition might be true assuming variables take their median values; but not when accounting for the uncertainty in their values. To improve this, we incorporate the shape of the distribution in the heuristic evaluation, rather than discarding it and using the median. Additionally, for Gaussian distributions, we explicitly track variance as a variable in the relaxed planning graph.

From the point of view of the planner, uncertainty can be split into the following two categories:

- It only increases monotonically – error accumulates and no actions exist that can rectify it.
- It may be purposefully corrected – there may be actions that reduce the error (e.g. recharging batteries to a fixed value, or visiting a precise weighing station).

We discuss these, in turn, in the two subsections below.

3.3.1 Uncertainty Increases Monotonically

Each action's precondition (outside the heuristic) is of the form $\mathbf{w}.\mathbf{v} \geq c$. A Monte Carlo simulation is used to estimate the probability distribution of the left-hand side, $\mathbf{w}.\mathbf{v}$. Using this distribution, we can check if the condition is satisfied with probability θ. That is to say, we test whether the $(1-\theta)$'th percentile of $\mathbf{w}.\mathbf{v}$ is greater than or equal to c. We represent this percentile as follows:

$$p_{1-\theta}(\mathbf{w}.\mathbf{v}) = \mathit{median}(\mathbf{w}.\mathbf{v}) - \mathit{offset}_\theta(\mathbf{w}.\mathbf{v})$$

In effect, offset_θ is the margin of error that must be tolerated, for the precondition to be true with probability θ. We illustrate the intuition behind this margin in Figure 3.1. The condition itself can then be rewritten:

$$\mathit{median}(\mathbf{w}.\mathbf{v}) \geq c + \mathit{offset}_\theta(\mathbf{w}.\mathbf{v})$$

We define that, if uncertainty is monotonically increasing, it means that offset_θ can never decrease. In this case, it is still a relaxation to use the offset values when determining which preconditions are true in the heuristic. The

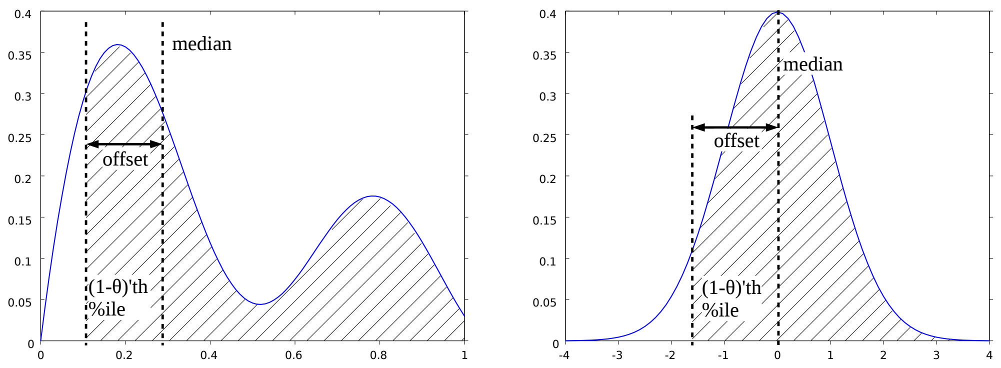


Figure 3.1: Possible probability distributions: Arbitrary (left) and Gaussian (right).

only way to make the condition above true is to apply actions that affect the values of $\mathbf{v}$, as no actions that decrease offset_θ exist.

An illustrative example would be an autonomous car with a certain amount of fuel, which is used gradually until it runs out; refuelling is not possible. The activities performed by the car (e.g. start engine, accelerate, stand still, park) each require fuel, but the amount varies non-deterministically. As the plan is constructed, uncertainty (and hence offset_θ) accumulates monotonically. We can thus heuristically evaluate a state by assuming offset_θ is constant, and takes its current value; this is guaranteed to be a relaxation, as offset_θ can never become smaller.

3.3.2 Uncertainty Can Decrease

So far, we explained how to incorporate distributions on the left-hand side of preconditions into heuristic computation, by using the offset_θ value to capture uncertainty information. The relaxation holds when error accumulates and cannot be lowered. However, problems may contain actions such as `recharge-batteries` or `visit-weigh-station`, which reduce uncertainty.

The challenge in these sorts of problems is to ensure the heuristic remains a relaxation. This is possible in a useful subset of domains, where the uncertainty is due to independent Gaussian-distributed effects on variables, and therefore has an analytic form. We can utilize this form and extend the Metric RPG to additionally track the variance on each variable, $\sigma^2(v)$. The expansion phase, building the RPG, proceeds as follows:

- For each variable $v \in \mathbf{v}$, we track the upper and lower bound on its median value. In the first RPG layer, these are equal to the value of v in the current state S. We additionally track $\sigma^2(v)$, the variance on v. In the first RPG layer, this is the value according to the BN for S.

- Normally when expanding an RPG, if a numeric effect is applied that increases (decreases) some $v \in \mathbf{v}$, the upper (lower) bound on v at the next fact layer is updated accordingly. Now, additionally, if a numeric effect decreases $\sigma^2(v)$, the lower bound on $\sigma^2(v)$ at the next fact layer is decreased[1].

- To decide which actions are applicable in each layer, we take variance into account when checking precondition satisfaction, as follows. For a precondition of the general form $\mathbf{w}.\mathbf{v} \geq c$, we can use the additive properties of Gaussians to compute the variance of $\mathbf{w}.\mathbf{v}$:

  $$\sigma^2(\mathbf{w}.\mathbf{v}) = \sum_{w.v \in \mathbf{w}.\mathbf{v}} w^2.\sigma^2(v)$$

  We obtain the offset using the Gaussian quantile function (the inverse of the Gaussian cumulative distribution function Φ for a given θ):

  $$\mathit{offset}_\theta(\mathbf{w}.\mathbf{v}) = \sigma(\mathbf{w}.\mathbf{v}).\Phi^{-1}(\theta)$$

  Hence, from Section 3.3.1, the precondition becomes:

  $$\mathit{median}(\mathbf{w}.\mathbf{v}) \geq c + \sigma(\mathbf{w}.\mathbf{v}).\Phi^{-1}(\theta)$$

This gives us everything we need to build an RPG. We can be confident that the offset_θ values used are relaxations: smaller values of variance result in smaller values of the Gaussian quantile function Φ^{-1}; and the semantics of the RPG guarantee we will underestimate variance.

The next step is to extract a relaxed plan from the RPG:

[1]Effects increasing $\sigma^2(v)$ are ignored when expanding the RPG. If $\theta \geq 0.5$, adding more uncertainty never contributes towards preconditions becoming true, so it suffices to track only the smallest reachable values of variance.

- For each condition $\mathbf{w}.\mathbf{v} \geq c + \mathit{offset}_\theta(\mathbf{w}.\mathbf{v})$ at the current layer, we need to compute $\mathit{offset}_\theta(\mathbf{w}.\mathbf{v})$ to later work out how to make the condition true. As stated above, we can assume that the probability distribution over the left-hand side is a Gaussian with mean $\mathbf{w}.\mathbf{v}$ and standard deviation $\sigma(\mathbf{w}.\mathbf{v})$. To deduce the offset, we use the Gaussian quantile function, Φ^{-1}. This function takes in the confidence θ imposed at the start, and the standard deviation $\sigma(\mathbf{w}.\mathbf{v})$. Our newly computed offset can now be used along with the constant c and the bounds on $\mathbf{w}.\mathbf{v}$ to represent the full condition numerically (in terms of bounds, just like in the original Metric RPG).

- Once the condition is fully represented as above, we begin to choose actions from action layer l that take the condition closer to being true. This is done in two steps – first, standard RPG-style effects on numeric fluents; and second, effects on variance:

  - In step one, as in the standard RPG, we choose actions that increase $\mathbf{w}.\mathbf{v}$, as well as actions that decrease c. We do this until either the condition is true, or all such actions in l have been used. If the condition is still not true, it must mean that a decrease in variance caused the precondition to become true (i.e. an action that reduced uncertainty, as we mentioned earlier); we proceed to step two.

  - In step two, we need to choose actions that decrease variance enough to achieve the precondition. To do this, we first work out by how much the offset needs to be reduced to make the precondition true. Afterwards we compute what variance this amount corresponds to for the given θ. We can now choose actions that decrease variance by the amount required to achieve the precondition.

We illustrate the solution extraction described above in Algorithm 1. The first thing to note is on lines 5 and 6, where we compute the offset_θ necessary for the condition to be met. Actions are then chosen in the standard way to attempt to meet the precondition, given this value of offset_θ. Then, if line 13 is reached and the precondition is still not true, it must mean that a decrease in variance caused it to become true at layer l (having been false at layer l-1). We now need to choose actions that decrease variance enough to achieve

Algorithm 1: RPG Solution Extraction

```
Data: $RPG$, a relaxed planning graph
Result: $p$, a relaxed plan
1  $last \leftarrow$ last layer index in $RPG$;
2  $goals[last] \leftarrow G$ (i.e. the problem goals);
3  for $l \in [last..0]$ do for $(\mathbf{w}.\mathbf{v} \geq c) \in goals[l]$ do
4      $prev \leftarrow$ max value of $\mathbf{w}.\mathbf{v}$ in fact layer $l$-1;
5      $prev\_\sigma^2 \leftarrow$ min value of $\sigma^2(\mathbf{w}.\mathbf{v})$ in fact layer $l$-1;
6      $prev\_offset_\theta \leftarrow prev\_\sigma.\Phi^{-1}(\theta)$;
7      if $prev \geq c + prev\_offset_\theta$ then
8          add $(\mathbf{w}.\mathbf{v} \geq c)$ to $goals[l$-1$]$; continue;
9      for $(w.v) \in \mathbf{w}.\mathbf{v}$ where $w \neq 0$ do
10         Choose actions from action layer $l$-1 that increase $(w.v)$;
11         Add them to the relaxed plan and subtract their effects from $c$;
12         if $prev \geq c + prev\_offset_\theta$ then break;
13     if $prev \geq c + prev\_offset_\theta$ then
14         add $(\mathbf{w}.\mathbf{v} \geq c)$ to $goals[l$-1$]$; continue;
15     $max\_offset \leftarrow prev - c$;
16     $max\_\sigma^2 \leftarrow (max\_offset/\Phi^{-1}(\theta))^2$;
17     add $(-\sigma^2(\mathbf{w}.\mathbf{v}) \geq -max\_\sigma^2)$ to $goals[l]$;
18     add $(\mathbf{w}.\mathbf{v} \geq prev)$ to $goals[l$-1$]$;
```

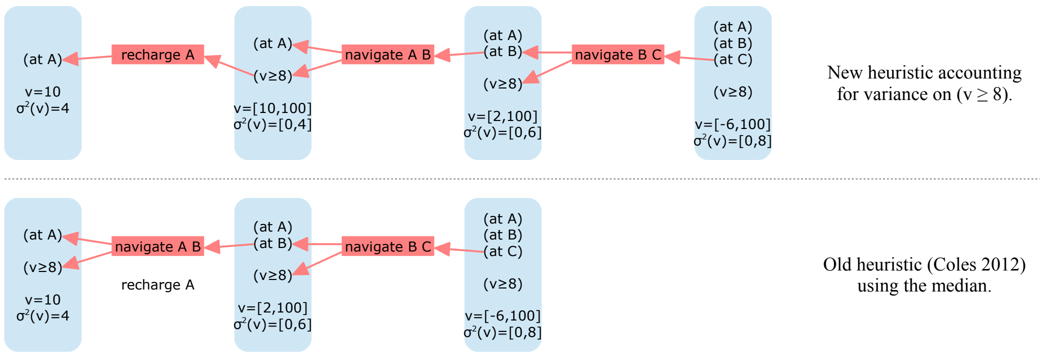


Figure 3.2: Example Relaxed Planning Graph, comparing the heuristic here to that in Coles (2012).

this. On line 15, we work out what $offset_\theta$ needs to be reduced to in order to make the precondition true; we then compute its corresponding variance on line 16. This variance can then be used to construct a new condition to be satisfied at this layer: this causes actions to be added to the relaxed plan in order to reduce variance on a later iteration of the loop.

As a result of Algorithm 1, the relaxed plan now contains actions that reduce uncertainty. This makes for a more informed heuristic, which is able to provide improved guidance and dead-end detection to the search, as will be demonstrated in the following section.

Figure 3.2 shows an example relaxed planning graph for a planetary rover domain, where navigation uses power v and increases variance on power $\sigma^2(v)$. In the top half of the figure, in the first layer, the only applicable action is recharge. Whilst there are 10 units of power, the condition $v \geq 8$ is not true until the next action layer, as $\sigma^2(v)$ is too high to allow it to be true with confidence θ. Conversely, in the bottom half of the figure (without our modifications), $v = 10$ would satisfy $v \geq 8$ since the value of $\sigma^2(v)$ is not taken into account when determining which numeric conditions are true. The consequence is that the new relaxed plan (top half) recognises the need to recharge, whilst the old one (bottom half) does not. If the action `recharge-A` were unavailable, then the new heuristic would detect a dead-end and the old one would not, as there would be no way of making $v \geq 8$ true for $\sigma^2(v)$.

3.4 Evaluation

The aim of our evaluation is to verify whether applying our novel heuristic improves planner performance when compared to the Metric RPG based heuristic used by Coles (2012) on three example domains with different manifestations of numeric uncertainty.

We implement our heuristic inside the Optic+ planner, and use the unmodified Optic+ planner as a baseline (as it uses the Metric RPG based heuristic). Our implementation [2] and the Optic+ implementation [3] are both available online.

We use two metrics to measure planner performance: 1) time elapsed and 2) number of nodes expanded in order to arrive at a solution plan. These metrics are measured on a set of 20 problems per domain, which cover a range of difficulty. With these metrics, our conditions for a successful evaluation are therefore: less time elapsed / fewer nodes generated / more problems solved when using our heuristic, compared to using the Metric RPG based heuristic used by Coles (2012).

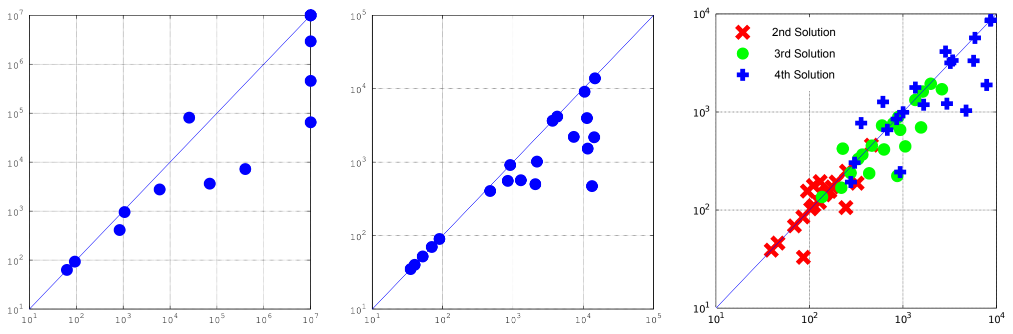
We chose evaluation domains in which numeric uncertainty is a realistic concern for the modelled problem, and which exhibit interesting differences from one another (in order to showcase the diverse set of problems that our heuristic can be applied to). These domains are Rovers (Coles, 2012), TPP (modified) (Gerevini et al., 2009), and AUV (Coles, 2012):

- In Rovers, the activities of a planetary rover are constrained by battery usage. Actions that use battery introduce Gaussian uncertainty over it, and the battery can be recharged to exactly 100% at certain locations.

- In TPP, the existing domain is modified to model the acquisition of sufficient weights of bulk materials. The action of acquiring materials introduces Gaussian uncertainty over the weight, and trucks can visit certain weighing stations called Khajiit to adjust their load and reduce uncertainty.

- AUV is an over-subscription domain where the activities of an underwater vehicle must be planned with a strict bound on total time taken. Activity durations are Gaussian distributed, and uncertainty over them cannot be reduced.

[2]https://github.com/chipsetgirl/rewrite/tree/plankton
[3]https://nms.kcl.ac.uk/planning/software/optic.html



(a) Rovers (b) TPP (c) AUV

Figure 3.3: Nodes generated to solve problems in the three evaluation domains. Axes are logarithmic, comparing prior work (X axis) with the new heuristic (Y axis). The two-tailed Wilcoxon signed-rank test confirms results are significant at $P \geq 0.95$.

To represent the uncertainty over any given numeric fluent in the above domains, we introduce an additional numeric fluent distinguished by a hard-coded suffix. For example, *v-variance* is a the fluent used to track the uncertainty over fluent v. A PDDL listing of the domains used in this evaluation (including all our modifications) can be found in Appendix A.

Tests were performed on 3.5GHz Core i5 machines with a limit of 4GB on memory and 1800s on CPU time.

Overall, the new heuristic leads to a substantial reduction in nodes generated to solve problems, as well as time taken to solve problems. Scatterplots for nodes generated are shown in Figure 3.3, and time taken scatterplots are virtually identical. The extra computational work (tracking variances etc.) does not adversely affect the time taken to heuristically evaluate a state. Thus, as significantly fewer states are generated, and per-state evaluation times are comparable, the performance of the planner is significantly better.

3.4.1 Overall Performance

Domain (a): For the Rovers domain (Figure 3.3a), most striking are the points on the far right-hand side of the graph – these indicate problems that were previously unsolvable but can now be solved. In part, this is because the new heuristic is able to recognize many more states as being dead ends, because it does not disregard uncertainty on the battery level when evaluating preconditions. In contrast, by ignoring uncertainty, the old relaxed plans relied on moving somewhere to recharge, even though in reality uncertainty made it impossible for that navigate action to be applied. The new heuristic often avoids this pitfall by accounting for uncertainty.

Domain (b): In TPP (Figure 3.3b), all the problems could be solved by both the old and the new heuristic. However, by not accounting for uncertainty, the old heuristic would reach states in which the relaxed plan does not need to buy more of any goods. In these states, the heuristic value is 0. As acquiring more goods requires combinations of travel and buy actions, a substantial amount of search must be performed with no effective heuristic guidance. Unlike Rovers, there are no dead ends due to these travel actions, so this blind search will succeed, but it is very time consuming – in problems furthest from the line $y = x$, the majority of nodes evaluated have an old heuristic value of 0.

Domain (c): AUV is an over-subscription problem: search reports a solution plan every time it finds one that solves more goals than the best so far. We are hence interested in the search effort to find progressively better solutions. Figure 3.3c compares the nodes generated by each configuration to find the 2nd, 3rd and 4th solutions. These correspond to satisfying 1, 2, and 3 goals respectively. The relaxed plans produced by the old heuristic, by ignoring uncertainty, more often use actions that there is actually no time to complete. Disregarding uncertainty is less of an impediment than in Rovers and TPP, as there is no scope for planning actions that reduce uncertainty – unlike battery charge or goods purchased, actions cannot create more time. Nonetheless, the new heuristic is generally able to find better solutions more quickly. If left to run for long enough, search with the old heuristic will tend to find solutions as good as search with the new heuristic, but loses out earlier in the search.

There are a small number of cases where our heuristic does worse than the old heuristic: when the former explores more states. This happens because the Metric RPG heuristic (and, in turn, ours) is neither admissible nor consistent. Changing the heuristic to account for uncertainty will generally improve the performance of the planner – but is not guaranteed to improve it. Pleasingly though, the planner with the new heuristic solves all the problems that were solvable with the old heuristic.

3.4.2 The Threshold θ

As a concluding remark for our results, we note that so far we assumed $\theta = 0.99$. At $\theta = 0.8$, the improvements from using the new heuristic are still noticeable, but not as substantial. By $\theta = 0.6$, which is close to the median ($\theta = 0.5$), there is no statistically significant difference between the two, as uncertainty has only a modest effect on the heuristic, or indeed search itself. This confirms that our heuristic meets our headline aim of being able to better guide the planner when the consequences of uncertainty have a significant effect upon what is a reasonable solution plan. In essence, the greater the impact of uncertainty on a problem, the greater the benefit of using our new heuristic.

3.5 Conclusion

In this chapter, we presented a novel search heuristic that extends the Metric Relaxed Planning Graph heuristic to include information about uncertainty, in a useful subset of problems:

- For cases where uncertainty is monotonically increasing, we showed how the offset_θ values (obtained from the Bayesian network) for a state can be incorporated into the heuristic evaluation for that state, to better reflect the margin of error that must be allowed for when checking if preconditions are true.

- For cases where uncertainty can be decreased, and its distribution is Gaussian, we showed how the variance on variables' values can be tracked explicitly in the RPG; and how RPG expansion and solution extraction can be updated to build relaxed plans that use uncertainty-reducing actions.

This chapter offered promising results which support our idea of including information about uncertainty in the heuristic. In the three domains we tested our approach on, we improved the performance of forward-chaining planning when the aim was to find a single plan that is overwhelmingly likely to succeed. While we did not aim for a theoretical proof that our approach is universally faster given certain problem conditions, our claim could be reasonably believed to hold on other domains similar to the ones presented in this evaluation.

Further in the thesis we will continue to use this type of uncertainty-aware planning kernel to support our contributions in the area of contingent planning, where actions have multiple outcomes.

Chapter 4

Building Policies Under Gaussian Uncertainty

In the previous chapter we discussed a setting where actions have a single outcome, and numeric effects are drawn from a probability distribution. Our work so far had the most impact on a useful subset of domains where this probability distribution is a Gaussian. However, in many non-deterministic domains, actions cannot reasonably be approximated to a single Gaussian outcome. On the quest for generality, allowing multiple Gaussian outcomes would be a considerable improvement. This is what we address now.

This chapter contains the second major thesis contribution: defining numeric regression in order to build policies for domains with multiple Gaussian outcomes. The contents of this chapter have been published in the Proceedings of the Twenty-Second European Conference on Artificial Intelligence (Marinescu & Coles, 2016b) (Marinescu & Coles, 2016c).

Within the ecosystem of domains with multiple action outcomes, there is a large body of work on policy building under propositional uncertainty. However, handling numeric uncertainty in such a setting has been given less consideration. In this chapter we present a novel offline policy building approach for problems with numeric uncertainty. In particular, inspired by the planner PRP (Muise et al., 2012), we define a representation that captures only relevant numeric information, supporting a more compact policy representation. We also show how PRP's dead end generalisation can be defined for numeric dead ends, to avoid redundant search. Empirical results show that we can substantially reduce the time taken to build a policy compared to prior work that takes a naive approach to numeric uncertainty.

4.1 Introduction

Unlike the class of problems considered by Chapter 3, there are many planning applications where a dynamic environment makes it difficult to plan under the assumption that actions have only a single outcome. For these problems, the task of planning may be to find a strong cyclic plan that will always reach the goal (Daniele et al., 1999), or a probabilistic plan that will succeed with a given probability (Yoon et al., 2007). These types of plans can be represented as a policy: a mapping from every reachable state to a prescribed action that should be taken in that state.

In problems with propositional uncertainty, where actions have multiple discrete outcomes, recent work on the planner PRP (Muise et al., 2012, 2014) builds a policy by making repeated calls to a deterministic planning kernel. This takes as input a determinised problem, where every non-deterministic action is replaced with a set of deterministic ones (Yoon et al., 2010), and produces as output a weak plan (i.e. a plan which assumes that we can choose which outcome occurs when it is applied). For this approach to scale, regression is used to keep only those parts of a state that are relevant, leading to a compact policy representation. Further, to work around dead ends (Little & Thiébaux, 2007), a dead end generalisation technique is used to efficiently prune unsuitable choices from the search space.

Having proven effective in propositional problems, this approach has great scope for being used in problems with numeric non-determinism. In this chapter, we develop a planner that does just this. We support a problem representation where each action has multiple outcomes that, in addition to prior work, can now have numeric effects. In addition, we allow these numeric effects themselves to be non-deterministic. Similarly to Chapter 3, effects can update state variables according to a continuous probability distribution. As a planning kernel, we use an adaptation (Coles, 2012) of RTU (Beaudry et al., 2010b) to find weak plans. This kernel uses Bayesian networks to manage the impact of the continuous probability distributions, finding a plan that will succeed with some confidence level. Around this, we present techniques for numeric regression and dead end generalisation. As in the propositional case, both of these techniques substantially reduce the amount of time spent searching for weak plans, and therefore allow the overall approach to scale.

We will first define the formalism we use, before describing our techniques for efficiently handling numeric uncertainty. We will then briefly discuss some issues that arise when trying to find strong cyclic plans in numeric domains, where it might be necessary to tolerate a small risk of failure. We then evaluate our techniques on benchmark domains and show consistent (and in some cases dramatic) reductions in the time taken to solve problems.

4.2 Background

Below we describe the particular elements of prior work upon which this chapter builds, namely:

- Planning where numeric effects introduce uncertainty according to some probability distribution (e.g. an effect might decrease battery by an amount with mean 10 and standard deviation 2).
- Building policies where actions can have several non-deterministic outcomes (e.g. an outcome might be stepping forward, and another might be failing to move).

4.2.1 Planning With Numeric Uncertainty

Similarly to the previous chapter, the planning formalism we use is based on RTU (Beaudry et al., 2010b). However, we make an important adaptation to it: actions can have multiple outcomes, to support the policy building mechanics we will detail in Section 4.2.2. A brief example of multiple outcomes appears below. In this case, a planning problem is 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 a known initial state: a subset of facts from F and a value for each variable in $\mathbf{v}$.
- A *condition* is a conjunction of a subset of facts from F and constraints on $\mathbf{v}$. Each constraint is written as $(\mathbf{w}.\mathbf{v} \geq c)$, where c is a real value and $\mathbf{w}$ is a vector of real values.
- G is a set of goal conditions.

- A is a set of actions, each $a \in A$ with:
  - *Pre*(a): a set of conditions on the execution of a.
  - *Eff*(a): a set of outcomes. For instance: the action `navigate` below has one outcome with duration 10, and another outcome with duration 15 that adds an additional fact. Each outcome $o \in \mathit{Eff}(a)$ is a tuple of the form $\langle \mathit{Eff}^{+}, \mathit{Eff}^{-}, \mathit{Eff}^{num} \rangle$, with:
    - $\mathit{Eff}^{+}(a)$, $\mathit{Eff}^{-}(a)$: sets of facts from F added and deleted by outcome o, respectively.
    - Eff^{num}: a set of numeric variable updates that occur when outcome o occurs. Each is 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.
- $\theta \in [0.5, 1)$ is a confidence level given at the start and applied to action preconditions.

The following is extracted from a version of the AUV domain in which the `navigate` action is non-deterministic:

```
(:action navigate-outcome-1
   :parameters (?a - auv ?w1 - waypoint ?w2 - waypoint)
   :precondition (and (can_traverse ?w1 ?w2)
                      (at ?a ?w1)
                      (>= (time-left) 10))
   :effect (and (not (at ?a ?w1))
                (at ?a ?w2)
                (decrease (time-left) 10)))

(:action navigate-outcome-2
   :parameters (?a - auv ?w1 - waypoint ?w2 - waypoint)
   :precondition (and (can_traverse ?w1 ?w2)
                      (at ?a ?w1)
                      (>= (time-left) 15))
   :effect (and (not (at ?a ?w1))
                (at ?a ?w2)
                (decrease (time-left) 15)
                (stuck-valve)))
```

Because there is uncertainty on numeric variables, as in the previous chapter, it is not possible to be absolutely certain that numeric conditions are satisfied. For this reason, RTU uses a Bayesian network (BN) to model uncertainty and to check that numeric conditions are satisfied with the prescribed confidence level θ. In the case where 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.

In this chapter we work with problems where the continuous numeric uncertainty is caused by independent Gaussian effects on variables, and therefore has an analytic form. In any state, we store both the mean value of each variable v and its variance $\sigma^2(v)$. Effects can change either (or both) of these; and they are taken into account when determining if preconditions are true.

For a precondition of the general form $\mathbf{w}.\mathbf{v} \geq c$, we can use the additive properties of Gaussians to compute the variance of $\mathbf{w}.\mathbf{v}$:

$$\sigma^2(\mathbf{w}.\mathbf{v}) = \sum_{w.v \in \mathbf{w}.\mathbf{v}} w^2.\sigma^2(v)$$

A precondition is then true with confidence θ *iff*:

$$\mathbf{w}.\mathbf{v} \geq c + \sigma(\mathbf{w}.\mathbf{v}).\Phi^{-1}(\theta)$$

Above, Φ^{-1} is the Gaussian quantile function – the inverse of the Gaussian cumulative distribution function Φ for a given θ. Effectively, this computes the offset we need to add onto the precondition so that, even accounting for uncertainty, it remains true with confidence θ.

A number of other techniques have been proposed for planning with uncertainty over resource usage, many of which are based on Markov Decision Processes (MDPs) (Meuleau et al., 2009; Mausam & Weld, 2008; Rachelson et al., 2008). Particularly relevant is the HAO* algorithm (Meuleau et al., 2009), which formulates problems as hybrid MDPs; i.e. with both discrete and continuous state variables. This approach is limited though to the case where the values of numeric variables are monotonically decreasing.

4.2.2 Building Policies With Propositional 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 (Yoon et al., 2007). This means that each action with preconditions $Pre(a)$ and effects $\mathit{Eff}(a)$ is replaced by several actions, one for each outcome $o \in \mathit{Eff}(a)$, whose preconditions are $Pre(a)$ and whose effects are just those corresponding to outcome o.

In the work of Muise *et al.* it is possible to build a policy incrementally by repeatedly invoking a deterministic planner to find weak plans. The core of this approach is set out in Algorithm 2. Key to the success of their approach is exploiting the concept of state relevance. By regressing backwards from the goal through a weak plan step-by-step, they determine which facts are relevant to plan success at each point.

Regression takes as input a partial state ps – here, a set of facts. 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 that 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.

Alternatively, if no weak plan can be found, a dead end has been reached. One caveat of the algorithm is that, in this case, the policy needs to be

Algorithm 2: Generating a Strong-Cyclic Plan in PRP (Muise *et al.*)

```
Data: A planning task, with initial state $I$ and goals $G$
Result: A policy $P$
1  $P \leftarrow \{\}$; $Open \leftarrow [I]$; $Seen \leftarrow \{\}$;
2  while $Open$ is not empty do
3      $S \leftarrow Open$.pop();
4      if $(S \in Seen) \vee (S \vDash G)$ then continue;
5      $Seen \leftarrow Seen \cup S$;
6      if $\exists \langle ps, a \rangle \in P$ such that $S \vDash ps$ then
7          for $S' \in apply\_outcomes(S, a)$ do
8              $Open$.push($S'$);
9      else
10         $(weak\_plan, G') \leftarrow$ run planning kernel from $S$;
11         if planning kernel could not solve problem then
12             $ps\_dead \leftarrow generalise\_dead\_end(S)$;
13             generate forbidden state–action pairs from $ps_{dead}$;
14             $P \leftarrow \{\}$; $Open \leftarrow [I]$; $Seen \leftarrow \{\}$;
15         else
16             $PS \leftarrow$ regress $G'$ through $weak\_plan$ to generate
               partial-state–action pairs;
17             $P \leftarrow P \cup PS$;
18             for $S' \in apply\_outcomes(S, weak\_plan_0)$ do
19                 $Open$.push($S'$);
20 return $P$
```

deleted and the policy building process must be restarted (line 14). We refer to this as resetting the policy. When a dead end state S is reached, Muise *et al.* record ⟨forbidden state, action⟩ pairs (FSAPs). First, at line 12, S is generalised to yield a partial state (ps_dead): to avoid resetting the policy too frequently, a greedy algorithm is used to remove facts from S that do not affect whether it is a dead end[1]. At line 13, ps_dead is regressed through all actions $[a_0..a_n]$ that could lead to it, yielding partial states $[ps_0..ps_n]$; then each $\langle ps_i, a_i \rangle$ is recorded as forbidden, since applying a_i in ps_i would reach ps_dead once again.

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.

4.3 Numeric Regression

As detailed in Section 4.2.2, Muise *et al.* employ regression for states containing propositional facts. Based on this, they are able to build policies for propositional problems efficiently. Thus, ideally, we would like to apply their framework to the problem class laid out in Section 4.2.1, which includes numeric variables and allows numeric uncertainty. For this to work well though, we need to define regression for this numeric formalism.

4.3.1 Motivating Relevant Numeric Constraints

In the approach described by Muise *et al.* depicted in Algorithm 2, they make calls to a planning kernel to find weak plans from states to the goal. At line 16, they use regression to find partial states, each of which is paired with the corresponding action and added to the policy. This allows their approach to perform much better than if regression was not used, and naively the states in the weak plan from S to the goal were used instead.

Their approach did not, however, consider numeric planning problems. As an initial attempt at supporting these, we could suggest that partial states store the full numeric information from the original states in the weak plan from S. For instance, for the plan depicted in Figure 4.1, these would

[1]We will return to the topic of dead end generalisation in section 4.4

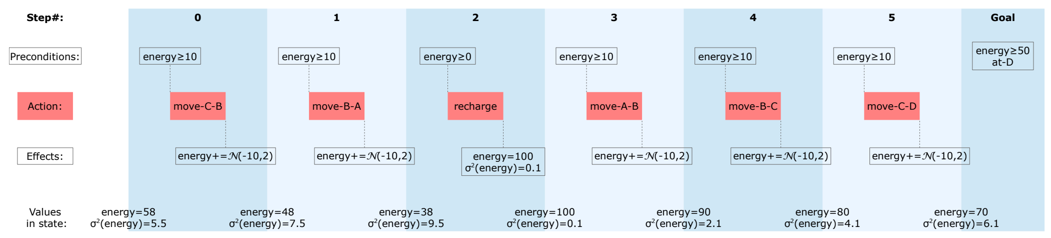


Figure 4.1: Example weak plan. For clarity, propositional preconditions and effects are omitted.

correspond to the values of *energy* and $\sigma^2(\mathit{energy})$ shown at the bottom of the figure. This is likely to perform badly, though; similarly to the naive propositional case, partial states would be so precisely defined that, at line 6, the state we are considering likely matches no partial states in the policy. We thus need to find a more effective way of capturing numeric information in partial states, accounting for the preconditions and effects of the actions in the weak plan.

In the propositional case, at any point, regression maintains a set of literals that need to be true – initially, those in the goal state. Regressing through the effects of actions removes literals from this set; and regressing through their preconditions adds literals to it. Thus, the cumulative effects of the actions that have been regressed through are implicit, and each partial state is defined as a set of literals that need to be true: analogous to the preconditions $\mathit{Pre}(a)$ on some action a.

In the numeric case without uncertainty, the representation needed for the numeric conditions in partial states is also equivalent to that used in the preconditions of actions. If we had a numeric condition $v \geq 7$, and regressed this through a numeric effect $v \mathrel{+}= -3$, then the resulting partial state would contain the numeric condition $v \geq 10$. Numeric effects accumulate as regression proceeds, but the representation remains the same.

In the presence of uncertainty, though, this is no longer the case. For instance, consider the situation depicted in Figure 4.1. This has a numeric goal that $(\mathit{energy} \geq 50)$. Preceding the goal is step 5 of the weak plan, with the effect $\mathit{energy}{+}{=}\mathcal{N}(-10, 2)$. It is not sufficient to stipulate that $(\mathit{energy} \geq 60)$ in the partial state prior to step 5, as this disregards the uncertainty on the value of *energy* introduced by the subsequent step. In fact, to ensure the numeric goal is met with confidence θ, *energy* will need to be somewhat larger than 60 before weak plan step 5. Continuing further back through the weak plan, prior to step 4, energy will need to be larger and account for further variance still, as step 4 also has the effect *energy* $\mathrel{+}= \mathcal{N}(-10, 2)$.

Thus, we will proceed to define a representation that captures:

- The condition that needs to hold – in general, $\mathbf{w}.\mathbf{v} \geq c$.
- The uncertainty on the condition's variables (due to any uncertain effects between the partial state and the step in the plan where the condition appeared).

4.3.2 Defining Regression Through Relevant Numeric Constraints

As noted in Section 4.2.1, we are assuming in this work that the numeric uncertainty is due to independent Gaussian effects on variables. The chain of effects $v \mathrel{+}= \mathcal{N}(\mu_0, \sigma_0^2), .., v \mathrel{+}= \mathcal{N}(\mu_n, \sigma_n^2)$ can be rewritten as:

$$v \mathrel{+}= \mathcal{N}(\sum_{i \in \{0..n\}} \mu_i, \sum_{i \in \{0..n\}} \sigma_i^2)$$

Or, alternatively, using a Gaussian with mean 0, as:

$$v \mathrel{+}= \sum_{i \in \{0..n\}} \mu_i + \mathcal{N}(0, \sum_{i \in \{0..n\}} \sigma_i^2)$$

This can be exploited when regressing through numeric effects that increase or decrease (rather than assign) a variable. Effects of this form accumulate, and can be expressed concisely as a single mean and standard deviation (or variance) value.

Effects that assign a variable though, do not accumulate so neatly. With reference to Figure 4.1, weak plan step 2 assigns a fixed value to *energy* and to $\sigma^2(\mathit{energy})$. If we regress through this effect, and then through the effects of weak plan step 1 (which include a decrease), the effects do not accumulate. Any effects earlier in the weak plan (i.e. preceding the assignment in step 2) do not affect the variable's value after that point in the weak plan.

Our representation is thus based on the following intuition. Suppose we have a numeric condition $\mathbf{w}.\mathbf{v} \geq c$, and are assuming there is no variance on $\mathbf{w}.\mathbf{v}$ to account for. Regressing this condition through an effect of an action does one of three things:

- The effect was an increase/decrease on some variable v, or on some variable's variance $\sigma^2(v)$. In this case we update the bound c on the condition according to the mean value of the effect, or record any additional variance that now needs to be accounted for, respectively.

- The effect was an assignment to a variable v. In this case the bound c is updated to account for the value that v was assigned, and v is removed from $\mathbf{w}.\mathbf{v}$. As noted previously, earlier effects would no longer affect the value of v after this point.

- The effect was an assignment to $\sigma^2(v)$ for some variable v. In this case we freeze $\sigma^2(v)$ to this value. Again, as noted previously, earlier effects on $\sigma^2(v)$ would no longer affect the value of $\sigma^2(v)$ after this point.

Formally, we define a Numeric Constraint Tuple (NCT) that includes an explicit record of any variance due to a chain of numeric effects. For a constraint $\mathbf{w}.\mathbf{v} \geq c$ we record an NCT $\langle nt, c, vt, av\rangle$ where:

- nt is a set of numeric terms, initially: $\{\langle w, v\rangle \mid w, v \in \mathbf{w}.\mathbf{v}\}$. Note we allow two sets of numeric terms to be added together by adding the respective weights on each variable.

- c is the right-hand-side constant, initially c.

- vt is a set of variance terms, initially: $\{\langle w^2, \sigma^2(v), rv\rangle \mid w, v \in \mathbf{w}.\mathbf{v}\}$ where rv denotes the regressed variance on the respective variable; initially, $rv = 0$.

- av is an accumulated variance value, initially 0.

Each NCT can be seen as a compact encoding of a numeric constraint and a chain of effects that precedes it in the weak plan. To determine whether a state S satisfies a given NCT, we refer to the values $S[v]$ and $S[\sigma^2(v)]$ for the mean and variance on v; and to the Gaussian quantile function (Section 4.2.1). We calculate the mean and variance of the NCT in S as:

$$\mu = \sum_{\langle w,v\rangle \in nt} w.S[v] \tag{4.1}$$

$$\sigma^2 = av + \sum_{\langle w^2,\sigma^2(v),rv\rangle \in vt} w^2.(S[\sigma^2(v)] + rv) \tag{4.2}$$

Then, we say S satisfies the NCT *iff*:

$$\mu \geq c + \sigma.\Phi^{-1}(\theta)$$

In effect, for each variance term in vt we account for the variance rv associated with it, and the variance in the state itself (each value $S[\sigma^2(v)]$), to determine whether some condition later in the plan will hold with an adequate degree of confidence.

If an NCT C is regressed through a numeric increase or decrease effect ($v \mathrel{+}= \mathbf{w}'.\mathbf{v}' + k$), where $\langle w, v\rangle \in nt$, and $k \in \Re$, the resulting NCT C' is:

$$\begin{aligned} nt' = &\quad nt + \{\langle w.w', v'\rangle \mid w'.v' \in \mathbf{w}'.\mathbf{v}'\} \\ c' = &\quad c - w.k \\ vt' = &\quad vt \\ av' = &\quad av \end{aligned}$$

This, intuitively, is what we would expect. Take, for instance, the simple condition $v \geq c$, regressed through the effect $v \mathrel{+}= -10$. The resulting value of c' is $c + 10$: prior to the effect, v needs to be 10 larger to ensure the condition remains true after it.

Regressing through an assignment effect ($v = \mathbf{w}'.\mathbf{v}' + k$) gives:

$$\begin{aligned} nt' = &\quad \{(\langle w', v'\rangle \in nt) \wedge (v' \neq v)\} + \{\langle w.w', v'\rangle \mid w'.v' \in \mathbf{w}'.\mathbf{v}'\} \\ c' = &\quad c - w.k \\ vt' = &\quad vt \\ av' = &\quad av \end{aligned}$$

Here, note that nt' is updated to remove the term $\langle w, v\rangle$. For instance, if we regress the condition $v \geq c$ through the effect $v = 100$, we remove the term associated with v from nt' and subtract 100 from the bound to give $c' = c - 100$.

Effects on variance update the variance terms vt. For a variance term $\langle w^2, \sigma^2(v), rv\rangle \in vt$, regressing through the effect $\sigma^2(v) \mathrel{+}= k$ increases rv by k. Simply, we now have to account for more variance on v than we did previously. Regressing through the effect $\sigma^2(v)$=k is somewhat more involved – it does two things:

- The term $\langle w^2, \sigma^2(v), rv\rangle$ is removed from vt. As noted earlier, when regressing, we stop accumulating effects on a variable once assignment has taken place: the same holds for the variance of a variable.

- The value $k.w^2$ is added to the accumulated variance av.

In effect, in Equation 4.2, we transfer the resulting value of the term referring to v from the set vt into the value av.

To demonstrate how numeric regression works in practice, we will now refer to Figure 4.1. This is an example weak plan for a Mars rover domain where the goal is to be at location D with 50 or more units of energy. The weak plan contains only two kinds of actions: `move-X-Y` and `recharge`. The former has a numeric effect $\mathit{energy}{+}{=}\mathcal{N}(-10, 2)$. The latter has two numeric effects: $\mathit{energy} = 100$; $\sigma^2(\mathit{energy}) = 0.1$. Regression proceeds as follows.

G: We begin with the constraints at the goal:
$\langle\{\langle 1, \mathit{energy}\rangle\}, 50, \{\langle 1, \sigma^2(\mathit{energy}), 0\rangle\}, 0\rangle,\ \mathit{at\text{-}D}$

5: We regress through the effects of `move-C-D`:
$\langle\{\langle 1, \mathit{energy}\rangle\}, 60, \{\langle 1, \sigma^2(\mathit{energy}), 2\rangle\}, 0\rangle,\ \mathit{at\text{-}C}$
Note the constraint above subsumes the precondition of `move-C-D`.

4: We regress through the effects of `move-B-C`:
$\langle\{\langle 1, \mathit{energy}\rangle\}, 70, \{\langle 1, \sigma^2(\mathit{energy}), 4\rangle\}, 0\rangle,\ \mathit{at\text{-}B}$

3: We regress through the effects of `move-A-B`:
$\langle\{\langle 1, \mathit{energy}\rangle\}, 80, \{\langle 1, \sigma^2(\mathit{energy}), 6\rangle\}, 0\rangle,\ \mathit{at\text{-}A}$

2: We regress through the effects of `recharge`:
$\langle\{\}, -20, \{\}, 0.1\rangle$
The constraint above is tautologous so we discard it.
We do however keep the precondition of `recharge`:
$\langle\{\langle 1, \mathit{energy}\rangle\}, 0, \{\langle 1, \sigma^2(\mathit{energy}), 0\rangle\}, 0\rangle$

1: We regress through the effects of `move-B-A`:
$\langle\{\langle 1, \mathit{energy}\rangle\}, 10, \{\langle 1, \sigma^2(\mathit{energy}), 2\rangle\}, 0\rangle,\ \mathit{at\text{-}B}$

0: We regress through the effects of `move-C-B`:
$\langle\{\langle 1, \mathit{energy}\rangle\}, 20, \{\langle 1, \sigma^2(\mathit{energy}), 4\rangle\}, 0\rangle,\ \mathit{at\text{-}C}$

4.4 Numeric Dead End Generalisation

In this section we consider dead end generalisation, another important element of Algorithm 2. We first recap how this is performed in propositional domains, then discuss how we develop an analogue of this for numeric variables.

4.4.1 Generalising Propositional Dead Ends

As discussed in Section 4.2.2, when building a policy, ⟨forbidden state, action⟩ pairs are generated each time a dead end is found. The dead end is regressed through all the actions that could lead to it, and in subsequent calls to the planning kernel these actions cannot be applied in states that would lead to that dead end.

In the propositional case, the dead end state $S = \{$`robot-at-A`, `truck-at-B`, $\neg$`package-in-truck`, `quicksand-at-A`$\}$ might be found. Prior to generating ⟨forbidden state, action⟩ pairs, S is generalised. For each literal in S in turn, the Relaxed Planning Graph (RPG) heuristic (Hoffmann & Nebel, 2001) is evaluated with the relaxation that both the literal and its negation are true. For instance, if the state $S \cup \{$`package-in-truck`$\}$ is still a dead end, we remove the literal $\neg$`package-in-truck` from S. After generalisation is complete, only a set of relevant literals remain – in this case $\{$`robot-at-A`, `quicksand-at-A`$\}$. This is quite intuitive: no matter where the truck and package are, the robot is still sinking in quicksand.

From then on, it is impossible to reach a dead end that contains a superset of these literals, as any action that would lead to such a dead end would be forbidden in the preceding state. For instance, a dead end $S' = \{$`robot-at-A`, `truck-at-E`, `quicksand-at-A`$\}$ would not be reached: the ⟨forbidden state, action⟩ pairs (Algorithm 2 line 13) generated from S would have prevented search from ever reaching S'. This reduces the number of dead ends reached by search, and hence the number of times the policy needs to be reset.

4.4.2 Motivating Variable Bound Expansion

Removing extraneous literals works well in the propositional case. However, in numeric problems, most of the gains are lost: the state S assigns specific values to each numeric variable, and these persist after propositional dead end generalisation.

Extending our previous example, S might be $\{$`robot-at-A`, `truck-at-B`, $\neg$`package-in-truck`, `quicksand-at-A`, `rescue-team-fuel`$=1.5\}$.

After propositional dead end generalisation, S would become $\{$`robot-at-A`, `quicksand-at-A`, `rescue-team-fuel`$=1.5\}$.

This does not prevent the planning kernel from later reaching another dead end state S' containing {`robot-at-A`, `truck-at-D`, `package-in-truck`, `quicksand-at-A`, `rescue-team-fuel`=1.1}, as the amount of fuel is different. It is intuitively clear, however, that having less fuel for the rescue team is worse, so S' a dead end.

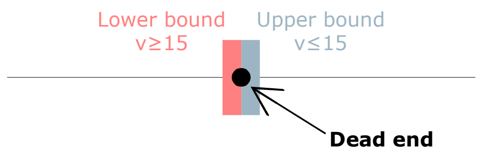
To address this, we will show how the RPG heuristic can also be used to relax the values assigned to numeric variables, broadening their range whilst ensuring the state is still a dead end. This then allows each dead end to match a greater number of states.

4.4.3 Expanding Bounds on Dead End Variables

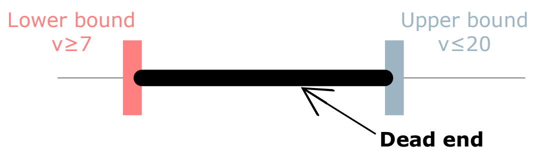
Algorithm 3: Lower-bounding v while allowing S to still be a dead end

```
   Data: A state $S$, a variable $v$
   Result: A modified state $S$, with relaxed lower bound on $v$
1  $Sv \leftarrow S[v]$;
2  Relax $S[v]$ to be in range $[-\infty, Sv]$;
3  if $RPG(S)$ is a dead end then return $S$;
4  $a \leftarrow glb(v)$; // global lower bound on $v$, typically $-\infty$ or 0;
5  $b \leftarrow Sv$;
6  while $(b - a) > \epsilon$ do
7      $mid \leftarrow (a + b)/2$;
8      Relax $S[v]$ to be in range $[mid, Sv]$;
9      if $RPG(S)$ is a dead end then $b \leftarrow mid$;
10     else $a \leftarrow mid$;
11 Relax $S[v]$ to be in range $[b, Sv]$;
12 return $S$;
```

In the Metric RPG heuristic (Hoffmann, 2003), the values of numeric variables are relaxed, lying in a range rather than taking on fixed values. The upper and lower bounds on each variable are then used to optimistically determine whether each precondition is true. Ordinarily, in the first fact layer, the upper and lower bounds on each variable are both set to the value the variable takes in the state being evaluated. But, conceptually, there is



(a) Before Generalisation



(b) After Generalisation

Figure 4.2: Generalising the value $v = 15$ in a dead end state S.

nothing preventing the RPG from being used to evaluate states in which variables' values can lie in a given range.

We exploit this observation in Algorithm 3, which illustrates how, for a given dead end state S and variable v, we can reduce the lower bound on v in a way that ensures S is still a dead end. An analogous algorithm can be used to find an upper bound on v.

To reduce the lower bound, we perform interval bisection, finding the smallest suitable value of v such that $RPG(S)$ is still a dead end[2]. This process is repeated in turn for each variable in the dead end state S. If the bounds on v become $[-\infty, +\infty]$, then we do not need to constrain the value of v at all. Otherwise, v lies in the range $[a, b]$ in the dead end, and transforming this into the NCT representation yields a pair of constraints:

$$\langle\{\langle 1, v\rangle\}, a, \{\}, 0\rangle, \langle\{\langle -1, v\rangle\}, -b, \{\}, 0\rangle$$

An example of setting bounds on a numeric variable is shown in Figure 4.2. Initially (at the top), $S[v] = 15$, so the range of values is taken to be $[15, 15]$. Algorithm 3 is then used to reduce the lower bound on $S[v]$ whilst ensuring S is still a dead end, converging on the bounds $S[v] \in [7, 15]$. The upper bound is then analogously increased, from 15 to 20. The conclusion (at

[2]As an implementation detail, we assume $-\infty$ is a very large, but finite negative number.

the bottom) is that if $S[v] \in [7, 20]$, then S is still a dead end; the resulting numeric constraint tuples would then be:

$$\langle \{\langle 1, v \rangle\}, 7, \{\}, 0 \rangle, \langle \{\langle -1, v \rangle\}, -20, \{\}, 0 \rangle$$

One additional note regarding the heuristic – as we are dealing with a domain where numeric variables might be drawn from Gaussian distributions, we use the improved Metric RPG heuristic from Chapter 3. It is a pleasing development that our work so far can serve the purpose of providing better dead end bounds in a different setting than it was originally designed for.

4.5 Tolerating a Risk of Failure

In this section we discuss issues that arise when attempting to find strong cyclic plans in domains with adverse numeric effects that are potentially fatal if they occur repeatedly. We discuss a modification of the policy building approach that finds plans with a tolerably small risk of failure.

4.5.1 Finding Strong Cyclic Plans in Numeric Domains

One notable property of the policies produced by Muise *et al.* is that they are guaranteed to produce strong cyclic plans. This means that, no matter what sequence of action outcomes occurs at execution time, using the policy will still lead to a goal, while allowing for states to be re-visited.

However, this property comes with a certain limitation, illustrated by the following example. A domain where a vacuuming robot needs to operate on slippery floors might have a `move` action which requires 10 *battery* and has three non-deterministic outcomes:

o_1**:** use 10 `battery`; do not slip; move forward;

o_2**:** use 10 `battery`; slip slightly; move forward-left;

o_3**: use 3 `battery`; slip critically; fail to move altogether.**

The existence of the third outcome makes it impossible to find a strong cyclic plan in such a domain. This outcome can occur over and over, using all the robot's battery without moving, hence rendering the goal location unreachable.

Because no strong cyclic plan exists, the technique by Muise *et al.* cannot produce a policy for domains such as the one above. There is a common denominator for this type of domains: actions have failure outcomes that cannot be undone or remediated for free. For example, if a bipedal robot falls down it is possible to apply an action to get up, but that action has a cost, hence cannot be applied indefinitely. Algorithm 2 would benefit from an adaptation allowing it to solve such problems, as they are not uncommon amongst various potential applications of planning.

We will refer to the example above to motivate our approach. The three different action outcomes might have the following likelihoods of occurring: $p(o_1) = 0.7$, $p(o_2) = 0.25$, $p(o_3) = 0.05$. If the goal is one step away, the robot starts out with 18 `battery` and o_3 occurs three times, there is no longer enough battery to meet the precondition of `move` and apply it again. The likelihood of this happening though, is very small: $0.05 \times 0.05 \times 0.05 = 0.000125$, or 0.0125%. This risk, however, is unavoidable; the only way to reach the goals is to try to move. Thus, we adapt the planner to allow it to track the cumulative risk of such failures.

4.5.2 Accumulating a Risk of Failure

We describe an approach where we tolerate reaching dead ends as long as the accumulated likelihood of reaching a dead end is less than an acceptable threshold τ. First, we add a variable *p_fail* to record the probability of failure; initially $\mathit{p\_fail} = 0$. We then modify the open list used in Algorithm 2 so that each state S is paired with the probability of reaching that state $p(S)$. When resetting the policy, the open list contains just one entry: $\langle I, 1.0\rangle$. Line 3 is rewritten:

$$\langle S, p(S)\rangle \leftarrow \mathit{Open}.\text{pop}();$$

We then check if $S \in \mathit{Seen}$, and if so, whether S was previously found to be a dead end. If it was, *p_fail* is incremented by $p(S)$; and if $\mathit{p\_fail} > \tau$, we reset the policy – the acceptable risk threshold has been exceeded.

The function $\mathit{apply\_outcomes}(\langle S, p(S)\rangle, a)$ is updated so that it returns a set of ⟨state, probability⟩ pairs $\langle S', p(S')\rangle$ – one for each outcome o of the action a, where:

$$p(S') = p(S) \times p(o)$$

These probabilities are then pushed, along with their corresponding states, onto the open list (lines 8 and 19).

If a dead end S is reached (line 11), p_fail is increased by $p(S)$. Then, at line 14, we only reset the policy (and set $p\_fail = 0$) if $p\_fail > \tau$. Thus, policy building will continue as long as the cumulative likelihood of failure is sufficiently small. Ultimately, the result of the algorithm is still a policy, but it no longer corresponds to a strong cyclic plan, as it might fail with probability τ.

4.6 Evaluation

We begin with an overall evaluation of the two principal contributions we made for finding strong cyclic plans in domains with numeric uncertainty: the use of regression to define relevant numeric constraints; and numeric dead end generalisation.

As a baseline, we take a configuration of our planner with neither of these, using only techniques from prior work:

- As alluded to in Section 4.3.1, regression is used for propositional preconditions and effects; but partial states contain the values of numeric variables from the original states in the weak plan. That is, if $v = k$ in some state S, the constraints $(v \geq k) \wedge (-v \geq -k)$ are added to the corresponding partial state ps.
- As described in Section 4.4.1, dead end generalisation is used for literals but not for numbers. Thus, again, if $v = k$ in a dead end state, the dead end partial state prescribes $(v \geq k) \wedge (-v \geq -k)$.

Our planner can be found online via the link provided in Chapter 3. Once again we used Optic+ as a starting point, first implementing Algorithm 2 from PRP inside Optic+, then adding in our numeric regression and dead end generalisation contributions.

We use three evaluation domains, derived from existing benchmarks - Rovers (Coles, 2012), a 'flat tyre' variant of TPP (Gerevini et al., 2009), and AUV (Coles, 2012):

- In Rovers, the activities of a planetary rover are constrained by battery usage, and the battery can only be recharged at certain locations. All energy usage is Gaussian. Additionally, navigation has three outcomes (each of which uses a Gaussian amount of energy).

- In Flat Tyre TPP, the domain is modified to model the acquisition of sufficient amounts of bulk materials (e.g. coal), allowing for Gaussian uncertainty in the amounts purchased. Additionally, drive actions risk a flat tyre that can be repaired indefinitely.

- In AUV, the activities of an underwater vehicle must be planned with a strict bound on total time taken, and with Gaussian activity durations. Some actions have outcomes that require an additional step to complete the activity.

A full listing of the three domains used in this evaluation can be found in Appendix B.

All tests were performed on 3.5GHz Core i5 machines with a limit of 4GB of memory and 1000s of CPU time. The confidence level for numeric conditions was set as $\theta = 0.9$. A range of other values were also considered; but the conclusions of this evaluation were the same for all cases.

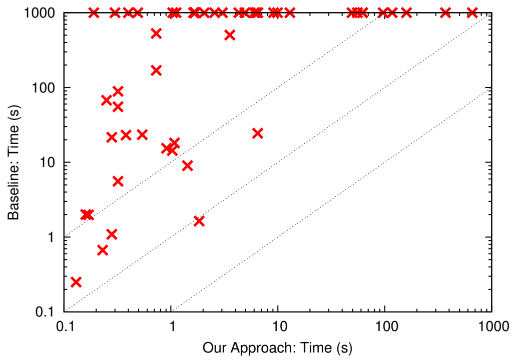
4.6.1 Overall Performance

Summary scatterplots of the results of these experiments are shown in Figure 4.3, comparing the time taken by our approach versus the baseline when solving the evaluation problems in these domains.

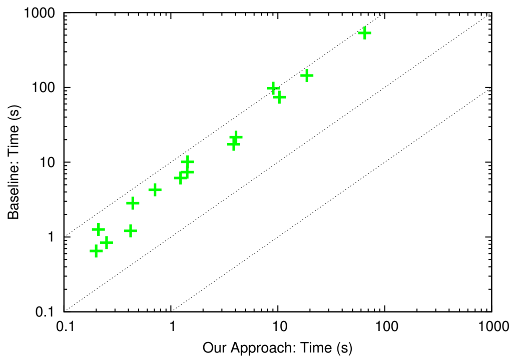
Domain (a): The results in Rovers are the most striking. Our approach is over an order of magnitude faster in most problems, and many previously unsolved problems (shown at $y = 1000$) are now solved. Looking at the runtime behaviour, the baseline planner makes many more calls to the planning kernel to find weak plans. This can be explained with reference to the battery charge (energy) levels stored in states:

- In the baseline version, the bounds on energy in partial states are taken from the states in the weak plan. If $\mathit{energy} = 40$ in some state S, this applies to the corresponding partial state ps too. Thus, in any other state S', if $\mathit{energy} \neq 40$, then $S' \not\models ps$ (even if S' has more energy, which is intuitively preferable).

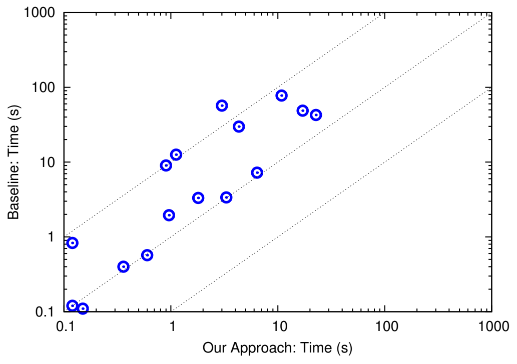
- With our approach to numeric regression, the partial state ps would instead contain a Numeric Constraint Tuple $\langle\{\langle 1, \mathit{energy}\rangle\}$, c, $\{\langle 1, \sigma^2(\mathit{energy}), rv\rangle\}$, $av\rangle$ that records what minimum amount of energy c is needed, and what variance needs to be accounted for due to



(a) Rovers



(b) Flat Tyre TPP



(c) AUV

Figure 4.3: Time taken to solve problems in the three evaluation domains, comparing our approach (X axis) to a baseline in which only propositional regression and propositional dead end generalisation are used (Y axis). Axes are log scaled. Tests that timed out are plotted at $t = 1000$.

later plan steps. Another otherwise identical state S' could then match *ps* if it had more energy than S; or less energy, but sufficient according to this NCT.

Domain (b): In Flat Tyre TPP, the improvements observed are solely due to numeric regression. Because tyres can always be repaired as many times as needed, and because more goods can always be purchased, the goals can always be reached. Nonetheless, this all needs to be planned for, and numeric regression again pays off in reducing the time spent searching for weak plans.

Domain (c): In AUV, the results are more intriguing than the scatterplot suggests. Clearly our techniques are beneficial overall, which is a pleasing result, but the improvement is less striking than in the other two domains. It turns out that many of the dead ends found contain no literals – they are typically the form:

$$(\textit{time-remaining} \leq k) \wedge (\textit{unsatisfied-goals} \geq l)$$

This is true both in the baseline and in our approach: propositional dead end generalisation is able to eliminate all the literals from most of the dead ends found. This matters as, when building ⟨forbidden state, action⟩ pairs from dead ends, the dead ends are regressed through every action that could possibly lead to them. Thus, as every action reduces the amount of time remaining, every dead end is regressed through every action in the problem, which has a non-trivial computational cost. As such, whilst our techniques are indeed reducing the amount of time spent searching for weak plans, this has a proportionally smaller impact on overall runtime, as plotted on the graph.

4.6.2 Numeric Dead End Generalisation

To ascertain the impact of numeric dead end generalisation, we ran a sensitivity analysis in Rovers and AUV[3], in which this feature was disabled and only propositional dead end generalisation was used. The results of this is shown in Figure 4.4, looking at time taken to solve problems (top) and how many ⟨forbidden state, action⟩ pairs were built (bottom). For dead ends that are more general, one can expect to see fewer ⟨forbidden state, action⟩

[3]Flat Tyre TPP has no dead ends – flat tyres can be repaired indefinitely.

pairs: making a dead end more general allows it to substitute a number of less general dead ends. In turn, this leads to improved performance as dead ends arise from expanding states during search, and hence take time to find.

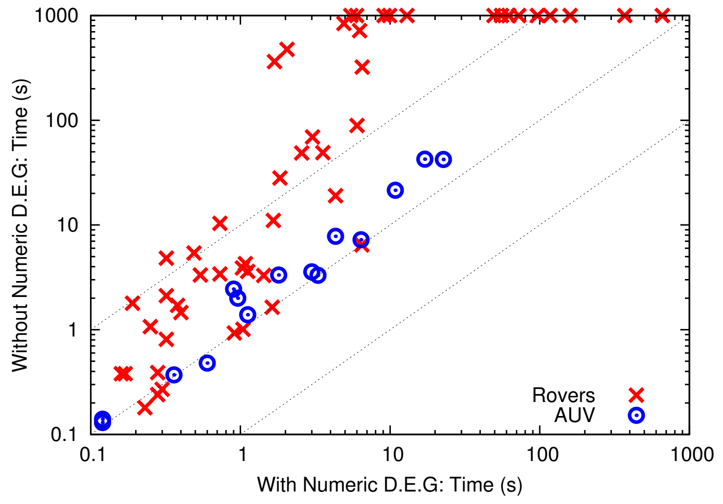
In terms of time taken, there is a clear benefit to using numeric dead end generalisation in Rovers. As in the earlier tests, there are a number of problems at $y = 1000$ that would otherwise be unsolvable. This is attributable to the reduction in the number of FSAPs needed – over an order of magnitude fewer in most cases. Though less dramatic, the same situation arises in AUV as well. A consistent reduction in the number of FSAPs needed leads to a consistent reduction in the time taken to solve problems.

4.6.3 Tolerating a Risk of Failure

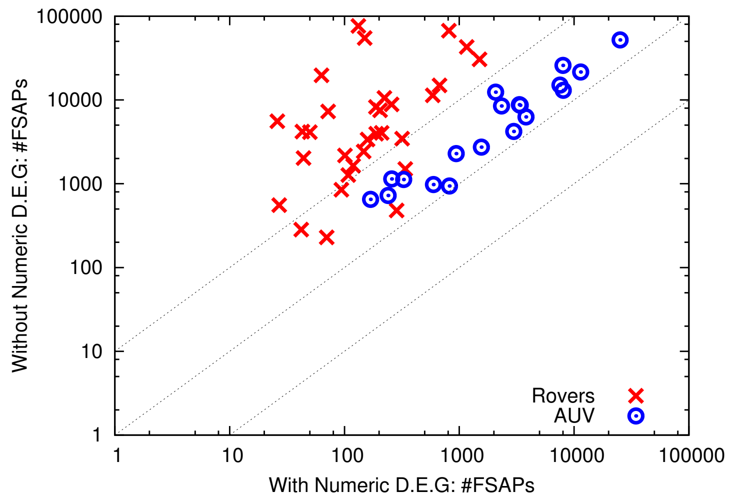
As discussed in Section 4.5, in domains with numeric uncertainty, it is easy to encounter a problem model where no strong cyclic plan can be found. There might always be a small risk of failure, due to a bad outcome repeatedly occurring, and its consequences accumulating numerically within the state. To evaluate our proposed adaptation of the planner to such domains, we modified the three benchmarks used:

- In Rovers, `navigate` has a low-probability outcome that uses energy but does not move the rover. This prevents strong cyclic plans from being found if the rover ever needs to leave a location at which it cannot recharge – there is a small but unlikely risk that repeated attempts to move will fail, using all available power.
- In Flat Tyre TPP, there are a finite number of spare tyres available. Once these have been exhausted, the truck can no longer move.
- In AUV, when turning a valve, there is a small chance the valve will get stuck and must be repaired. It is impossible then to find a strong cyclic plan that relies on turning a valve, as there is a small but non-zero risk that it will repeatedly get stuck, and the available time will be exhausted.

As a baseline, again, we disable our core contributions – numeric regression and numeric dead end generalisation – and use only the propositional variants. However, since it is not possible to find a strong cyclic plan in these domains, the baseline still accounts for probabilities. When solving problems



(a) Time taken



(b) Number of ⟨Forbidden State, Action⟩ Pairs

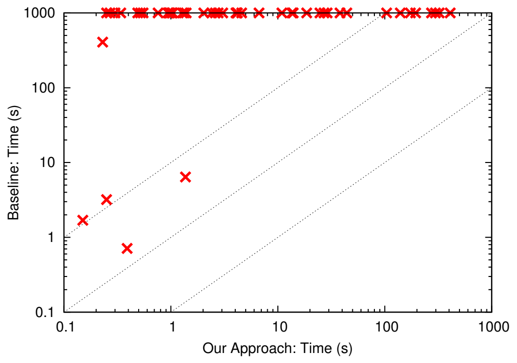
Figure 4.4: Performance with/without numeric dead end generalisation.

from these three domains, the threshold τ (Section 4.5) is set to 0.01, and θ remains at 0.9 (as before, different values of τ or θ lead to the same overall empirical conclusions). Scatterplots of the time taken to find a policy that will reach the goal with this threshold are shown in Figure 4.5.

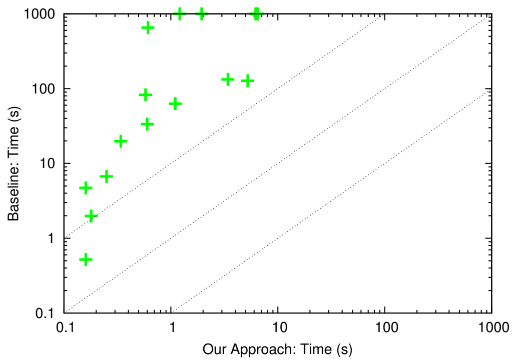
Domain (a): In Rovers, the use of numeric regression and dead end generalisation leads to a staggering improvement in performance. The baseline is able to solve just 5 problems, whereas 50 can be solved with our approach (there are 45 points on the scatterplot along the line $y = 1000$). This echoes the situation in the Rovers domain without probabilities (Figure 4.3), but the difference is even more noticeable now. The domain model here is more challenging, and leads to many more calls to the planning kernel being made:

- In the model without probabilities, `navigate` would always move the rover to its intended destination – the worst outcome has a high energy cost, but if it occurs, `navigate` will still have made progress towards solving the problem. When building the policy and planning for this outcome, it is only necessary to run the planning kernel if there is not enough energy to carry on with what otherwise would have been the rest of the plan (according to the NCTs in the partial states in the policy).
- In the model with probabilities, though, one outcome of `navigate` is that the rover does not move at all – it just uses power, which makes no progress towards solving the problem. This is a far worse situation to have to plan for. Consider a state S, where $S \vDash ps$ for some $(ps, \texttt{navigate})$ in the policy. If the 'failure' outcome of `navigate` occurs, one of three things happens. First, in the best case, a state S' is reached where $S' \vDash ps$, as there is still enough energy remaining, hence navigate is tried again. Second, the planning kernel is invoked and finds an alternative weak plan to the goals. Third, in the worst case, the planning kernel is invoked and, after extensive search, concludes that S' is a dead end. This third case results in a substantial increase in the amount of time spent planning, compared to the previous model.

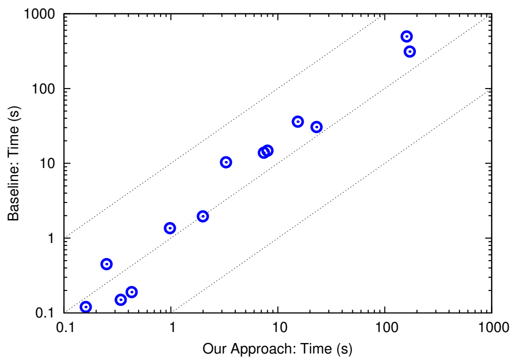
Domain (b): In Flat Tyre TPP, our techniques reduce the time taken to solve problems by over an order of magnitude in all but the smallest evaluation problem. Mirroring what we saw in Rovers, this is a larger improvement than in the domain model without probabilities, since limiting the number



(a) Rovers with Probabilities



(b) Flat Tyre TPP with Probabilities



(c) AUV with Probabilities

Figure 4.5: Time taken to solve problems in the three evaluation domains with probabilistic outcomes, comparing our approach (X axis) to a baseline in which only propositional regression and propositional dead end generalisation are used (Y axis). Axes are log scaled. Tests that timed out are plotted at $t = 1000$.

of tyre repairs makes the problem fundamentally harder. The extent of the improvement is that, when using our techniques, there are 4 problems that were not otherwise solvable by the baseline (plotted at $y = 1000$).

Domain (c): In AUV, as with the domain model without probabilities, the planner spends a lot of time generating $\langle$forbidden state, action$\rangle$ pairs in either configuration, masking the reduction in time spent searching for weak plans. Nonetheless, there is still a consistent reduction in overall time taken to find a policy by around a factor of 3 (excluding noise due to problems solved in less than a second). This is a pleasing result – even in domains such as this that are especially troublesome with respect to the underlying policy building approach, we offer a useful reduction in time taken, allowing the approach to be considered for application to a broader range of domains.

4.7 Conclusion

In this chapter we presented an effective planner for problems with both propositional and numeric uncertainty, where actions have multiple outcomes and numeric effects are drawn from Gaussian distributions. We introduced a partial state representation that captures only relevant numeric information, allowing us to build upon prior work and define the following key concepts of effective policy building:

- Numeric regression, which limits the total number of partial states contained by the policy.
- Numeric dead end generalisation, which allows a newly discovered dead end to expand into a range of similar dead ends.

We evaluated our contributions by comparing a version of our planner with the new features enabled against a baseline version with them disabled. Empirical results showed a significant reduction in the time needed to build a policy, both when we require a strong cyclic plan, and when we accept a risk of failure.

Our work so far has focused on numeric uncertainty drawn from Gaussian probability distributions. In terms of the breadth and generality of domains our planner can tackle, one could argue it suffices to stop here and approximate all uncertainty as a combination of Gaussians. However, we believe it worth investigating whether or not it is feasible to represent and use arbitrary probability distributions in the planning process. We explore this in the following chapters.

Chapter 5

Guiding Search Under Arbitrary Uncertainty

In this chapter we focus once again on numeric uncertainty in forwards planning, as we did in Chapter 3. In particular, we focus on the representation of non-deterministic numeric effects as probability distributions. In Chapter 3, we approximated all uncertain values as Gaussian distributions, but this is not always accurate. In this chapter 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. We integrate our approach into an existing forwards planning setting, and use it to improve how well the states expanded by search reflect reality.

The contents of this chapter have been published in the Proceedings of the Workshop on Heuristic Search for Domain-Independent Planning at the Twenty-Eighth International Conference on Automated Planning and Scheduling (Marinescu & Coles, 2018).

5.1 Introduction

As we have already discussed earlier in this thesis, there is no question that solution accuracy 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. The situation improves with a Gaussian approximation, as we have investigated in the previous two chapters.

However, there are domains for which a purely Gaussian approximation would not be adequate – domains 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. For instance, if a wheel gets unexpectedly stuck in an obstacle, the battery usage sees large upward spikes and the navigate effect no longer falls in a normal distribution. For this reason, we believe it is important for the planner to be aware of the probability distribution this uncertainty is drawn from, whatever shape it has. We now propose a representation to address this problem.

The goal of the work presented in this chapter is not to improve the speed of the planner, or the amount of nodes expanded by search. Instead the goal is to allow the planner to accurately represent (and still manage to solve) a much broader range of problems than it was previously able to.

5.2 Background

We revisit some of the concepts introduced in Chapter 3, but we now assume arbitrary rather than Gaussian probability distributions for uncertain effects.

Prior work on the planner RTU (Beaudry et al., 2010b) uses a Bayesian network (BN) to model resources and time based on continuous random variables. This planner queries the BN to check the likelihood of a variable remaining in a valid state. We base our planning kernel on this approach, as detailed further below.

The core non-deterministic planning formalism we use in this chapter is the same as the one we defined in Chapter 3, based on Beaudry *et al.*. 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. To tackle this, Beaudry *et al.* use the BN to model uncertainty, and to check that numeric conditions are satisfied with confidence level θ. When each action has only a single outcome, the task of planning is to find a sequence of steps $[a_0, .., a_n]$ that give a state trajectory to the goal $[I, S_0, .., S_n]$, with the BN ensuring that each action's preconditions are true with confidence θ.

The confidence level θ is given as input to the planner from the beginning, as it constitutes a part of the world model. It essentially represents a "cutoff" certainty level necessary to consider a precondition met. For instance, if the uncertain variables from the precondition are sampled from their probability distributions 1000 times, and θ is set to 0.85, then the precondition is considered to hold if the inequality is true 850 times. In our planner, it is possible to prescribe different values of θ for different actions – this is useful in the case of specific actions which may have negative consequences if attempted without fully met preconditions.

The prior work by Beaudry *et al.* is adapted by Coles to assume, for heuristic purposes, that variables take their median value (Coles, 2012). Coles proposes a method to first generate plans that are conservative about resource usage, and then to create branches that can exploit cases when resource usage is less than pessimistically expected. This method of branching inspired part of our work as well.

Building on top of Coles' approach, in addition to employing the median when computing the heuristic, we introduce the concept of "offset" in Chapter 3. This is a safety margin with which preconditions must be met $\theta\%$ of the time. It is calculated as the difference between the median and the θ'th percentile of a distribution.

Using the offset, we evaluate whether numeric preconditions are true for a threshold of certainty θ as above, and we define a heuristic which is admissible for monotonically worsening uncertainty. In the case where an effect influences the uncertainty non-monotonically (e.g. assigns it a value rather than increases it by a value), then in the heuristic the offset is reset back to zero.

The techniques we introduce in Chapter 3 were predicated on the uncertainty having a Gaussian shape, thus allowing us to use its analytic form in our calculations. They can however be applied in the case of arbitrary uncertainty as well. As an analytic form is no longer available, in this chapter we propose a sampling-based approach to uncertainty, which still allows us to check if preconditions are true with θ certainty and preserves the improvements brought about by our novel heuristic.

By "sampling-based" approach we are referring to an approach which is given information about the world model in the form of probability distributions which can be repeatedly sampled during planning. These probability distributions are created together with the domain file, as they constitute part of the world model. They are provided to the planner as input. Planning happens entirely offline - there are no calls to the planner made during plan execution. Further in this chapter we explain how sampling is done from any relevant probability distributions during the search for a potential plan, via a belief network.

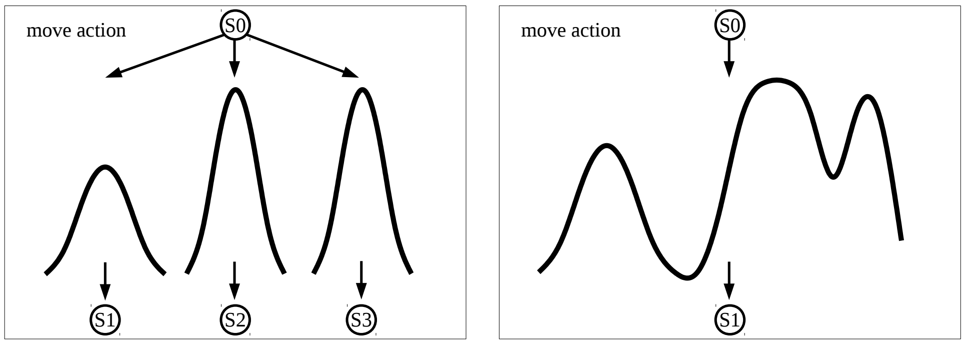


Figure 5.1: Multiple outcomes (left, Chapter 4) vs single outcome (right, Chapter 5).

5.3 Approach

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 distributions (not just ones with analytic representations like Gaussians), without incurring a high computational cost. Our representation should allow the planner to find strong plans which meet a required precondition certainty threshold.

5.3.1 Motivating Comparison

Before we present our approach, we would like to draw attention to a competing piece of prior work – that in Chapter 4. There, we use a domain model where actions have multiple uncertain numeric effects, each of them drawn from a Gaussian distribution. For example, the `navigate` action has 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).

Since it is not unreasonable to approximate an arbitrary probability distribution to a combination of Gaussians, this prior work can arguably be used instead of the more general and possibly more computationally expensive approach we propose in this chapter. However, there is a problem with these hard-coded multiple outcomes – they need to be specified in the world model. This may be impractical, as it forces the model designer to make guesses about an uncertain environment, beyond just using the data they have available.

Hence, we strongly believe it is worth investigating an alternative implementation that only uses one outcome and offers only the known information about that outcome in a single, complex probability distribution. This implementation should not require giving the planner any clues about whether discrete outcome modes exist, or how they are shaped.

Figure 5.1 shows these two competing approaches side by side:

- (a) An action with three outcomes, each drawn from a Gaussian distribution. These outcomes, together with their Gaussian parameters, are hard-coded in the PDDL domain file. Depending on how the physical environment behaves, these Gaussians may not be a highly accurate representation of reality.

- (b) An action with one outcome, drawn from an arbitrarily-shaped distribution provided in a separate data file. This distribution could be improved with additional data at a later time. Due to the lack of constraints on its shape, this distribution has the potential to more accurately represent reality.

5.3.2 The Bayesian Plan Network

Our goal in this chapter - similarly to our goal in Chapter 3 - is to search for a strong plan with preconditions that hold $\theta\%$ of the time, in the presence of numeric uncertainty. In order to check precondition satisfaction, we need a way to track uncertain variables in any state (this time without assuming uncertainty is Gaussian).

To help us represent an arbitrary probability distribution from which a numeric variable is drawn in any given state, we introduce the concept of a Bayesian Plan Network (BPN). This is a sampling-based representation of uncertainty at any point in the reachable search space, based on the accumulated uncertainty in past states. Its purpose is to check whether an action precondition holds, given the uncertainty at the time of action application.

The BPN is essentially a belief network - a directed acyclic graph where each node is a random variable, and each edge is a dependency between two random variables. Such a belief network can be used to answer probabilistic questions about the random variables it contains. We build this network sequentially from a given plan, by applying numeric action effects one by one. The following sections show a step-by-step example of how a BPN for a given plan is built (culminating in Figure 5.7) and how it is then queried in order to answer the question "is this precondition satisfied with more than X% probability, given the uncertainty in the plan so far?".

The result of querying the BPN is used to better inform state expansion about the uncertain environmental conditions. When querying the BPN, we input the cutoff value θ mentioned in earlier sections. Throughout the following, we use the value $\theta = 0.9$ to illustrate our approach (i.e. a precondition needs to hold in 90% of sampled instances to be considered true). This fixed value is for demonstration purposes only; the concepts we introduce work the same way regardless of the value of θ.

To briefly summarise our approach before working through an example below - we're achieving the same goal we did in Chapter 3, but we aren't using the concept of "offset" anymore. Instead we are checking preconditions in their original form ($v \geq c$) via multiple sample runs of the BPN, to see whether they are true θ% of the time. We do this both when expanding states during search, and when building the RPG for heuristic computation.

5.3.3 Building a Bayesian Plan Network

The BPN is a directed graph constructed from a plan, where each node represents either a probability distribution D or a variable v.

Distribution nodes can be described as "source" nodes – they have no parent nodes, as their value does not depend on other variables in the plan. They are akin to buckets of samples (which can be described analytically, like the shape of a Gamma distribution, or empirically, like a collection of sensor measurements). Distribution nodes are represented as blue rectangles in Figures 5.2-5.7.

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 samples all nodes in the BPN once. Distribution nodes are represented as grey ovals in Figures 5.2-5.7.

A BPN corresponding to a given plan contains nodes for all variables affected by the numeric effects of the actions in that plan. The BPN also contains nodes for all probability distributions corresponding to these actions. If an numeric action effect is not uncertain, its distribution is "degenerate", i.e. consists of only one constant value. This is the case for the effect of the `drill` action on the *data* variable in the example PDDL shown further below.

The edges in the BPN graph can be weighted – coefficients from action effects can be used to reflect how a variable is a weighted sum of previous variables. However, because the example actions below all have coefficients of 1 (e.g. $energy = 1 \times previous\_energy - 50$), they have been omitted from all example figures for simplicity.

The steps for building a BPN are the following:

1. Input a problem description and a potential plan found by the planning kernel (using only median values for all uncertain effects).

2. For each variable that is set in the initial state, create one variable node and one distribution node. Each distribution node is the parent of its corresponding variable node, and represents a degenerate distribution (a distribution that only contains one sample: the value set by the initial state). Figure 5.2 shows an example of this.

3. For each action in the plan, loop through its effects.

   (a) For each effect, create one new variable node for the affected variable. Then create parent links between the new node and all variable nodes whose values are used by that effect (with their respective weights assigned to the parent link, if any).

   (b) If the effect above is non-deterministic, then create one new distribution node containing the samples corresponding to that effect.

   (c) If the effect above uses a constant, then create one new distribution node containing a degenerate distribution (a distribution that only contains one sample: the constant itself).

Consider the following plan in an illustrative Rovers-like domain: [`navigate`, `navigate`, `transmit`, `navigate`, `drill`]. The action definitions are:

```
(:action navigate
   :parameters (?r - rover ?w1 - waypoint ?w2 - waypoint)
   :precondition (and (at ?r ?w1)
                      (>= (energy ?r) 10))
   :effect (and (not (at ?r ?w1))
                (at ?r ?w2)
                (decrease (energy ?r) 10))) ; with uncertainty

(:action drill
   :parameters (?r - rover ?w - waypoint)
   :precondition (and (at ?r ?w)
                      (not (drilled ?w))
                      (>= (energy ?r) 50))
   :effect (and (drilled ?w)
                (increase (data ?r) 1)
                (decrease (energy ?r) 50))) ; with uncertainty

(:action transmit
   :parameters (?r - rover)
   :precondition (>= (data ?r) 1)
   :effect (decrease (data ?r) 1))
```

The `navigate` action and the `drill` action each have a non-deterministic decrease effect on the *energy* variable. We denote the probability distributions governing the range of these effects as $D_{navigate\_energy}$ and D_{drill_energy}. During the BPN building process, when adding such an uncertain action effect, we first create a node for the new value of *energy* (the value it will have after the effect is applied). We then add two parent links: one to the prior node for *energy* (which already exists in the BPN), and one to the distribution node for e.g. D_{drill_energy} (which we create immediately before adding the parent link). These links denote that the distribution of the new value of *energy* is the sum of its parents.

Figures 5.2-5.6 show snapshots from the BPN building process for the example plan & action definitions above, with the final BPN appearing in Figure 5.7.

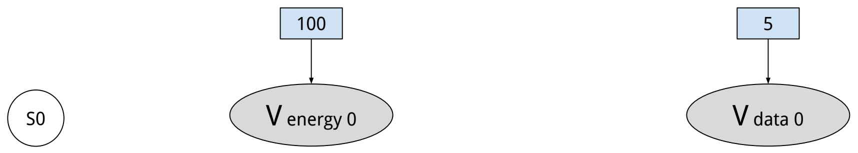


Figure 5.2: BPN in state S0.

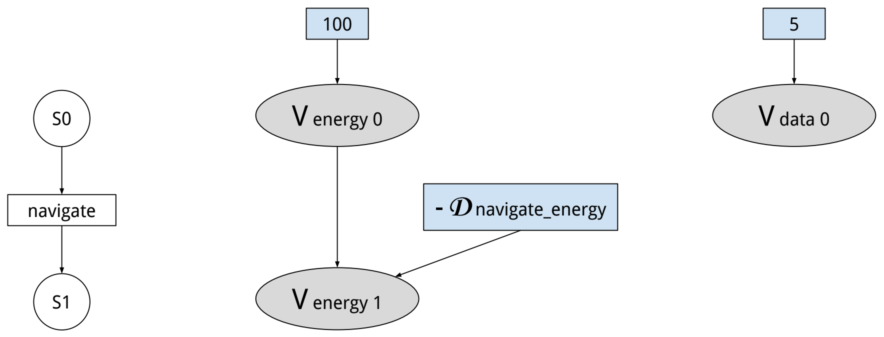


Figure 5.3: BPN in state S1, after applying the `navigate` action. (Corresponding plan shown on the left-hand side.)

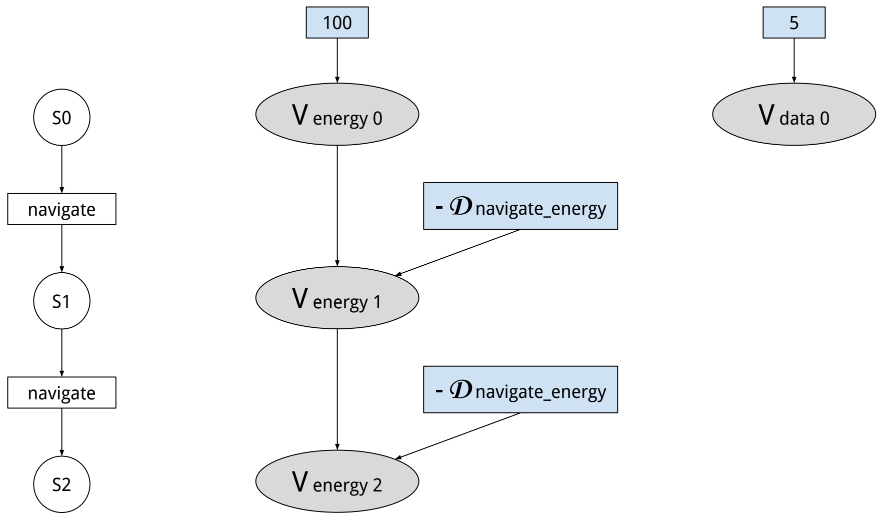


Figure 5.4: BPN in state S2, after applying the `navigate` action. (Corresponding plan shown on the left-hand side.)

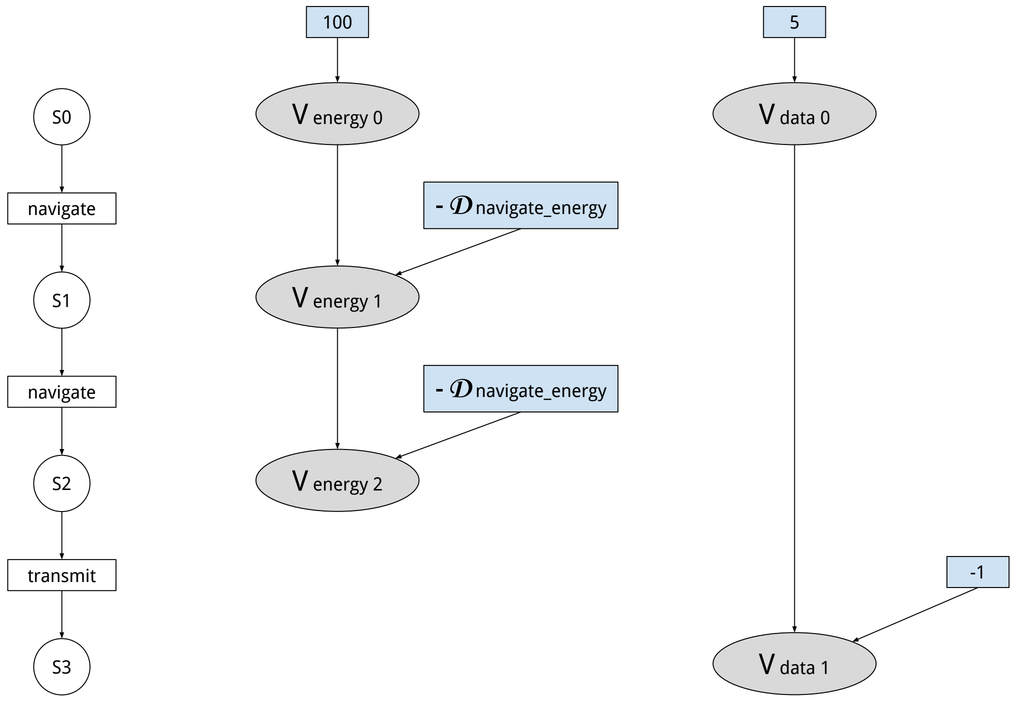


Figure 5.5: BPN in state S3, after applying the `transmit` action. (Corresponding plan shown on the left-hand side.)

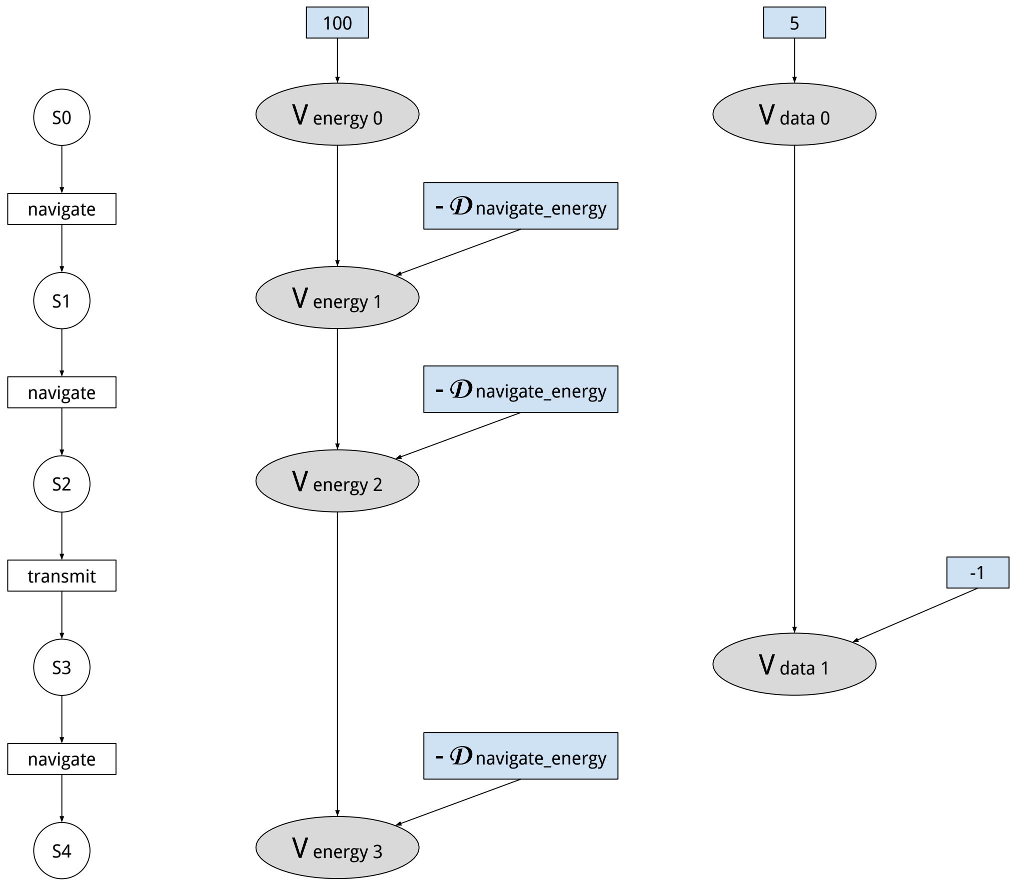


Figure 5.6: BPN in state S4, after applying the `navigate` action.
(Corresponding plan shown on the left-hand side.)

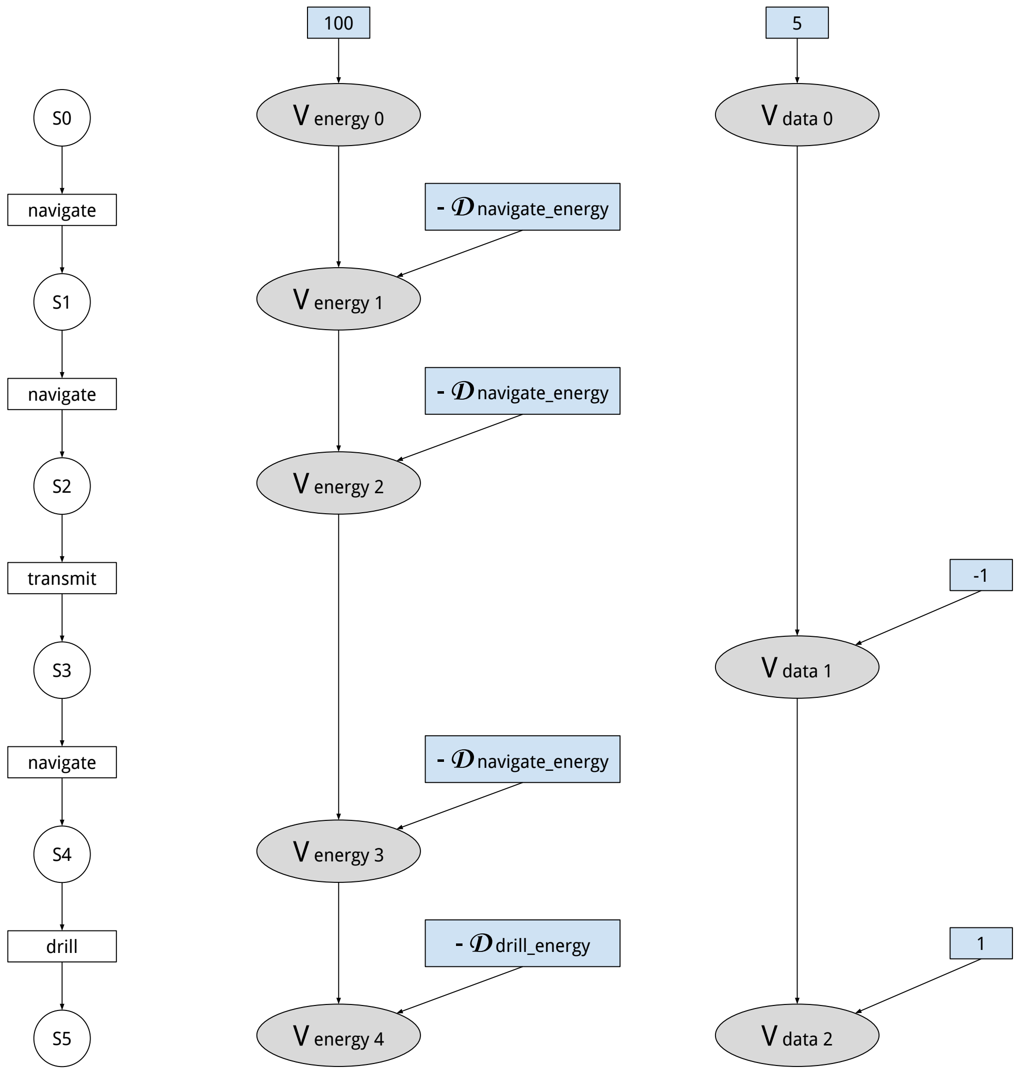


Figure 5.7: Final BPN in state S5, after applying the `drill` action. (Corresponding plan shown on the left-hand side.)

5.3.4 Sampling a Bayesian Plan Network

As mentioned earlier in this chapter, the BPN is used to check if preconditions hold as we expand our Chapter 3 strong plan technique from Gaussian distributions to arbitrary distributions. These precondition checks are done both during search and during RPG building for heuristic computation. Each time we do a precondition check, we input into the BPN a plan so far (from the initial state up until the state in which we are checking the precondition validity), and we receive a boolean output. Internally, this output is computed via a number of sampling operations.

The process of sampling a BPN involves going through all preconditions in the inputted plan, identifying the relevant variable nodes for each precondition, querying those variable nodes N times, and reporting how many times the queried values resulted in the preconditions holding. If the amount exceeds the required fraction θ (e.g. the preconditions in the inputted plan held more than 85% of the time for $\theta = 0.85$), then the BPN reports success.

Each time a variable node is queried, it recursively queries its parent nodes. Each time a distribution node is queried, it returns a randomly sampled value from its probability distribution.

The probability distributions in our BPN can be arbitrarily shaped. For convenient implementation purposes, we split them into three categories:

- Distributions with an analytical form (e.g. Gamma, Gaussian, any polynomials, etc.);
- Data-based distributions (i.e. collections of samples);
- Multi-modal distributions (i.e. any weighted combination of the distributions above).

Below we include a listing of two example distribution specifications - one parametrised Gamma distribution, and one data-based distribution. The parameters of the Gamma distribution represent:

- `zero:` The "origin" point of the Gamma distribution on the X axis.
- `from, to:` The "window" surrounding the Gamma distribution on the X axis.
- `alpha, beta:` The shape parameter and rate parameter of the Gamma distribution.

```
"action-name:effect-name":
{
   "type": "gamma",
   "from": 5,
   "to": 11,
   "zero": 10,
   "parameters":
   {
      "alpha": 2,
      "beta": 2
   }
}

"action-name:effect-name":
{
   "type": "data",
   "parameters":
   {
      "file": "navigate-energy-historical-samples.txt",
   }
}
```

These distribution specifications are stored in a separate file that accompanies the domain file and has a custom JSON syntax. Further listings can be found in Appendix C.

5.3.5 Practical Considerations

To recap, the required elements for using the BPN technique are the following:

1. A domain file (provided by the user);
2. A separate JSON file describing the probability distributions for each uncertain action effect (provided by the user);
3. A required certainty value θ (provided by the user);

4. A sequence of actions (plan so far) whose preconditions need to be checked for validity with certainty θ. This sequence is not provided by the user but rather generated during forward-search planning. There is no need to handcraft plans in order to use a BPN.

In addition to the standard domain modelling done for any planning problem, inputs 2 and 3 will need to be handcrafted as part of the world model, in advance of planning. As mentioned in Section 5.2, our planner allows the possibility of requiring different certainty values for different actions - we might, for instance, want the `navigate` action preconditions to hold 90% of the time, and the `drill` action preconditions to hold 99% of the time, as the consequences of attempting to drill without fully met preconditions might be detrimental to rover integrity.

As far as the inputted probability distributions go, the BPN technique is best suited for situations when we already have some data at our disposal about the uncertain action effects in our domain. From the point of view of domain design, being forced to extrapolate or make assumptions about the world model is almost certainly detrimental to plan quality. If we have accurate uncertainty data, perhaps collected from past experiments in the target environment, or perhaps resulting from some orthogonal investigation (e.g. materials analysis), it would be beneficial to make use of that data fully, rather than abstract it away into a Gaussian.

5.3.6 Representing a Bayesian Plan Network Efficiently

An obvious question to pre-empt would be – won't we sacrifice computational time in order to accommodate sampling a sufficient number of times during the planning process?

The graphical representation of the BPN described in the previous section is useful to aid understanding of how the network functions. However, the sequential computations that stem from this representation can 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 sections above ("is each precondition in a given plan satisfied with certainty θ?"). Our method allows each sample run (populating all nodes in the BPN with 1 sample each) 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 represents a node, and each column represents 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 the above matrix representation with a second matrix containing all the sampled initial values for all uncertain variables. Finally, we use the result to count the number of sample runs in which all the preconditions were satisfied, and to check if that number is above the threshold $\theta \times total\_number\_of\_sample\_runs$.

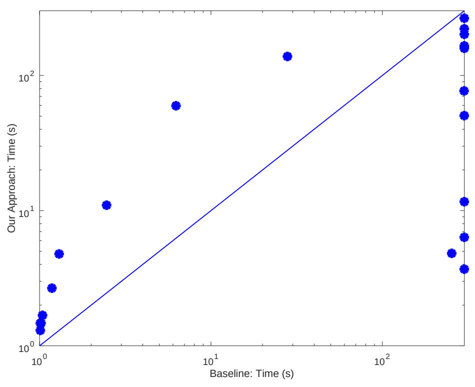
5.4 Evaluation

The aim of our evaluation is to find out whether it is computationally feasible to represent uncertainty generally, rather than represent it as an approximation. We compare the multiple-outcome Gaussian approach from Chapter 4 with the single-outcome arbitrary approach from this chapter.

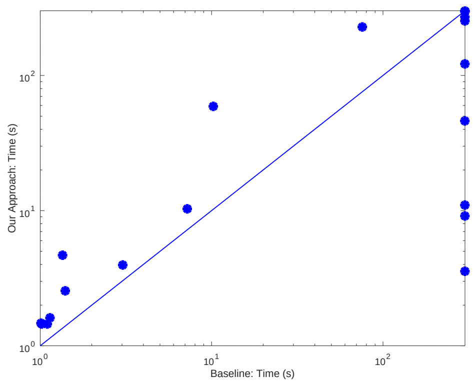
The techniques introduced in this chapter are implemented on top of the Optic+ expansion described in Chapter 3 (finding strong plans under numeric uncertainty).

Our conditions for a successful evaluation are not tied to an improvement in certain metrics, but rather to whether or not the same problems solved by the Chapter 4 approach are still solvable with our current approach, within reasonable bounds (i.e. the metrics have the same order of magnitude). As before, we measure 1) time elapsed and 2) number of nodes expanded in order to arrive at a solution, on a set of 20 problems.

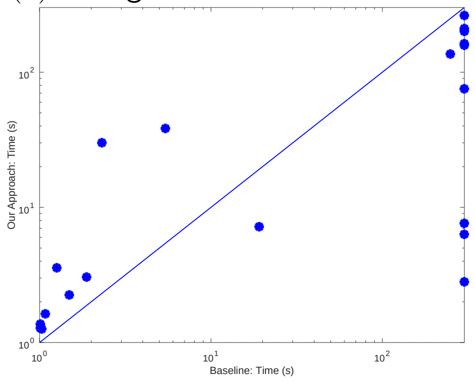
To support the goal of showing that our approach is computationally feasible, we choose a domain where numeric uncertainty can reasonably be represented as several Gaussians, but would also benefit from a more fine-grained representation in case our approach does prove feasible. The Rovers domain (Coles, 2012) fits these requirements: the `navigate` action exhibits uncertainty due to different factors that are difficult to model: wheel slippage, soil characteristics, incomplete obstacle data. While a collection of Gaussians would be adequate to model these factors, we believe that, if more accurate data on uncertainty is available (e.g. sampled battery data from past runs



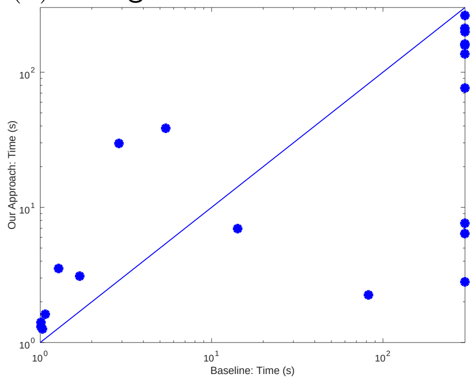
(a) `navigate` action has 2 modes



(b) `navigate` action has 3 modes



(c) `navigate` action has 4 modes



(d) `navigate` action has 5 modes

Figure 5.8: Time taken to find strong cyclic policies (X axis) versus strong plans (Y axis). Axes are logarithmic.

of the rover), the planner should make use of it.

We run our experiments on a system with a 1.8GHz Intel Core i7 and 16GB of RAM, with execution limited at 300 seconds (5 minutes) and 1 GB of RAM.

Information about uncertainty is conveyed differently to each planner. For the baseline (Chapter 4) planner, the PDDL domain file contains the parameters of the Gaussian distributions associated with each non-deterministic effect. For the current (Chapter 5) planner, an additional data file contains a set of samples taken in a preprocessing phase. While our approach supports sampling this set out of any probability distribution, in this case it is taken out of several Gaussian distributions, in order to match the baseline setup. A listing of the files used for each approach is shown in Appendix C.

Figure 5.8 shows the time taken by:

- X axis: The baseline approach – finding a strong cyclic policy while having multiple outcomes for the `navigate` action, each represented as a Gaussian distribution.
- Y axis: The current approach – finding a strong plan while having a single outcome for the `navigate` action, represented as an arbitrary probability distribution with multiple modes.

The most striking result is the amount of problems on the far right-hand side of each graph – these were not solved by the policy approach within the given time limit, however were solved by the strong plan approach. We will come back to this result after describing what is different in each of the 4 graphs.

5.4.1 Variants

We chose to evaluate both approaches on several variants of the Rovers domain, each variant with a different number of outcome modes for the `navigate` action. The reason for introducing these variants was to check whether or not an increased policy branching factor would have a negative impact on the time taken by the baseline approach. We made an early guess that, since policies might get larger with a higher number of outcomes, while strong plans would still only draw samples from one outcome, the latter would be faster.

The 4 evaluation graphs show the results obtained on each of these variants, as follows:

- 2 outcomes/modes - `unlucky` (16 energy, 16 variance), `nominal` (10 energy, 10 variance).
- 3 outcomes/modes - Same as above, plus `lucky` (8 energy, 8 variance).
- 4 outcomes/modes - Same as above, plus `luckier` (7 energy, 7 variance).
- 5 outcomes/modes - Same as above, plus `luckiest` (6 energy, 6 variance).

With the baseline approach, each variant is represented in a different domain file containing a different number of outcomes for the `navigate` action. Each outcome indicates its parameters (median and variance) directly in the PDDL file.

With our current approach, the domain file is the same for all 4 variants, but each one is linked to a different data file (associated with the `navigate` action by name). A data file can contain either sample data, or a set of parameters to one or more probability distributions. In the latter case, the parameters will be used to generate sample data in a preprocessing phase (before planning begins). In this evaluation, in order to keep the comparison fair, we populated the data files with a combination of Gaussians with identical parameters to those in the baseline approach.

Our early guess about an increased policy branching factor having a negative impact on the baseline approach turned out to be incorrect. There is only a marginal suggestion in the data that this might be the case. Out of 20 problems, this is how many were solved faster by our current approach compared to the Chapter 4 approach:

- For 2 outcomes: 11 problems.
- For 3 outcomes: 10 problems.
- For 4 outcomes: 11 problems.
- For 5 outcomes: 12 problems.

The slight increase in the number of problems where the policy approach is slower as the number of outcomes grows is not sufficient to indicate a trend.

5.4.2 Solved and Unsolved Problems

Both approaches suffer from possible challenges with regard to solving problems. The strong cyclic policy might take too long to build for particularly large problems. The strong plan might take too long or be impossible to find for problems where a single outcome does not model the behaviour of the environment well enough.

The experimental results we obtained indicate than the former occurs more frequently than the latter in the Rovers domain. A total of 38 out of 80 problems (with no clear trend with respect to the number of outcomes) were solved by the strong plan approach, but not by the policy approach. Nearly all of these were large problems with 4 or more rovers, which is consistent with the potential bottleneck of strong cyclic policies mentioned in the previous paragraph.

The data points above each bisector line (solved faster via policy than via strong plan) tended to be small in size. This suggests that, on small problems, Gaussians offer a good enough approximation of uncertainty and allow policies to leverage the speed of the analytical Gaussian mathematics used in Chapter 4. They still, however, have the drawback of being reliant on explicit information on outcome modes from the user via PDDL.

5.5 Conclusion

In this chapter, 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 by using matrix multiplication.

What we have essentially shown is the feasibility of going beyond Gaussians in the pursuit of an accurate representation of the real world, without incurring any significant computational costs.

On the one hand, if a Gaussian approximation of uncertainty is deemed adequate for the problem at hand, and no additional information is available to the planner, the approach in Chapter 4 is indeed effective.

On the other hand, representing arbitrary probability distributions requires less effort in terms of domain modelling, does not rely on making guesses or giving the planner clues about how the environment might react, and is computationally feasible.

According to our research so far, if information about uncertain probability distributions is available when designing the domain model, it is beneficial for the planner to use this information in its entirety, rather than abstract it into a Gaussian distribution, or into a single median. In this chapter we presented an implementation that makes it tractable to do so, thus allowing the planner to take advantage of all the available information.

Chapter 6

Building Policies Under Arbitrary Uncertainty

In this chapter we address two limiting factors of our work so far. First, policy building for non-deterministic domains (Chapter 4) is constrained to outcomes drawn from Gaussian probability distributions. Second, strong planning with effects drawn from arbitrary probability distributions (Chapter 5) cannot deal with problems that require splitting these effects into branches.

We propose what we consider to be the next logical step in pushing our planner towards increasingly more general representations of uncertainty: an extension of our prior policy building approach that allows each individual action outcome to be arbitrarily distributed.

6.1 Introduction

Our prior work on policy building with non-deterministic numeric effects is limited in scope when it comes to the types of probability distributions it can accommodate. We acknowledge that data driven models of uncertainty are more practical than the assumption that all processes are simply Gaussians. However, the policy building approach we presented in Chapter 4 is only applicable if all uncertain action outcomes are Gaussian.

In addition, the contribution to strong planning we described in Chapter 5 is unable to tackle branching on multiple action outcomes. It is not uncommon for non-deterministic planning problems to require branching in order to plan for different eventualities, and simply trying to find an unbranched plan with certainty θ may not be possible. Consider the following example, where D is a probability distribution containing two similarly large clusters of data, one around 25 battery consumption, and another around 75 battery consumption:

```
(:action a
   :precondition (and (>= battery 100)
                      (not (a-done)))
   :effect (and (decrease battery D)
                (a-done)))

(:action low-battery-procedure
   :precondition (and (a-done)
                      (<= battery 50))
   :effect (and (goal)))

(:action high-battery-procedure
   :precondition (and (a-done)
                      (>= battery 51))
   :effect (and (goal)))
```

Assuming that in the initial state the battery level is 100, action `a` will consume an amount of battery dictated by the distribution D. Afterwards, the planner must choose which one of the goal-achieving actions to apply. Due to the nature of distribution D, it is unlikely that either action's preconditions will be satisfied with confidence θ, therefore the problem will not be solved by our Chapter 5 approach. We have conducted a small-scale experiment on the domain above and have observed that this is indeed the case.

Allowing the planner to branch on the outcome of action `a` would solve this issue. Since the outcome is represented as an arbitrary distribution, it stands to reason that any potential branches should also be represented as arbitrary distributions. This preserves all of the known data instead of approximating it and discarding potentially useful information. As has been the case throughout this thesis, we believe that, as long as it is computationally feasible to do so, it is always beneficial to have more information about uncertainty.

6.2 Background

Since our proposed approach is centred around policy building, we reiterate below some of the policy-related formalisms already defined in Chapter 4. We also revisit the Bayesian Plan Network structure from Chapter 5.

We are once again in a setting where actions can have multiple outcomes, and each outcome has a set of associated effects. A solution to problems containing such actions can be represented as 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 of the action outcomes 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 $Eff(a)$ is replaced by several actions, one for each $o \in Eff(a)$, whose preconditions are $Pre(a)$ and whose effects are just those corresponding to o.

In the case of propositional uncertainty, prior work (Muise et al., 2012) (Muise et al., 2014) on the planner PRP builds a policy by making repeated calls to a deterministic planning kernel that finds weak plans. 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 above, we extend the policy building process from propositional to numeric uncertainty in Chapter 4. We achieve this by defining the process of regression through non-deterministic Gaussian distributed numeric effects. This offers us the additional benefit of generalising numeric dead ends in order to prune them more efficiently.

The process of regression takes as input a partial state ps. 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 – i.e. applying the action in ps' returns us to ps.

To regress through the weak plan, we begin from the goals and go backwards, producing one ⟨partial state, action⟩ pair corresponding to each plan step. These pairs $\langle ps, a \rangle$ are added to the policy one by one. If, while regressing, a state S is reached that matches a pair already in the policy, then all the outcomes of the corresponding action are applied. Otherwise, the planning kernel is invoked again from S. This produces another weak plan, either to the goals (G) or to some other partial state already in the policy (G'), in which case the $\langle ps, a\rangle$ pairs from this new weak plan are also 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.

6.3 Approach

In this section we show how the policy building process described above 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.

We regard regression as having two distinct components:

1. The propositional elements of regression; these remain the same as in prior work (Muise et al., 2012).

   We leverage the prior work approach to propositional uncertainty for its considerable speed when triaging possible matches to a given partial state. This is important to ensure computational feasibility, as the added computational effort of our contribution is largely concentrated inside the procedure of checking the policy for existing partial state matches.

2. The numeric elements of regression; these are the focus of the approach in this chapter.

   We show that it is computationally feasible to perform regression through non-Gaussian effects, and we provide an efficient implementation to do so.

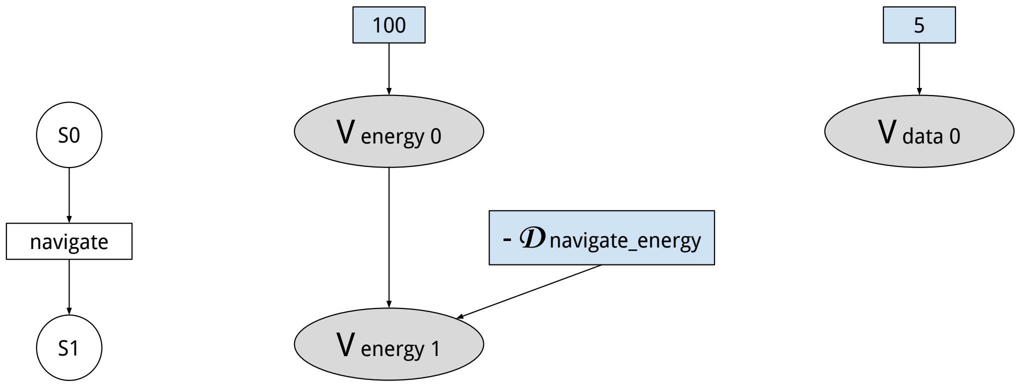


Figure 6.1: Single-outcome BPN in state S1, after applying the `navigate` action.

6.3.1 Revisiting the BPN

In Chapter 5 we introduced a representation of uncertainty that allows effects to be drawn from an arbitrary probability distribution. We defined a directed graph-like structure called a Bayesian Plan Network (BPN) to capture how the accumulated uncertainty at any point in a plan depends on past action effects.

Previously, we used this BPN during search and RPG expansion, in order to check whether a potentially reachable state was indeed reachable with certainty θ. To perform this check, we built a BPN from the actions which led up to that state, then queried the BPN about the preconditions of those actions. Internally, the BPN used repeated sampling to verify whether all preconditions in the chain were true $\theta\%$ of the time.

In this chapter, we use a BPN for a slightly different purpose, as we detail in Section 6.3.3 and Section 6.3.4. The task of planning is to build a policy, and the crucial step during which we need the BPN is when checking if a state S matches a partial state ps that already exists in the policy.

6.3.2 Including Multiple Outcomes in the BPN

We briefly summarise the BPN construction steps laid out in Chapter 5 Section 5.3.3. A BPN is made based on a sequence of actions (a plan), and it contains nodes for all variables affected by those actions, and for all probability distributions corresponding to those actions. To build a BPN, we first

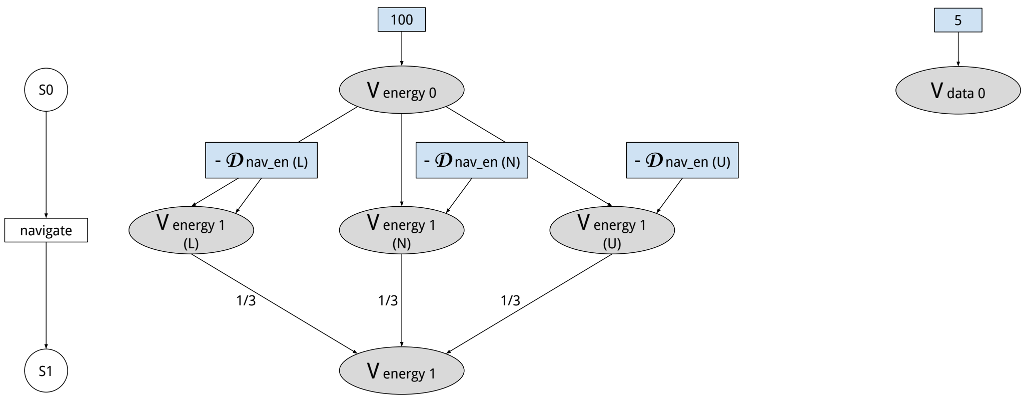


Figure 6.2: Multi-outcome BPN in state S1, after applying the `navigate` action.

create two nodes per value set in the initial state (one variable node and one parent distribution node containing a constant). We then go through the plan step by step, and for each action effect we create additional variable nodes and distribution nodes, corresponding to each affected variable. Newly created variable nodes are linked to existing variable nodes via directed edges reflecting a sequential dependency (e.g. the value of `energy` in state 3 depends on the value of `energy` in state 2). Once built, the purpose of the BPN is to be recursively sampled starting from the leaf nodes, in order to check whether all action preconditions in the plan are true for at least θ% of the sampled values.

In addition to the BPN construction steps above, we introduce a new step specific to domains with multiple action outcomes. Each time an action with multiple outcomes is encountered during BPN construction, we first create one variable node for each outcome (and its corresponding distribution node). Afterwards we create an additional "mixed" node whose parents are the nodes of each outcome. The parent links are either weighted equally with 1/N each (for N outcomes), or weighted based on probabilistic information found in the domain. The newly created mixed node will now be treated as the most up-to-date version of the variable it represents.

To visualise this additional step, contrast Figure 6.1 with Figure 6.2. The single-outcome setting of Chapter 5 corresponds to the BPN snippet in Figure 6.1. The multi-outcome representation of the same "plan" [`navigate`] is shown in Figure 6.2.

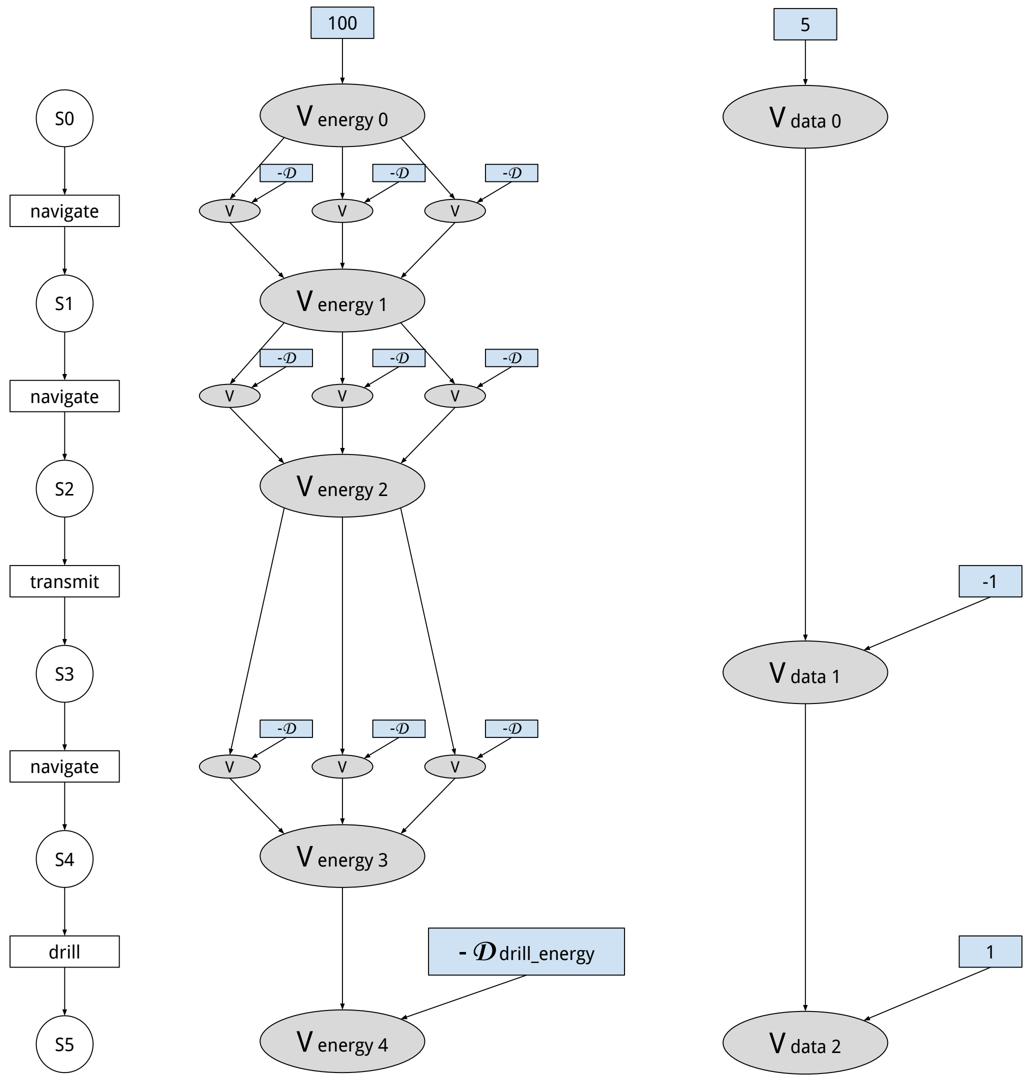


Figure 6.3: Final multi-outcome BPN in state S5, after applying the `drill` action.

As a worked example corresponding to Figure 6.2: if the `navigate` action has three outcomes (`lucky`, `nominal`, `unlucky`) that change `energy` differently, then we create three intermediary variable nodes for `energy`. Each intermediary variable node is created together with its corresponding distribution node, as though only one of the action outcomes had occurred. Afterwards we create an additional node for `energy`, and make parent links from all three prior nodes to the current one. If the domain file contains probabilistic outcome information (e.g. 0.25, 0.5, 0.25 respectively), then these values appear as weights on the parent links. Otherwise, each of the three parent links has a weight of 1/3.

The example plan in Chapter 5 ([`navigate`, `navigate`, `transmit`, `navigate`, `drill`]) has its full multi-outcome BPN represented in Figure 6.3, with minor index and weight simplifications in the interest of visibility.

6.3.3 Building a Policy with the BPN

So far we are able to build a BPN and recursively sample it to check whether a potential plan (which can now include multi-outcome actions) is a valid plan, given the uncertainty in the domain. We can take advantage of this feature in the policy building process.

We briefly reiterate what the policy building process looks like, and draw attention to the fact that we can no longer use a Gaussian-based *offset* when checking the validity of preconditions in a potential plan, as we are now in a setting where probability distributions can be arbitrarily shaped. We are instead proposing to tackle this issue by using the BPN.

The steps for building a policy are as follows (we expand further on the BPN usage in Section 6.3.4):

- Use the planning kernel to find an initial weak plan.
- Use the weak plan to build an initial BPN as described in Chapter 5. If the BPN indicates the weak plan is valid, use the weak plan to seed a policy.
- Start the recursive policy building algorithm described in Section 6.2. Use the BPN when checking if a newly encountered state already has a matching entry in the policy, or if it needs to invoke the planning kernel again (this BPN check will be detailed in the next subsection). This check happens on line 5 of Algorithm 4.

- Stop when all reachable states have a matching entry in the policy.

When we check if a new state S matches an existing policy entry ps on line 5, we concatenate the plan up to S with the plan from ps onwards indicated by the policy entry. If this concatenation results in a valid plan according to the BPN, then we have found a match.

To be able to do the check on line 5, we need to implement an extension to the elements stored by the policy. Instead of the ⟨partial state, action⟩ pairs described in Chapter 4, the policy is now augmented to contain ⟨partial state, list of actions⟩ pairs. These lists of actions are obtained by storing the weak plan steps from each partial state to the goal.

Algorithm 4: Building a Policy with the BPN

```
   Data: Planning task, initial state $I$ and goals $G$
   Result: Policy $P$
1  add $I$ to open list $O$;
2  while $O$ not empty do
3      take state $S$ from $O$;
4      // The BPN contribution: check if a ⟨partial state, list of actions⟩
       pair exists in $P$ such that $S$ matches it;
5      if $\exists\langle ps, [a_0, ..., a_n]\rangle \in P$ such that $S \vDash ps$ then
6          apply $a_0$ to $S$; add the resulting states in $O$;
7      else
8          find a weak plan from $S$ to $G$;
9          if weak plan not found then
10             reset $P$ and mark $S$ forbidden;
11         else
12             generate $\langle ps, [a_0, ..., a_n]\rangle$; add them to $P$;
13             apply $weak\_plan\_a_0$ to $S$; add the resulting states in $O$;
14 return $P$
```

6.3.4 Partial State Matching with the BPN

So far we presented the high-level steps of policy building when actions have arbitrarily-distributed outcomes; one of these high-level steps makes use of a

BPN to check for state matches. In this section we give a detailed explanation of what happens during this check.

To find out whether or not the policy already knows what to do in a new state S, we first do the propositional triage. We look for all the partial states in the policy which match the facts in our new state, and thus obtain a list of possible candidate matches.

To then choose a candidate match from the list, we need to change how we check if preconditions hold in the original policy building algorithm from Chapter 4. We are no longer using a Gaussian-based *offset* - instead we are checking the original form of the preconditions ($v \geq c$) multiple times, each time plugging in the variable values obtained from one sample run of the BPN. We then check if the fraction of sample runs resulting in preconditions holding is at least θ.

Below is the sequence of steps taken to check whether a new state S matches a given $\langle$partial state, list of actions$\rangle$ pair that already exists in the policy:

- Obtain the following two plans:

    1. The plan so far from the initial state up to the new state S which we are currently looking up in the policy.
    2. The list of actions from the potentially matching partial state to the goal. As mentioned earlier, this list of actions can be found stored in the policy, together with the partial state we are currently checking.

  Concatenate the two plans (line 1 in Algorithm 5).

- Build a BPN from the potential plan resulted from the concatenation (lines 2-10 in Algorithm 5).

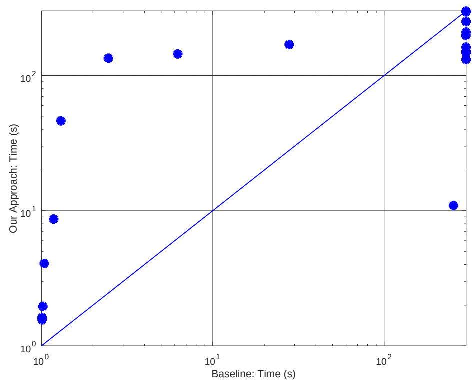
- Sample the BPN a suitable amount of times (line 11 in Algorithm 5).

- Loop through the potential plan and count how many times the preconditions at each step were true given the sampled values (lines 12-17 in Algorithm 5).

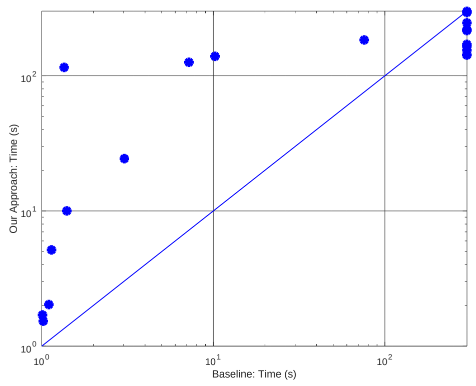
- Check the counters to verify if all preconditions were satisfied with the required degree of certainty θ (e.g. for $\theta = 0.9$, if they were satisfied at least $(0.9 \times N)$ times out of N samples) (lines 18-22 in Algorithm 5).

Algorithm 5: Partial State Matching with the BPN

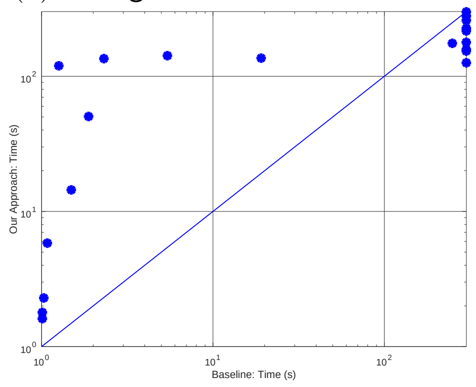
```
Data: Planning task, initial state $I$, goals $G$, policy so far $P$,
queried state $S$, queried $\langle$partial state, list of actions$\rangle$ pair
Result: Bool (match / no match)
1  concatenate [plan from $I$ to $S$] and [plan from $S$ to $G$] into $p$;
2  for $v$ in $I$ do
3      create node $V_{v\_0}$ with parent node $D_{v\_0}$;
4  for $a$ in $p$ do
5      i $\leftarrow$ index of state after applying $a$;
6      for $eff$ in $Eff^{num}(a)$ do
7          create node $V_{v\_i}$;
8          create node $D_{a\_eff\_v}$ and make it a parent of node $V_{v\_i}$;
9          if $eff$ is an increase effect then
10             make the previous node of $v$ a parent of node $V_{v\_i}$;
11 sample the BPN recursively $N$ times starting from the leaf nodes;
12 for $a$ in $p$ do
13     $counter \leftarrow 0$;
14     for sample run $i$ of $N$ do
15         assign values from sample run $i$ to the variables in $Pre(a)$;
16         if $Pre(a)_{sample\_i}$ holds then
17             $counter++$;
18     if $counter < \theta \times N$ then
19         // If we found one $Pre(a)$ which fails to hold with certainty $\theta$,
           then $p$ is not a valid plan;
20         return $False$
21 // If we reached this point, no $Pre(a)$ failed to hold with certainty $\theta$,
   therefore $p$ is a valid plan;
22 return $True$
```



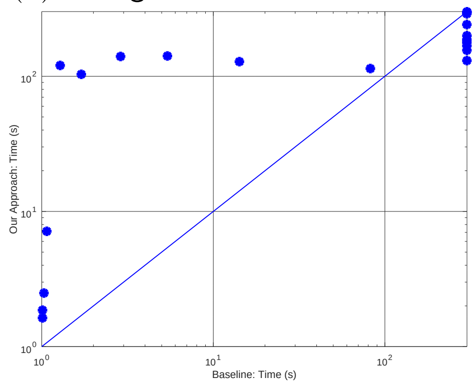
(a) `navigate` action has 2 modes



(b) `navigate` action has 3 modes



(c) `navigate` action has 4 modes



(d) `navigate` action has 5 modes

Figure 6.4: Time taken to build a policy with Gaussians only (X axis) versus with the BPN supporting any distribution (Y axis). Axes are logarithmic.

6.4 Evaluation

The aim of this evaluation is to investigate the computational feasibility of our "hyper-generalised" approach to numeric uncertainty. What if, in the multi-outcome setting of Chapter 4, we also had the ability to represent each outcome as a general distribution rather than a Gaussian distribution? Intuitively, the former would be more computationally intensive than the latter - below we measure this difference, and we check whether any patterns emerge for different problem variants/sizes. Similarly to previous chapters, we measure 1) time elapsed and 2) number of nodes expanded in order to arrive at a solution policy.

The BPN-based policy building technique described in this chapter is implemented on top of the Optic+ expansion described in Chapter 4 (finding strong cyclic policies under numeric uncertainty).

In the baseline planner (the one from Chapter 4), information about uncertainty is hard-coded in the PDDL domain file. Each action enumerates all its non-deterministic outcomes and sets the parameters of their Gaussian distributions.

In the new version of the planner (the one from this chapter), the domain file only specifies which non-deterministic outcomes exist. The probability distribution for each outcome is found in a separate data file containing a collection of samples (or a parametrised function that is later turned into a collection of samples).

Our evaluation uses the Rovers domain (Coles, 2012) where the `navigate` action is non-deterministic - it has multiple outcomes. This is similar to how we measured the feasibility of the (albeit single-outcome) Chapter 5 approach against the Chapter 4 baseline. Like before, this domain was chosen to illustrate our work due to its potential to benefit from increasingly fine-grained representations of uncertainty (see Chapter 5 Section 5.4 for details). A full listing of the domain together with the additional JSON file containing the probability distribution information can be found in Appendix D.

The system we use for experiments has a 1.8GHz Intel Core i7 and 16GB of RAM. Execution time is limited at 300 seconds (5 minutes) and the memory limit is set to 1 GB.

In Figure 6.4 we compare the time taken to find a strong cyclic policy with each of the following approaches:

- Using the numeric constraint representation we defined in Chapter 4 for Gaussian distributed effects.

- Using the BPN representation as shown in the current chapter in order to support arbitrarily distributed effects.

The 4 graphs in Figure 6.4 represent different variants of the domain, as presented in Chapter 5. In each variant, the `navigate` action has a different number of outcomes. Both our baseline and our new approach involve building policies by filling in what to do for each action outcome – therefore we expect there to be no significant difference between the 4 graphs, which is indeed the case according to the experiment data. Exactly half of the data points are located under the bisector line (i.e. solved faster by the new approach) in all 4 graphs. The number of outcomes therefore has no impact on whether or not our approach performs better. The two-tailed Wilcoxon signed-rank test confirms that results are not significant at $P \leq 0.05$.

An interesting though unsurprising observation arises when comparing Figure 6.4 with Figure 5.8 from the previous chapter. For a majority of problems, the Y axis value has increased by a comparable amount – they appear higher on the graph. This result is consistent with the fact that both chapters adopt a sampling-based approach (i.e. use the BPN), and the current chapter (building policies) performs sampling operations more frequently than the previous one (building strong plans).

Another noticeable element in Figure 6.4 is the clustering of problems solved by our current approach and not solved by the baseline: all large problems involving 4 or 5 rovers. These clusters are located in the upper right hand corner of each graph (not counting any points located on the bisector line – there is one such point in each graph). Each cluster contains:

- 8 problems for the 2-outcome case.
- 9 problems for the 3-outcome case.
- 8 problems for the 4-outcome case.
- 9 problems for the 5-outcome case.

In general, problems on the right hand edge of each graph would be a strongly positive indicator of good performance. However in this particular case, they are clustered together near the bisector line – this suggests that the performance gap might be quite narrow, which is consistent with the Wilcoxon test mentioned above.

The section of Figure 6.4 above the bisector line shows that, for small and medium sized problems (involving 2 or 3 rovers), policy building performance is generally better when uncertainty can be represented analytically. It may be the case that the computational advantages provided by Gaussian mathematics have more opportunity to shine in problems of this size. As the amount of reachable states increases, policies may become large enough in both the baseline and in the new approach such that any speed gains offered by Gaussians are negligible.

6.5 Conclusion

In this chapter, our goal was to show that we are able to use a very detailed representation of uncertainty if the problem calls for it. We described an alternative to our prior policy building approach, for cases when uncertainty is not Gaussian. We used a BPN-based representation of uncertainty to allow each action outcome to be drawn from an arbitrary probability distribution. When building the policy, we repeatedly sampled the BPN in order to check whether or not a given state matched any existing policy entry.

Our experimental results showed that, on the illustrative Rovers domain, renouncing our prior Gaussian approximation of uncertainty came at a computational cost. For problems of small and medium size, the approach in Chapter 4 performed better; for large problems, the current approach performed somewhat better, but not by a large margin.

Chapter 7

Conclusion

This chapter sums up the contributions we detailed in the previous chapters, and outlines several possible directions for future research.

7.1 Summary

The purpose of this thesis was, in very broad terms, to allow AI planning software to account for accumulated error due to numeric measurements. We explored the following research question:

Can we improve planner performance by tracking numeric uncertainty and taking it into account at different stages of the planning process?

We found that the answer is **yes**.

The results we obtained from each of our contributions, reiterated below, support our hypothesis that we can indeed improve planner performance by incorporating numeric uncertainty in novel ways.

We introduced means to track uncertainty in a manner that is both computationally feasible and straightforward for the planner to use. In terms of problem modelling, our aim was to be agnostic to the shape of the probability distributions the uncertain effects were drawn from. First we dealt with the particular case of Gaussian distributions, as their analytic form and their additive properties made them especially appealing, and they could also be used to approximate a great deal of real-world effects. Later we introduced a way of representing and tracking arbitrary distributions, to investigate whether such a high level of generality when modelling a problem – as opposed to approximating and simplifying – was worth the additional computational time.

In both of the above cases we either adapted existing elements of the planning process, or introduced novel ones, in order to make use of the newly tracked information. We integrated uncertainty into building strong plans and strong cyclic policies, in cases that did not previously take it into account. The experimental results reiterated here showed how planner performance was indeed improved in three out of four contributions.

Each chapter investigated a different combination of the uncertain factors above, as follows:

	Gaussian distribution	Arbitrary distribution
Single outcome	Chapter 3	Chapter 5
Multiple outcomes	Chapter 4	Chapter 6

7.2 Contributions

Below is a closing summary for each of our main contributions:

- **Contribution 1: Guiding Search Under Fully Observable Gaussian Uncertainty**
  We proposed a novel heuristic based on Metric RPG that takes into account information about uncertain numeric effects. This information made the heuristic more accurate and better able to prune dead ends from the search space when compared to the unmodified Metric RPG heuristic, as shown by our experimental results. In addition, for problems where uncertainty is Gaussian distributed and can be reduced, we introduced a variance tracking technique that allowed the solution extraction phase to strategically use uncertainty-reducing actions.

- **Contribution 2: Building Policies Under Fully Observable Gaussian Uncertainty**
  We defined regression through numeric action effects in order to extend prior work on policy building from the propositional case to the numeric case. Results have shown that by using our method it is possible to

build a policy faster than by naively storing all the numeric information in each state. Further, our definition of regression can accommodate uncertain action outcomes drawn from Gaussian probability distributions. Along similar lines with numeric regression, we also defined numeric dead end generalisation, which was shown experimentally to be successful at pruning away ranges of dead ends.

- **Contribution 3: Guiding Search Under Fully Observable, Arbitrarily Distributed Uncertainty**
  We introduced a representation that allowed us to track and sample uncertain action effects from any probability distribution. Our approach was based on a directed graph capturing how the accumulated uncertainty at any point in a plan depends on previous action effects. We compared the time taken to find strong plans with effects drawn from a single distribution, to the time taken to build strong cyclic policies with effects having several outcomes, each drawn from a Gaussian distribution. In our experiments, the strong plans were found comparatively fast, and occasionally faster than the policies – while having the advantage of not requiring the domain model to approximate all uncertainty as Gaussian.

- **Contribution 4: Building Policies Under Fully Observable, Arbitrarily Distributed Uncertainty**
  We implemented a policy building approach which allowed uncertain numeric outcomes to be sampled from any distributions, as opposed to just Gaussian distributions. We used a sampling-based technique to check whether or not a given state is a match for any existing policy entry. Our experiments compared this general approach to the earlier one which performed numeric regression through Gaussian effects only. Results have shown that policy building performance tends to be better for the prior, Gaussian case, in spite of the increased generality provided by our new representation.

Our work has resulted in the following publications:

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

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

- Marinescu, L., & Coles, A. I. (2018). Representing general numeric uncertainty in non-deterministic forwards planning. In *Proceedings of the Workshop on Heuristics and Search for Domain-Independent Planning at the Twenty-Eighth International Conference on Automated Planning and Scheduling*

7.3 Future Work

Our work opens up automated planning to more possibilities of learning from the environment as execution is happening. 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 uncertainty in 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. In terms of data gathering and function fitting, there are many opportunities to apply techniques from the field of machine learning. An interesting set of question arise from this setting:

- How frequently should we replan – after a certain amount of additional data, or after a tipping point where the additional data changes the median or the modes of the distribution?

- Should newer data be given more weight than original domain information?

- How to distinguish between noise or spikes and a genuine sudden change in the environment?

- Should we use function fitting to determine if new data is consistent, or to proactively extrapolate more data points?

Following on from the idea of gathering information from the environment during execution, we are especially excited about the potential for practical experimentation on physical platforms with our planner. The initial spark that drove us to work on numeric uncertainty in the first place was the development of a quadcopter that could manoeuvre itself through thick foliage with a potentially changing pattern of density. In terms of other possible application domains, 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.

Another interesting feature of our architecture that we have not yet fully explored is enabling the planner to be proactive about improving on its initial solution. Our planner can not only generate a plan with θ certainty, but also 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 BPN that indicates success or failure can be modified to instead report the number of successful sample runs out of the total ones attempted. The opposite of 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.

In the context of policy building, our work can also be easily extended to dynamically generate branches as needed, instead of relying on branches specified in the domain. One possible way to achieve this is by successively bisecting the probability distribution of a single-outcome non-deterministic effect. Whether to generate a branch or not will depend on the success or failure of the weak plan from that branch outcome to the goal, according to the BPN. Another possible alternative to introduce branches would be to compute the bounds within which the original weak plan is valid, and treat the regions outside those bounds as separate branches.

Bibliography

Albore, A., Palacios, H., & Geffner, H. (2009). A translation-based approach to contingent planning. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence.*

Anderson, C. R., Smith, D. E., & Weld, D. S. (1998). Conditional effects in Graphplan. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems.*

Babaki, B., Guns, T., & 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.*

Bacchus, F. (2001). AIPS 2000 planning competition: The fifth international conference on artificial intelligence planning and scheduling systems. *AI Magazine.*

Bäckström, C., & Nebel, B. (1995). Complexity results for SAS+ planning. *Computational Intelligence.*

Beaudry, E., Kabanza, F., & Michaud, F. (2010a). Planning for concurrent action executions under action duration uncertainty using dynamically generated Bayesian networks. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling.*

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

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

Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence.*

Blum, A. L., & Langford, J. C. (1999). Probabilistic planning in the Graphplan framework. In *Proceedings of the Fifth European Conference on Planning.*

Bonet, B., & Geffner, H. (2005). mGPT: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research.*

Brafman, R. I., & Hoffmann, J. (2004). Conformant planning via heuristic forward search: A new approach. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling.*

Camacho, A., Muise, C., & McIlraith, S. (2016). From FOND to robust probabilistic planning: Computing compact policies that bypass avoidable deadends. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling.*

Cimatti, A., Pistore, M., Roveri, M., & Traverso, P. (2001). Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence.*

Cimatti, A., Roveri, M., & Traverso, P. (1998). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proceedings of the Fifteenth AAAI Conference on Artificial Intelligence.*

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.*

Daniele, M., Traverso, P., & Vardi, M. Y. (1999). Strong cyclic planning revisited. *Recent Advances in Artificial Intelligence Planning.*

Domshlak, C., & Hoffmann, J. (2006). Fast probabilistic planning through weighted model counting. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling.*

Domshlak, C., & Hoffmann, J. (2007). Probabilistic planning via heuristic forward search and weighted model counting. *Journal of Artificial Intelligence Research.*

Feng, Z., Dearden, R., Meuleau, N., & Washington, R. (2004). Dynamic programming for structured continuous Markov decision problems. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence.*

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the Second International Joint Conference on Artificial Intelligence.*

Fox, M., & Long, D. (2002). PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research.*

Fu, J., Ng, V., Bastani, F. B., & Yen, I. L. (2011). Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence.*

Geffner, H., & Bonet, B. (2013). *A concise introduction to models and methods for automated planning.* Morgan & Claypool.

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

Gerevini, A., Saetti, A., & Serina, I. (2003). Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research.*

Gerevini, A., Saetti, A., & Serina, I. (2004). Planning with numerical expressions in LPG. In *Proceedings of the Sixteenth European Conference on Artificial Intelligence.*

Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research.*

Hendler, J. A., Tate, A., & Drummond, M. (1990). AI planning: Systems and techniques. *AI Magazine.*

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

Hoffmann, J., & Brafman, R. I. (2005). Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*.

Hoffmann, J., & Brafman, R. I. (2006). Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*.

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

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*.

Keller, T., & Eyerich, P. (2012). PROST: Probabilistic planning based on UCT. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*.

Keyder, E., & Geffner, H. (2008). The HMDP planner for planning with probabilities. In *Proceedings of the Sixth International Planning Competition*.

Kissmann, P., & Edelkamp, S. (2009). Solving fully-observable non-deterministic planning problems via translation into a general game. In *Proceedings of the Thirty-Second Annual German Conference on Artificial Intelligence*.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Proceedings of the Seventeenth European Conference on Machine Learning*.

Little, I., Aberdeen, D., & Thiébaux, S. (2005). Prottle: A probabilistic temporal planner. In *Proceedings of the Twentieth AAAI Conference on Artificial Intelligence*.

Little, I., & Thiébaux, S. (2007). Probabilistic planning vs replanning. In *Proceedings of the Workshop on the International Planning Competition: Past, Present and Future at the Fourteenth International Conference on Automated Planning and Scheduling*.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*.

Long, D., & Fox, M. (2003). The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*.

Luo, Y., Bai, H., Hsu, D., & Lee, W. S. (2019). Importance sampling for online planning under uncertainty. *International Journal of Robotics Research*.

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

Marinescu, L., & Coles, A. I. (2016c). Non-deterministic planning with numeric uncertainty. Tech. rep., King's College London.

Marinescu, L., & Coles, A. I. (2018). Representing general numeric uncertainty in non-deterministic forwards planning. In *Proceedings of the Workshop on Heuristics and Search for Domain-Independent Planning at the Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Mausam, M., & Weld, D. S. (2008). Planning with durative actions in stochastic domains. *Journal of Artificial Intelligence Research*.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL - The planning domain definition language. Tech. rep., Yale Center for Computational Vision and Control.

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

Muise, C. (2014). *Exploiting relevance to improve robustness and flexibility in plan generation and execution.* Ph.D. thesis, University of Toronto.

Muise, C., McIlraith, S. A., & Beck, J. C. (2011). Monitoring the execution of partial-order plans via regression. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence.*

Muise, C. J., Belle, V., & 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., & 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.*

Muise, C. J., McIlraith, S. A., & Belle, V. (2014). Non-deterministic planning with conditional effects. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling.*

Palacios, H., Bonet, B., Darwiche, A., & Geffner, H. (2005). Pruning conformant plans by counting models on compiled d-DNNF representations. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling.*

Palacios, H., & Geffner, H. (2007). From conformant into classical planning: Efficient translations that may be complete too. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling.*

Palacios, H., & Geffner, H. (2009). Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research.*

Rachelson, E., Quesnel, G., Garcia, F., & Fabiani, P. (2008). A simulation-based approach for solving temporal Markov problems. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence.*

Russell, S. J., & Norvig, P. (2010). *Artificial intelligence: A modern approach.* Prentice Hall.

Sanner, S. P. (2010). Relational dynamic influence diagram language (RDDL): Language description. Australian National University.

Santana, P., Thiébaux, S., & Williams, B. (2016). RAO*: An algorithm for chance constrained POMDPs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence.*

Schoppers, M. J. (1997). Universal plans for reactive robots in unpredictable environments. In *Proceedings of the Tenth International Joint conference on Artificial Intelligence.*

Smith, D. E., & Weld, D. S. (1998). Conformant Graphplan. In *Proceedings of the Fifteenth AAAI Conference on Artificial Intelligence.*

Thiébaux, S., & Cordier, M. O. (2001). Supply restoration in power distribution systems – a benchmark for planning under uncertainty. In *Proceedings of the Sixth European Conference on Planning.*

Vallati, M., Chrpa, L., Grześ, M., McCluskey, T. L., Roberts, M., & Sanner, S. (2015). The 2014 international planning competition: Progress and trends. *AI Magazine.*

Weld, D. S., Anderson, C. R., & Smith, D. E. (1998). Extending Graphplan to handle uncertainty & sensing actions. In *Proceedings of the Fifteenth AAAI Conference on Artificial Intelligence.*

Yoon, S., Fern, A., & Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling.*

Yoon, S., Ruml, W., Benton, J., & Do, M. B. (2010). Improving determinization in hindsight for online probabilistic planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling.*

Younes, H. L. S., & Simmons, R. G. (2004). Policy generation for continuous-time stochastic domains with concurrency. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling.*

Younes, H. S. L., & Littman, M. L. (2004). PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Tech. rep., Carnegie Mellon University.

Appendix A: Domains Used in Chapter 3

Rovers - single outcome, Gaussian uncertainty

```
(define (domain Rover)
(:requirements :typing :fluents)
(:types rover waypoint store camera mode lander
   objective)

(:predicates (at ?x - rover ?y - waypoint)
             (at_lander ?x - lander ?y - waypoint)
             (can_traverse ?r - rover ?x - waypoint ?y
                - waypoint)
             (equipped_for_soil_analysis ?r - rover)
             (equipped_for_rock_analysis ?r - rover)
             (equipped_for_imaging ?r - rover)
             (empty ?s - store)
             (have_rock_analysis ?r - rover ?w -
                waypoint)
             (have_soil_analysis ?r - rover ?w -
                waypoint)
             (full ?s - store)
             (calibrated ?c - camera ?r - rover)
             (supports ?c - camera ?m - mode)
             (available ?r - rover)
             (visible ?w - waypoint ?p - waypoint)
             (have_image ?r - rover ?o - objective ?m -
                 mode)
             (communicated_soil_data ?w - waypoint)
             (communicated_rock_data ?w - waypoint)
             (communicated_image_data ?o - objective ?m
                 - mode)
             (at_soil_sample ?w - waypoint)
             (at_rock_sample ?w - waypoint)
             (visible_from ?o - objective ?w - waypoint
                )
             (store_of ?s - store ?r - rover)
             (calibration_target ?i - camera ?o -
                objective)
             (on_board ?i - camera ?r - rover)
             (channel_free ?l - lander)
             (in_sun ?w - waypoint)

)

(:functions       (energy ?r - rover)
                  (energy-variance ?r - rover) ; This was
                       introduced first to test the
                     canSatisfy function with confidence.
                  (position-error ?r - rover) ; This was
                     introduced second to allow for error
                       accumulation that can be fixed with
                       look-at-stars.
)

(:action navigate
:parameters (?x - rover ?y - waypoint ?z - waypoint)
:precondition (and (can_traverse ?x ?y ?z) (available ?
   x) (at ?x ?y)
                  (visible ?y ?z) (>= (energy ?x) 8)
             )
:effect (and (increase (position-error ?x) 1) (decrease
    (energy ?x) 8) (increase (energy-variance ?x) 8) (
   not (at ?x ?y)) (at ?x ?z)
                  )
```

```
)

(:action recharge
:parameters (?x - rover ?w - waypoint)
:precondition (and (at ?x ?w) (in_sun ?w))
:effect (and (increase (energy ?x) 20) (assign (
   energy-variance ?x) 0)) ; Recharging is
   deterministic.
)

(:action look-at-stars
:parameters (?x - rover)
:precondition (>= (position-error ?x) 1)
:effect (and
                (when (>= (position-error ?x) 5) (
                   decrease (position-error ?x) 5))
                (when (<= (position-error ?x) 4) (
                   assign (position-error ?x) 0))
        )
)

(:action sample_soil
:parameters (?x - rover ?s - store ?p - waypoint)
:precondition (and (at ?x ?p)(>= (energy ?x) 2) (
   at_soil_sample ?p) (equipped_for_soil_analysis ?x) (
   store_of ?s ?x) (empty ?s) (<= (position-error ?x)
   2)
                )
:effect (and (not (empty ?s)) (full ?s) (decrease (
   energy ?x) 2)  (increase (energy-variance ?x) 2) (
   have_soil_analysis ?x ?p) (not (at_soil_sample ?p))
                )
)

(:action sample_rock
:parameters (?x - rover ?s - store ?p - waypoint)
:precondition (and (at ?x ?p)(>= (energy ?x) 4) (
   at_rock_sample ?p) (equipped_for_rock_analysis ?x) (
   store_of ?s ?x) (empty ?s) (<= (position-error ?x)
   1)
               )
:effect (and (not (empty ?s)) (full ?s) (decrease (
   energy ?x) 4)  (increase (energy-variance ?x) 4) (
   have_rock_analysis ?x ?p) (not (at_rock_sample ?p))
               )
)

(:action drop
:parameters (?x - rover ?y - store)
:precondition (and (store_of ?y ?x) (full ?y)
               )
:effect (and (not (full ?y)) (empty ?y)
        )
)

(:action calibrate
 :parameters (?r - rover ?i - camera ?t - objective ?w
    - waypoint)
 :precondition (and (equipped_for_imaging ?r) (>= (
    energy ?r) 2)(calibration_target ?i ?t) (at ?r ?w)
    (visible_from ?t ?w)(on_board ?i ?r)
               )
 :effect (and (decrease (energy ?r) 2)  (increase (
    energy-variance ?r) 2)(calibrated ?i ?r) )
)

(:action take_image
 :parameters (?r - rover ?p - waypoint ?o - objective ?
    i - camera ?m - mode)
 :precondition (and (calibrated ?i ?r)
                         (on_board ?i ?r)
                      (equipped_for_imaging ?r)
                      (supports ?i ?m)
                          (visible_from ?o ?p)
                     (at ?r ?p)
                        (>= (energy ?r) 1)
```

```
                )
 :effect (and (have_image ?r ?o ?m)(not (calibrated ?i
    ?r))(decrease (energy ?r) 1)  (increase (
    energy-variance ?r) 1)
                 )
)

(:action communicate_soil_data
 :parameters (?r - rover ?l - lander ?p - waypoint ?x -
     waypoint ?y - waypoint)
 :precondition (and (at ?r ?x)(at_lander ?l ?y)(
    have_soil_analysis ?r ?p)
                   (visible ?x ?y)(available ?r)(
                      channel_free ?l)(>= (energy ?r)
                      4)
             )
 :effect (and (not (available ?r))(not (channel_free ?l
    ))
(channel_free ?l) (communicated_soil_data ?p)(available
    ?r)(decrease (energy ?r) 4)  (increase (
   energy-variance ?r) 4)
        )
)

(:action communicate_rock_data
 :parameters (?r - rover ?l - lander ?p - waypoint ?x -
     waypoint ?y - waypoint)
 :precondition (and (at ?r ?x)(at_lander ?l ?y)(
    have_rock_analysis ?r ?p)(>= (energy ?r) 4)
                   (visible ?x ?y)(available ?r)(
                      channel_free ?l)
             )
 :effect (and   (not (available ?r))(not (channel_free
    ?l))
(channel_free ?l)
(communicated_rock_data ?p)(available ?r)(decrease (
   energy ?r) 4)  (increase (energy-variance ?r) 4)
          )
```

```
)

(:action communicate_image_data
 :parameters (?r - rover ?l - lander ?o - objective ?m
    - mode ?x - waypoint ?y - waypoint)
 :precondition (and (at ?r ?x)(at_lander ?l ?y)(
    have_image ?r ?o ?m)(visible ?x ?y)(available ?r)(
    channel_free ?l)(>= (energy ?r) 6)
            )
 :effect (and (not (available ?r))(not (channel_free ?l
    ))
(channel_free ?l)
(communicated_image_data ?o ?m)(available ?r)(decrease
   (energy ?r) 6)  (increase (energy-variance ?r) 6)
           )
)

)
```

TPP - single outcome, Gaussian uncertainty

```
(define (domain TPP-Metric)
(:requirements :typing :fluents)
(:types depot market - place
        truck goods khajiit - locatable)

(:predicates (at ?t - locatable ?p - place) (likes ?k -
    khajiit ?g - goods))

(:functions
        (on-sale ?g - goods ?m - market)
        (on-sale-variance ?g - goods ?m - market) ; How
            unsure the vendor is of his quantity.
            (drive-cost ?p1 ?p2 - place)
            (price ?g - goods ?m - market)
            (bought ?g - goods)
        (bought-variance ?g - goods) ; How unsure the
           buyer is of how much he's bought so far.
            (request ?g - goods)
            (total-cost))

(:action drive
 :parameters (?t - truck ?from ?to - place)
 :precondition (and (at ?t ?from))
 :effect (and (not (at ?t ?from)) (at ?t ?to)
              (increase (total-cost) (drive-cost ?from
                 ?to)))
 )

(:action buy-allneeded
 :parameters (?t - truck ?g - goods ?m - market)
 :precondition (and (at ?t ?m) (> (on-sale ?g ?m) 0) (>
     (request ?g) 0)
                   (> (on-sale ?g ?m) (- (request ?g)
                      (bought ?g))))
 :effect (and
          (assign (on-sale ?g ?m) 0)
          (decrease (on-sale ?g ?m) (- (request ?g) (
             bought ?g)))
              (increase (total-cost) (* (- (request ?g)
                  (bought ?g))
                                      (price ?g ?m)))
              (assign (bought ?g) (+ (request ?g) 9000)
                 ))
 )

(:action buy-all
 :parameters (?t - truck ?g - goods ?m - market)
 :precondition (and (at ?t ?m) (> (on-sale ?g ?m) 0) (>
     (request ?g) 0)
                   )
 :effect (and (assign (on-sale ?g ?m) 0)
             (increase (total-cost) (* (on-sale ?g ?m)
                 (price ?g ?m)))
             (increase (bought ?g) (on-sale ?g ?m))
             (increase (bought-variance ?g) (
                on-sale-variance ?g ?m))
             )
 )

(:action khajiit-has-wares ; Magical action that weighs
    your goods and adjusts their quantity to perfection
   . Elder Scrolls FTW!
    :parameters (?t - truck ?g - goods ?p - place ?k -
       khajiit)
    :precondition (and (at ?t ?p) (at ?k ?p) (likes ?k
       ?g) (>= (bought ?g) 1)) ; Do not bother Khajiit
       if you haven't bought your goods yet!
    :effect (and (assign (bought-variance ?g) 0) (not (
       at ?t ?p)) (at ?t ?p))
)

)
```

AUV - single outcome, Gaussian uncertainty

```
(define (domain auv)
  (:requirements :strips :typing :fluents :preferences)
  (:types auv waypoint target ship)

  (:predicates
   (at ?v - auv ?w - waypoint)
   (at_ship ?s - ship ?w - waypoint)
   (can_traverse ?w1 - waypoint ?w2 - waypoint)
   (have_image ?t - target)
   (communicated_photo_data ?t - target)
```

```
 (communicated_survey_data ?t - target)
 (target-at ?t - target ?w - waypoint)
 (target-starts ?t - target ?w - waypoint)
 (target-ends ?t - target ?w - waypoint)
 (visible_from ?t - target ?w - waypoint)
 (at_surface ?w - waypoint)
 (turned ?v - target)
 (available)
 (need_cleaned ?t - target)
 (need_photo_data  ?t - target)
 (need_survey_data ?t - target)
 (need_turned  ?t - target)
 )

(:functions
 (traverse_time ?w1 ?w2 - waypoint)
 (time-remaining)
 (time-remaining-variance)
 (available-memory)
 (cleaning_cost ?t - target)
 (amount_cleaned ?t - target)
 (amount_cleaned-variance ?t - target)
 (have_some_survey_data_chain ?t - target)
 (have_some_survey_data_chain-variance ?t - target)
 )


(:action navigate
  :parameters (?v - auv ?y - waypoint ?z - waypoint)
  :precondition (and (can_traverse ?y ?z)
                     (at ?v ?y)
                     (available)
                     (>= (time-remaining) (
                        traverse_time ?y ?z))
                )
  :effect (and (not (at ?v ?y))
               (at ?v ?z)
               (not (available))
```

```
                (available)
                (increase (time-remaining-variance) (*
                    (* 0.3 (traverse_time ?y ?z)) (*
                   0.3 (traverse_time ?y ?z))) )
                (decrease (time-remaining) (
                   traverse_time ?y ?z)))
  )

(:action clean-long-target
  :parameters (?v - auv ?t - target ?w1 ?w2 -
     waypoint)
  :precondition (and (at ?v ?w1)
              (or ; Made the AUV able to start
                 cleaning from either side of a target
                 .
                   (and
                       (target-starts ?t ?w1)
                       (target-ends ?t ?w2)
                   )
                   (and
                       (target-starts ?t ?w2)
                       (target-ends ?t ?w1)
                   )
              )
                      (available)
                      (need_cleaned ?t)
                      (>= (time-remaining) (*5 (
                         traverse_time ?w1 ?w2)))

                 )
  :effect (and (not (at ?v ?w1))
        (increase (amount_cleaned ?t) 1) ; Made the
           AUV clean incrementally - it needs at least
            0.9 cleanliness (set in the goal).
        (increase (amount_cleaned-variance ?t) 0.1)
                (at ?v ?w2)
                (not (available))
                (available)
                (increase (time-remaining-variance) (*
                    (* 4 (traverse_time ?w1 ?w2)) (* 4
                    (traverse_time ?w1 ?w2))) )
                (decrease (time-remaining) (*5 (
                   traverse_time ?w1 ?w2)))
           )
)

(:action video-survey-long-target
  :parameters (?v - auv ?t - target ?w1 ?w2 -
     waypoint)
  :precondition (and (at ?v ?w1)
               (or ; Made the AUV able to start a
                  survey from either side of a target.
                    (and
                        (target-starts ?t ?w1)
                        (target-ends ?t ?w2)
                    )
                    (and
                        (target-starts ?t ?w2)
                        (target-ends ?t ?w1)
                    )
                       )
                       (available)
                       (need_survey_data ?t)
                       (>= (time-remaining) (*2 (
                          traverse_time ?w1 ?w2)))

                 )
  :effect (and (not (at ?v ?w1))
                (at ?v ?w2)
                (increase (have_some_survey_data_chain
                    ?t) 1) ; Made the AUV get survey
                   data incrementally - it needs at
                   least 0.9 to send the data.
                (increase (
                   have_some_survey_data_chain-variance
                    ?t) 0.1)
                (not (available))
                (available)
                (increase (time-remaining-variance) (*
                    (* 0.7 (traverse_time ?w1 ?w2)) (*
                    0.7 (traverse_time ?w1 ?w2))) )
                (decrease (time-remaining) (*2 (
                   traverse_time ?w1 ?w2)))
           )
)

 (:action turn-valve
  :parameters (?v - auv ?valve - target ?w1 -
     waypoint)
  :precondition (and (at ?v ?w1)
                     (target-at ?valve ?w1)
                     (available)
                     (need_turned ?valve)
                     (>= (time-remaining) 10)


                )
  :effect (and (turned ?valve)
               (not (need_turned ?valve))
               (increase (time-remaining-variance) (*
                   (* 0.2 10) (* 0.2 10)) )
               (decrease (time-remaining) 10)
               (not (available))
               (available)
          )
)

(:action photograph_valve
  :parameters (?v - auv ?w - waypoint ?t - target)
  :precondition (and (target-at ?t ?w)
                     (available)
                     (need_photo_data ?t)
                     (>= (time-remaining) 1)
```

```
                        (at ?v ?w))
    :effect (and (have_image ?t)
                 (not (available))
                 (not (need_photo_data ?t))
                 (available)
                 (decrease (time-remaining) 1)
                 )
    )

  (:action communicate_photo_data
    :parameters (?v - auv ?t - target ?w - waypoint)
    :precondition (and (at ?v ?w)
                        (available)
                        (have_image ?t))
    :effect (and (communicated_photo_data ?t)
                 (not (have_image ?t))
                 (not (available))
                 (available)
            )
    )

  (:action communicate_survey_data
    :parameters (?v - auv ?t - target ?w - waypoint)
    :precondition (and (at ?v ?w)
                        (available)
                        (>= (have_some_survey_data_chain
                            ?t) 0.9))
    :effect (and (communicated_survey_data ?t)
                 (assign (have_some_survey_data_chain ?
                    t) 0)
                 (not (available))
                 (available)
            )
    )
)
```

Appendix B: Domains Used in Chapter 4

Rovers - multiple outcomes, Gaussian uncertainty

```
(define (domain Rover)
(:requirements :typing :fluents)
(:types rover waypoint store camera mode lander
   objective)

(:predicates (at ?x - rover ?y - waypoint)
             (at_lander ?x - lander ?y - waypoint)
             (can_traverse ?r - rover ?x - waypoint ?y
                - waypoint)
             (equipped_for_soil_analysis ?r - rover)
             (equipped_for_rock_analysis ?r - rover)
             (equipped_for_imaging ?r - rover)
             (empty ?s - store)
             (have_rock_analysis ?r - rover ?w -
                waypoint)
             (have_soil_analysis ?r - rover ?w -
                waypoint)
             (full ?s - store)
             (calibrated ?c - camera ?r - rover)
             (supports ?c - camera ?m - mode)
             (available ?r - rover)
```

```
                (visible ?w - waypoint ?p - waypoint)
                (have_image ?r - rover ?o - objective ?m -
                    mode)
                (communicated_soil_data ?w - waypoint)
                (communicated_rock_data ?w - waypoint)
                (communicated_image_data ?o - objective ?m
                    - mode)
                (at_soil_sample ?w - waypoint)
                (at_rock_sample ?w - waypoint)
                (visible_from ?o - objective ?w - waypoint
                   )
                (store_of ?s - store ?r - rover)
                (calibration_target ?i - camera ?o -
                   objective)
                (on_board ?i - camera ?r - rover)
                (channel_free ?l - lander)
                (in_sun ?w - waypoint)

)

(:functions      (energy ?r - rover)
                 (energy-variance ?r - rover) ; This was
                      introduced first to test the
                    canSatisfy function with confidence.
                 (position-error ?r - rover) ; This was
                    introduced second to allow for error
                      accumulation that can be fixed with
                      look-at-stars.
)

(:action navigate-outcome1
:parameters (?x - rover ?y - waypoint ?z - waypoint)
:precondition (and (can_traverse ?x ?y ?z) (available ?
   x) (at ?x ?y)
                 (visible ?y ?z) (>= (energy ?x) 10)
            )
:effect (and (increase (position-error ?x) 1) (decrease
    (energy ?x) 8)  (increase (energy-variance ?x) 8) (
   not (at ?x ?y)) (at ?x ?z)
                )
)

(:action navigate-outcome2
:parameters (?x - rover ?y - waypoint ?z - waypoint)
:precondition (and (can_traverse ?x ?y ?z) (available ?
   x) (at ?x ?y)
                (visible ?y ?z) (>= (energy ?x) 10)
            )
:effect (and (increase (position-error ?x) 1) (decrease
    (energy ?x) 10)  (increase (energy-variance ?x) 10)
    (not (at ?x ?y)) (at ?x ?z)
                )
)

(:action navigate-outcome3
:parameters (?x - rover ?y - waypoint ?z - waypoint)
:precondition (and (can_traverse ?x ?y ?z) (available ?
   x) (at ?x ?y)
                (visible ?y ?z) (>= (energy ?x) 10)
            )
:effect (and (increase (position-error ?x) 1) (decrease
    (energy ?x) 12)  (increase (energy-variance ?x) 12)
    (not (at ?x ?y)) (at ?x ?z)
                )
)

(:action recharge
:parameters (?x - rover ?w - waypoint)
:precondition (and (at ?x ?w) (in_sun ?w))
:effect (and (increase (energy ?x) 20) (assign (
   energy-variance ?x) 0)) ; Recharging is
   deterministic.
)

(:action look-at-stars-a-bit
:parameters (?x - rover)
```

```
:precondition (>= (position-error ?x) 5)
:effect (and
                (decrease (position-error ?x) 5)
        )
)

(:action look-at-stars-a-bit-more
:parameters (?x - rover)
:precondition (<= (position-error ?x) 4)
:effect (and
                (assign (position-error ?x) 0)
        )
)

; Removed action variants, added a condition for
   position-error <= 2.
(:action sample_soil
:parameters (?x - rover ?s - store ?p - waypoint)
:precondition (and (at ?x ?p)(>= (energy ?x) 2) (
   at_soil_sample ?p) (equipped_for_soil_analysis ?x) (
   store_of ?s ?x) (empty ?s) (<= (position-error ?x)
   2)
                )
:effect (and (not (empty ?s)) (full ?s) (decrease (
   energy ?x) 2)  (increase (energy-variance ?x) 2) (
   have_soil_analysis ?x ?p) (not (at_soil_sample ?p))
                )
)

; Removed action variants, added a condition for
   position-error <= 1.
(:action sample_rock
:parameters (?x - rover ?s - store ?p - waypoint)
:precondition (and (at ?x ?p)(>= (energy ?x) 4) (
   at_rock_sample ?p) (equipped_for_rock_analysis ?x) (
   store_of ?s ?x) (empty ?s) (<= (position-error ?x)
   1)
                )
```

```
:effect (and (not (empty ?s)) (full ?s) (decrease (
   energy ?x) 4)  (increase (energy-variance ?x) 4) (
   have_rock_analysis ?x ?p) (not (at_rock_sample ?p))
                )
)

(:action drop
:parameters (?x - rover ?y - store)
:precondition (and (store_of ?y ?x) (full ?y)
                )
:effect (and (not (full ?y)) (empty ?y)
        )
)

(:action calibrate
 :parameters (?r - rover ?i - camera ?t - objective ?w
    - waypoint)
 :precondition (and (equipped_for_imaging ?r) (>= (
    energy ?r) 2)(calibration_target ?i ?t) (at ?r ?w)
    (visible_from ?t ?w)(on_board ?i ?r)
                )
 :effect (and (decrease (energy ?r) 2)  (increase (
    energy-variance ?r) 2)(calibrated ?i ?r) )
)

(:action take_image
 :parameters (?r - rover ?p - waypoint ?o - objective ?
    i - camera ?m - mode)
 :precondition (and (calibrated ?i ?r)
                         (on_board ?i ?r)
                      (equipped_for_imaging ?r)
                      (supports ?i ?m)
                          (visible_from ?o ?p)
                     (at ?r ?p)
                        (>= (energy ?r) 1)
               )
 :effect (and (have_image ?r ?o ?m)(not (calibrated ?i
    ?r))(decrease (energy ?r) 1)  (increase (
    energy-variance ?r) 1)
                )
)

(:action communicate_soil_data
 :parameters (?r - rover ?l - lander ?p - waypoint ?x -
     waypoint ?y - waypoint)
 :precondition (and (at ?r ?x)(at_lander ?l ?y)(
    have_soil_analysis ?r ?p)
                  (visible ?x ?y)(available ?r)(
                     channel_free ?l)(>= (energy ?r)
                     4)
            )
 :effect (and (not (available ?r))(not (channel_free ?l
    ))
(channel_free ?l) (communicated_soil_data ?p)(available
    ?r)(decrease (energy ?r) 4)  (increase (
   energy-variance ?r) 4)
        )
)

(:action communicate_rock_data
 :parameters (?r - rover ?l - lander ?p - waypoint ?x -
     waypoint ?y - waypoint)
 :precondition (and (at ?r ?x)(at_lander ?l ?y)(
    have_rock_analysis ?r ?p)(>= (energy ?r) 4)
                  (visible ?x ?y)(available ?r)(
                     channel_free ?l)
            )
 :effect (and   (not (available ?r))(not (channel_free
    ?l))
(channel_free ?l)
(communicated_rock_data ?p)(available ?r)(decrease (
   energy ?r) 4)  (increase (energy-variance ?r) 4)
          )
)

(:action communicate_image_data
 :parameters (?r - rover ?l - lander ?o - objective ?m
    - mode ?x - waypoint ?y - waypoint)
 :precondition (and (at ?r ?x)(at_lander ?l ?y)(
    have_image ?r ?o ?m)(visible ?x ?y)(available ?r)(
    channel_free ?l)(>= (energy ?r) 6)
            )
 :effect (and (not (available ?r))(not (channel_free ?l
    ))
(channel_free ?l)
(communicated_image_data ?o ?m)(available ?r)(decrease
   (energy ?r) 6)  (increase (energy-variance ?r) 6)
           )
)

)
```

TPP - multiple outcomes, Gaussian uncertainty

```
(define (domain TPP-Metric)
(:requirements :typing :fluents)
(:types depot market - place
        truck goods khajiit - locatable)

(:predicates (at ?t - locatable ?p - place) (likes ?k -
    khajiit ?g - goods)
             (tyres-fine ?t - truck)
             (flat-tyre ?t - truck)
)

(:functions
        (on-sale ?g - goods ?m - market)
        (on-sale-variance ?g - goods ?m - market) ; How
            unsure the vendor is of his quantity.
            (drive-cost ?p1 ?p2 - place)
```

```
                (price ?g - goods ?m - market)
                (bought ?g - goods)
            (bought-variance ?g - goods) ; How unsure the
               buyer is of how much he's bought so far.
                (request ?g - goods)
                (total-cost))

(:action drive-outcome1
 :parameters (?t - truck ?from ?to - place)
 :precondition (and (at ?t ?from) (tyres-fine ?t))
 :effect (and (not (at ?t ?from)) (at ?t ?to)
              (increase (total-cost) (drive-cost ?from
                 ?to)))
 )

(:action drive-outcome2
 :parameters (?t - truck ?from ?to - place)
 :precondition (and (at ?t ?from) (tyres-fine ?t))
 :effect (and (not (tyres-fine ?t))
              (flat-tyre ?t)
              (not (at ?t ?from)) (at ?t ?to)
              (increase (total-cost) (drive-cost ?from
                 ?to)))
 )


(:action fix-tyre
 :parameters (?t - truck)
 :precondition (flat-tyre ?t)
 :effect (and (not (flat-tyre ?t))
              (tyres-fine ?t)
              (increase (total-cost) 10)
         )
)

(:action buy-allneeded
 :parameters (?t - truck ?g - goods ?m - market)
 :precondition (and (at ?t ?m) (> (on-sale ?g ?m) 0) (>
    (request ?g) 0)
                  (> (on−sale ?g ?m) (− (request ?g)
                     (bought ?g))))
 :effect (and
          (assign (on−sale ?g ?m) 0)
          (decrease (on−sale ?g ?m) (− (request ?g) (
             bought ?g)))
              (increase (total−cost) (∗ (− (request ?g)
                  (bought ?g))
                                       (price ?g ?m)))
              (assign (bought ?g) (+ (request ?g) 9000)
                 ))
 )

(:action buy−all
 :parameters (?t − truck ?g − goods ?m − market)
 :precondition (and (at ?t ?m) (> (on−sale ?g ?m) 0) (>
    (request ?g) 0)
                  )
 :effect (and (assign (on−sale ?g ?m) 0)
              (increase (total−cost) (∗ (on−sale ?g ?m)
                  (price ?g ?m)))
              (increase (bought ?g) (on−sale ?g ?m))
              (increase (bought−variance ?g) (
                 on−sale−variance ?g ?m))
              )
 )

(:action khajiit−has−wares ; Magical action that weighs
    your goods and adjusts their quantity to perfection
   . Elder Scrolls FTW!
    :parameters (?t − truck ?g − goods ?p − place ?k −
       khajiit)
    :precondition (and (at ?t ?p) (at ?k ?p) (likes ?k
       ?g) (>= (bought ?g) 1)) ; Do not bother Khajiit
       if you haven't bought your goods yet!
    :effect (and (assign (bought−variance ?g) 0) (not (
       at ?t ?p)) (at ?t ?p))
```

```
)

)
```

AUV - multiple outcomes, Gaussian uncertainty

```
(define (domain auv)
  (:requirements :strips :typing :fluents :preferences)
  (:types auv waypoint target ship)

  (:predicates
   (at ?v - auv ?w - waypoint)
   (at_ship ?s - ship ?w - waypoint)
   (can_traverse ?w1 - waypoint ?w2 - waypoint)
   (have_image ?t - target)
   (communicated_photo_data ?t - target)
   (communicated_survey_data ?t - target)
   (target-at ?t - target ?w - waypoint)
   (target-starts ?t - target ?w - waypoint)
   (target-ends ?t - target ?w - waypoint)
   (visible_from ?t - target ?w - waypoint)
   (at_surface ?w - waypoint)
   (turned ?v - target)
   (available)
   (need_cleaned ?t - target)
   (need_photo_data  ?t - target)
   (need_survey_data ?t - target)
   (need_turned  ?t - target)
   (stuck ?t - target)
   (stuck-at ?v - auv ?w - waypoint)
   )

  (:functions
   (traverse_time ?w1 ?w2 - waypoint)
```

```
 (time-remaining)
 (time-remaining-variance)
 (available-memory)
 (cleaning_cost ?t - target)
 (amount_cleaned ?t - target)
 (amount_cleaned-variance ?t - target)
 (have_some_survey_data_chain ?t - target)
 (have_some_survey_data_chain-variance ?t - target)
 (unsatisfied-goals)
 )


(:event cleaned-enough
 :parameters (?t - target)
 :precondition (and (need_cleaned ?t)
                    (>= (time-remaining) 0)
                    (>= (amount_cleaned ?t) 0.9)

 )
 :effect (and (not (need_cleaned ?t))
              (decrease (unsatisfied-goals) 1)

 )

)

(:action navigate
  :parameters (?v - auv ?y - waypoint ?z - waypoint)
  :precondition (and (can_traverse ?y ?z)
                     (at ?v ?y)
                     (available)
                     (>= (time-remaining) (
                        traverse_time ?y ?z))
                )
  :effect (and (not (at ?v ?y))
               (at ?v ?z)
               (not (available))
               (available)
```

```
                (increase (time-remaining-variance) (*
                   (* 0.3 (traverse_time ?y ?z)) (*
                  0.3 (traverse_time ?y ?z))) )
                (decrease (time-remaining) (
                  traverse_time ?y ?z)))
    )

  (:action clean-long-target
    :parameters (?v - auv ?t - target ?w1 ?w2 -
       waypoint)
    :precondition (and (at ?v ?w1)
               (or ; Made the AUV able to start
                  cleaning from either side of a target
                  .
                    (and
                        (target-starts ?t ?w1)
                        (target-ends ?t ?w2)
                    )
                    (and
                        (target-starts ?t ?w2)
                        (target-ends ?t ?w1)
                    )
               )
                       (available)
                       (need_cleaned ?t)
                       (>= (time-remaining) (*5 (
                          traverse_time ?w1 ?w2)))

                  )
    :effect (and (not (at ?v ?w1))
         (increase (amount_cleaned ?t) 1) ; Made the
            AUV clean incrementally - it needs at least
             0.9 cleanliness (set in the goal).
         (increase (amount_cleaned-variance ?t) 0.1)
                (at ?v ?w2)
                (not (available))
                (available)
                (increase (time-remaining-variance) (*
                    (* 4 (traverse_time ?w1 ?w2)) (* 4
                    (traverse_time ?w1 ?w2))) )
                (decrease (time-remaining) (*5 (
                   traverse_time ?w1 ?w2)))
            )
)

(:action video-survey-long-target
  :parameters (?v - auv ?t - target ?w1 ?w2 -
     waypoint)
  :precondition (and (at ?v ?w1)
              (or ; Made the AUV able to start a
                 survey from either side of a target.
                   (and
                       (target-starts ?t ?w1)
                       (target-ends ?t ?w2)
                   )
                   (and
                       (target-starts ?t ?w2)
                       (target-ends ?t ?w1)
                   )
                      )
                      (available)
                      (need_survey_data ?t)
                      (>= (time-remaining) (*2 (
                         traverse_time ?w1 ?w2)))

                 )
  :effect (and (not (at ?v ?w1))
                (at ?v ?w2)
                (increase (have_some_survey_data_chain
                    ?t) 1) ; Made the AUV get survey
                   data incrementally - it needs at
                   least 0.9 to send the data.
                (increase (
                   have_some_survey_data_chain-variance
                    ?t) 0.1)
                (not (available))
                (available)
                (increase (time-remaining-variance) (*
                    (* 0.7 (traverse_time ?w1 ?w2)) (*
                    0.7 (traverse_time ?w1 ?w2))) )
                (decrease (time-remaining) (*2 (
                   traverse_time ?w1 ?w2)))
          )
)

 (:action turn-valve-outcome1
  :parameters (?v - auv ?valve - target ?w1 -
     waypoint)
  :precondition (and (at ?v ?w1)
                     (target-at ?valve ?w1)
                     (available)
                     (need_turned ?valve)
                     (>= (time-remaining) 10)


                 )
  :effect (and (turned ?valve)
               (not (need_turned ?valve))
               (increase (time-remaining-variance) (*
                   (* 0.2 10) (* 0.2 10)) )
               (decrease (time-remaining) 10)
               (not (available))
               (available)
               (decrease (unsatisfied-goals) 1)
          )
)

 (:action turn-valve-outcome2
  :parameters (?v - auv ?valve - target ?w1 -
     waypoint)
  :precondition (and (at ?v ?w1)
                     (target-at ?valve ?w1)
                     (available)
                     (need_turned ?valve)
```

```
                        (>= (time−remaining) 10)


                    )
  :effect (and (stuck ?valve)

               (not (at ?v ?w1))
               (stuck−at ?v ?w1)

               (not (need_turned ?valve))
               (increase (time−remaining−variance) (*
                   (* 0.2 10) (* 0.2 10)) )
               (decrease (time−remaining) 10)
               (not (available))
               (available)
          )
)

(:action mend−valve
 :parameters (?v − auv ?valve − target ?w1 − waypoint
    )
  :precondition (and (stuck−at ?v ?w1)
                     (target−at ?valve ?w1)
                     (available)
                     (stuck ?valve)
                    (>= (time−remaining) 10)


                    )
  :effect (and (turned ?valve)
               (not (stuck ?valve))

               (not (stuck−at ?v ?w1))
               (at ?v ?w1)


               (increase (time−remaining−variance) (*
                   (* 0.2 10) (* 0.2 10)) )
```

```
                (decrease (time-remaining) 10)

                (not (available))
                (available)
                (decrease (unsatisfied-goals) 1)
           )
)

(:action photograph_valve
  :parameters (?v - auv ?w - waypoint ?t - target)
  :precondition (and (target-at ?t ?w)
                     (available)
                     (need_photo_data ?t)
                     (>= (time-remaining) 1)

                     (at ?v ?w))
  :effect (and (have_image ?t)
                (not (available))
                (not (need_photo_data ?t))
                (available)
                (decrease (time-remaining) 1)
                )
  )

(:action communicate_photo_data
  :parameters (?v - auv ?t - target ?w - waypoint)
  :precondition (and (at ?v ?w)
                     (available)
                     (have_image ?t)
                     (>= (time-remaining) 0)
                 )
  :effect (and (communicated_photo_data ?t)
                (not (have_image ?t))
                (not (available))
                (available)
                (decrease (unsatisfied-goals) 1)
           )
  )
  (:action communicate_survey_data
    :parameters (?v - auv ?t - target ?w - waypoint)
    :precondition (and (at ?v ?w)
                       (available)
                       (need_survey_data ?t)
                       (>= (time-remaining) 0)
                       (>= (have_some_survey_data_chain
                            ?t) 0.9))
    :effect (and (communicated_survey_data ?t)
                 (assign (have_some_survey_data_chain ?
                    t) 0)
                 (not (available))
                 (available)
                 (decrease (unsatisfied-goals) 1)
                 (not (need_survey_data ?t))
             )
    )
)
```

Appendix C: Domain & Configuration Used in Chapter 5

Rovers - single outcome, arbitrary uncertainty

```
(define (domain Rover)

(:requirements :typing :fluents)

(:types rover waypoint store camera mode lander
   objective)

(:predicates
  (at ?r - rover ?w - waypoint)
  (at_lander ?l - lander ?w - waypoint)
  (can_traverse ?r - rover ?w1 - waypoint ?w2 -
     waypoint)
  (equipped_for_soil_analysis ?r - rover)
  (equipped_for_rock_analysis ?r - rover)
  (equipped_for_imaging ?r - rover)
  (empty ?s - store)
  (have_rock_analysis ?r - rover ?w - waypoint)
  (have_soil_analysis ?r - rover ?w - waypoint)
  (full ?s - store)
  (calibrated ?c - camera ?r - rover)
  (supports ?c - camera ?m - mode)
  (available ?r - rover)
  (visible ?w1 - waypoint ?w2 - waypoint)
  (have_image ?r - rover ?o - objective ?m - mode)
  (communicated_soil_data ?w - waypoint)
  (communicated_rock_data ?w - waypoint)
  (communicated_image_data ?o - objective ?m - mode)
  (at_soil_sample ?w - waypoint)
  (at_rock_sample ?w - waypoint)
  (visible_from ?o - objective ?w - waypoint)
  (store_of ?s - store ?r - rover)
  (calibration_target ?c - camera ?o - objective)
  (on_board ?c - camera ?r - rover)
  (channel_free ?l - lander)
  (in_sun ?w - waypoint)
)

(:functions
  (energy ?r - rover)
)

(:action navigate
  :parameters (?r - rover ?w1 - waypoint ?w2 - waypoint
     )
  :precondition
    (and (can_traverse ?r ?w1 ?w2)
    (available ?r)
    (at ?r ?w1)
    (visible ?w1 ?w2)
    (>= (energy ?r) 10)
    )
  :effect
    (and (not (at ?r ?w1))
    (at ?r ?w2)
    (decrease (energy ?r) 10)
    )
)
```

```
(:action recharge
  :parameters (?r - rover ?w - waypoint)
  :precondition
    (and (at ?r ?w)
    (in_sun ?w)
    )
  :effect
    (and (assign (energy ?r) 100)
    )
)

(:action sample_soil
  :parameters (?r - rover ?s - store ?w - waypoint)
  :precondition
    (and (at ?r ?w)
    (at_soil_sample ?w)
    (equipped_for_soil_analysis ?r)
    (store_of ?s ?r)
    (empty ?s)
    (>= (energy ?r) 2)
    )
  :effect
    (and (not (empty ?s))
    (full ?s)
    (have_soil_analysis ?r ?w)
    (not (at_soil_sample ?w))
    (decrease (energy ?r) 2)
    )
)

(:action sample_rock
  :parameters (?r - rover ?s - store ?w - waypoint)
  :precondition
    (and (at ?r ?w)
    (at_rock_sample ?w)
    (equipped_for_rock_analysis ?r)
    (store_of ?s ?r)
    (empty ?s)
    (>= (energy ?r) 4)
    )
  :effect
    (and (not (empty ?s))
    (full ?s)
    (have_rock_analysis ?r ?w)
    (not (at_rock_sample ?w))
    (decrease (energy ?r) 4)
    )
)

(:action drop
  :parameters (?r - rover ?s - store)
  :precondition
    (and (store_of ?s ?r)
    (full ?s)
    )
  :effect
    (and (not (full ?s))
    (empty ?s)
    )
)

(:action calibrate
  :parameters (?r - rover ?c - camera ?o - objective ?w
      - waypoint)
  :precondition
    (and (equipped_for_imaging ?r)
    (calibration_target ?c ?o)
    (at ?r ?w)
    (visible_from ?o ?w)
    (on_board ?c ?r)
    (>= (energy ?r) 2)
    )
  :effect
    (and (calibrated ?c ?r)
    (decrease (energy ?r) 2)
    )
```

```
)

(:action take_image
  :parameters (?r - rover ?w - waypoint ?o - objective
     ?c - camera ?m - mode)
  :precondition
    (and (calibrated ?c ?r)
    (on_board ?c ?r)
    (equipped_for_imaging ?r)
    (supports ?c ?m)
    (visible_from ?o ?w)
    (at ?r ?w)
    (>= (energy ?r) 1)
    )
  :effect
    (and (have_image ?r ?o ?m)
    (not (calibrated ?c ?r))
    (decrease (energy ?r) 1)
    )
)

(:action communicate_soil_data
  :parameters (?r - rover ?l - lander ?w1 - waypoint ?
     w2 - waypoint ?w3 - waypoint)
  :precondition
    (and (at ?r ?w2)
    (at_lander ?l ?w3)
    (have_soil_analysis ?r ?w1)
    (visible ?w2 ?w3)
    (available ?r)
    (channel_free ?l)
    (>= (energy ?r) 4)
    )
  :effect
    (and (not (available ?r))
    (not (channel_free ?l))
    (channel_free ?l)
    (communicated_soil_data ?w1)
    (available ?r)
    (decrease (energy ?r) 4)
    )
)

(:action communicate_rock_data
  :parameters (?r - rover ?l - lander ?w1 - waypoint ?
     w2 - waypoint ?w3 - waypoint)
  :precondition
    (and (at ?r ?w2)
    (at_lander ?l ?w3)
    (have_rock_analysis ?r ?w1)
    (visible ?w2 ?w3)
    (available ?r)
    (channel_free ?l)
    (>= (energy ?r) 4)
    )
  :effect
    (and (not (available ?r))
    (not (channel_free ?l))
    (channel_free ?l)
    (communicated_rock_data ?w1)
    (available ?r)
    (decrease (energy ?r) 4)
    )
)

(:action communicate_image_data
  :parameters (?r - rover ?l - lander ?o - objective ?m
      - mode ?w1 - waypoint ?w2 - waypoint)
  :precondition
    (and (at ?r ?w1)
    (at_lander ?l ?w2)
    (have_image ?r ?o ?m)
    (visible ?w1 ?w2)
    (available ?r)
    (channel_free ?l)
    (>= (energy ?r) 6)
    )
  :effect
    (and (not (available ?r))
    (not (channel_free ?l))
    (channel_free ?l)
    (communicated_image_data ?o ?m)
    (available ?r)
    (decrease (energy ?r) 6)
    )
)

)
```

```
{
  "navigate:energy": {
    "type": "zoo",
    "scales": [0.33, 0.33, 0.33],
    "components": [{
      "type": "gaussian",
      "from": 7,
      "to": 9,
      "zero": 8,
      "parameters": {
        "sigma": 1
      }
    }, {
      "type": "gaussian",
      "from": 9,
      "to": 11,
      "zero": 10,
      "parameters": {
        "sigma": 1
      }
    }, {
      "type": "gaussian",
        "from": 11,
        "to": 21,
        "zero": 16,
        "parameters": {
          "sigma": 1
        }
      }]
    }
}
```

Appendix D: Domain & Configuration Used in Chapter 6

Rovers - multiple outcomes, arbitrary uncertainty

```
(define (domain Rover)

(:requirements :typing :fluents)

(:types rover waypoint store camera mode lander
   objective)

(:predicates
  (at ?r - rover ?w - waypoint)
  (at_lander ?l - lander ?w - waypoint)
  (can_traverse ?r - rover ?w1 - waypoint ?w2 -
     waypoint)
  (equipped_for_soil_analysis ?r - rover)
  (equipped_for_rock_analysis ?r - rover)
  (equipped_for_imaging ?r - rover)
  (empty ?s - store)
  (have_rock_analysis ?r - rover ?w - waypoint)
  (have_soil_analysis ?r - rover ?w - waypoint)
  (full ?s - store)
  (calibrated ?c - camera ?r - rover)
  (supports ?c - camera ?m - mode)
  (available ?r - rover)
  (visible ?w1 - waypoint ?w2 - waypoint)
  (have_image ?r - rover ?o - objective ?m - mode)
  (communicated_soil_data ?w - waypoint)
  (communicated_rock_data ?w - waypoint)
  (communicated_image_data ?o - objective ?m - mode)
  (at_soil_sample ?w - waypoint)
  (at_rock_sample ?w - waypoint)
  (visible_from ?o - objective ?w - waypoint)
  (store_of ?s - store ?r - rover)
  (calibration_target ?c - camera ?o - objective)
  (on_board ?c - camera ?r - rover)
  (channel_free ?l - lander)
  (in_sun ?w - waypoint)
)

(:functions
  (energy ?r - rover)
)

(:action navigate-outcome-lucky
  :parameters (?r - rover ?w1 - waypoint ?w2 - waypoint
     )
  :precondition
    (and (can_traverse ?r ?w1 ?w2)
    (available ?r)
    (at ?r ?w1)
    (visible ?w1 ?w2)
    (>= (energy ?r) 10)
    )
  :effect
    (and (not (at ?r ?w1))
    (at ?r ?w2)
    (decrease (energy ?r) 8)
    )
)
```

```
(:action navigate-outcome-nominal
  :parameters (?r - rover ?w1 - waypoint ?w2 - waypoint
     )
  :precondition
    (and (can_traverse ?r ?w1 ?w2)
    (available ?r)
    (at ?r ?w1)
    (visible ?w1 ?w2)
    (>= (energy ?r) 10)
    )
  :effect
    (and (not (at ?r ?w1))
    (at ?r ?w2)
    (decrease (energy ?r) 9)
    )
)

(:action navigate-outcome-unlucky
  :parameters (?r - rover ?w1 - waypoint ?w2 - waypoint
     )
  :precondition
    (and (can_traverse ?r ?w1 ?w2)
    (available ?r)
    (at ?r ?w1)
    (visible ?w1 ?w2)
    (>= (energy ?r) 10)
    )
  :effect
    (and (not (at ?r ?w1))
    (at ?r ?w2)
    (decrease (energy ?r) 16)
    )
)

(:action recharge
  :parameters (?r - rover ?w - waypoint)
  :precondition
```

```
    (and (at ?r ?w)
    (in_sun ?w)
    )
  :effect
    (and (assign (energy ?r) 100)
    )
)

(:action sample_soil
  :parameters (?r - rover ?s - store ?w - waypoint)
  :precondition
    (and (at ?r ?w)
    (at_soil_sample ?w)
    (equipped_for_soil_analysis ?r)
    (store_of ?s ?r)
    (empty ?s)
    (>= (energy ?r) 2)
    )
  :effect
    (and (not (empty ?s))
    (full ?s)
    (have_soil_analysis ?r ?w)
    (not (at_soil_sample ?w))
    (decrease (energy ?r) 2)
    )
)

(:action sample_rock
  :parameters (?r - rover ?s - store ?w - waypoint)
  :precondition
    (and (at ?r ?w)
    (at_rock_sample ?w)
    (equipped_for_rock_analysis ?r)
    (store_of ?s ?r)
    (empty ?s)
    (>= (energy ?r) 4)
    )
  :effect
```

```
    (and (not (empty ?s))
    (full ?s)
    (have_rock_analysis ?r ?w)
    (not (at_rock_sample ?w))
    (decrease (energy ?r) 4)
    )
)

(:action drop
  :parameters (?r - rover ?s - store)
  :precondition
    (and (store_of ?s ?r)
    (full ?s)
    )
  :effect
    (and (not (full ?s))
    (empty ?s)
    )
)

(:action calibrate
  :parameters (?r - rover ?c - camera ?o - objective ?w
      - waypoint)
  :precondition
    (and (equipped_for_imaging ?r)
    (calibration_target ?c ?o)
    (at ?r ?w)
    (visible_from ?o ?w)
    (on_board ?c ?r)
    (>= (energy ?r) 2)
    )
  :effect
    (and (calibrated ?c ?r)
    (decrease (energy ?r) 2)
    )
)

(:action take_image
  :parameters (?r - rover ?w - waypoint ?o - objective
     ?c - camera ?m - mode)
  :precondition
    (and (calibrated ?c ?r)
    (on_board ?c ?r)
    (equipped_for_imaging ?r)
    (supports ?c ?m)
    (visible_from ?o ?w)
    (at ?r ?w)
    (>= (energy ?r) 1)
    )
  :effect
    (and (have_image ?r ?o ?m)
    (not (calibrated ?c ?r))
    (decrease (energy ?r) 1)
    )
)

(:action communicate_soil_data
  :parameters (?r - rover ?l - lander ?w1 - waypoint ?
     w2 - waypoint ?w3 - waypoint)
  :precondition
    (and (at ?r ?w2)
    (at_lander ?l ?w3)
    (have_soil_analysis ?r ?w1)
    (visible ?w2 ?w3)
    (available ?r)
    (channel_free ?l)
    (>= (energy ?r) 4)
    )
  :effect
    (and (not (available ?r))
    (not (channel_free ?l))
    (channel_free ?l)
    (communicated_soil_data ?w1)
    (available ?r)
    (decrease (energy ?r) 4)
    )
```

```
)

(:action communicate_rock_data
  :parameters (?r - rover ?l - lander ?w1 - waypoint ?
     w2 - waypoint ?w3 - waypoint)
  :precondition
    (and (at ?r ?w2)
    (at_lander ?l ?w3)
    (have_rock_analysis ?r ?w1)
    (visible ?w2 ?w3)
    (available ?r)
    (channel_free ?l)
    (>= (energy ?r) 4)
    )
  :effect
    (and (not (available ?r))
    (not (channel_free ?l))
    (channel_free ?l)
    (communicated_rock_data ?w1)
    (available ?r)
    (decrease (energy ?r) 4)
    )
)

(:action communicate_image_data
  :parameters (?r - rover ?l - lander ?o - objective ?m
      - mode ?w1 - waypoint ?w2 - waypoint)
  :precondition
    (and (at ?r ?w1)
    (at_lander ?l ?w2)
    (have_image ?r ?o ?m)
    (visible ?w1 ?w2)
    (available ?r)
    (channel_free ?l)
    (>= (energy ?r) 6)
    )
  :effect
    (and (not (available ?r))
    (not (channel_free ?l))
    (channel_free ?l)
    (communicated_image_data ?o ?m)
    (available ?r)
    (decrease (energy ?r) 6)
    )
)

)
```

```
{
  "navigate-outcome-lucky:energy": {
    "type": "gaussian",
    "from": 7.5,
    "to": 8.5,
    "zero": 8,
    "parameters": {
      "sigma": 1
    }
  },
  "navigate-outcome-nominal:energy": {
    "type": "gaussian",
    "from": 8.5,
    "to": 9.5,
    "zero": 9,
    "parameters": {
      "sigma": 1
    }
  },
  "navigate-outcome-unlucky:energy": {
    "type": "gaussian",
    "from": 11,
    "to": 21,
    "zero": 16,
    "parameters": {
         "sigma": 1
      }
   }
}
```