Attributing Blame To Decisions Using Tractable Probabilistic Models

Lewis Hammond

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2018

Abstract

The increasing use of artificial intelligence systems to make decisions in morally sensitive domains has led to concerns over the safety, explainability, and responsibility of these decisions. Other concerns arising from larger data resources and more complex environments have prompted the development of a new class of tractable probabilistic models that can be learnt from data. Motivated by these two factors, we combine the formal framework of blameworthiness in [1] and a recently proposed tractable model known as a probabilistic sentential decision diagram [2] in order to create a system that can correctly and efficiently attribute blame to decisions in a variety of complex, uncertain settings. As part of this, we provide: a theoretical mapping between the framework and the model; results on the computational complexity of key calculations; an algorithmic pipeline that implements the system; and a series of experiments to empirically demonstrate the suitability of our approach. We hope that the results of this project go some way towards bridging the gap between the philosophical work on moral responsibility and the technical work on decision-making in artificial intelligence systems.

Acknowledgements

First and foremost I would like to thank my supervisor Vaishak Belle for his help and guidance during the project. I have also benefitted greatly from many of the school's other staff this year, including (though not limited to) Jacques Fleuriot, Iain Murray, and Michael Gutmann, each of whom introduced me to important and interesting ideas that I have made use of in this work. I am grateful for many stimulating and enjoyable conversations with my peers, and in particular to Sophia Jones and Michael Varley for proofreading earlier versions of this dissertation. Further afield, I would like to thank Yitao Liang for his patience and technical assistance with respect to several of the resources used here. Finally, and most importantly, I thank my family, without whose support (financial, emotional, or otherwise) I would certainly not have been able to complete this degree, let alone this dissertation.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Lewis Hammond*)

Table of Contents

Chapter 1

Introduction

We begin with a summary of our motivation, and the objectives and contributions of this project. Following this we give an overview of the structure of the report including brief descriptions of the remaining sections. Please note that the first two sections of this chapter are based largely upon [3], as is appropriate given the nature of this prior work.

1.1 Motivation and Objectives

As artificial intelligence (AI) systems have continued to become more powerful, so too have they become more frequently employed to make autonomous or semi-autonomous decisions. The kinds of decisions made vary widely but increasingly they are in safety-critical applications such as medical decision support systems and autonomous vehicles [4; 5; 6]. The use of AI systems to help make decisions of moral consequence immediately leads one to consider the problem of attributing moral responsibility, and in particular blame, to such decisions. As interactions between artificial agents and human agents become further embedded into society, there is a greater need for a computational framework that can be used as a solution. At the time of writing, the only comprehensive proposal to this end is the blameworthiness framework (BF) in [1], although we know of no existing attempt to apply these abstract definitions to a particular model or implementation.

Probabilistic graphical models (PGMs) can be used to compactly encode and reason about complex, uncertain domains. The two primary disadvantages of PGMs (with respect to our purposes) are that both inference and learning become intractable for large models. This has led to the development of a new class of tractable probabilistic

models (TPMs) that can be efficiently learnt from data and for which many kinds of inference and computation have complexities that are linear in the size of the model. Here we focus on one recently proposed member of this class known as a probabilistic sentential decision diagram (PSDD) [2]. PSDDs possess a range of desirable properties including several that make them particularly suitable for modelling moral decision-making scenarios, as will be discussed in the following chapter.

Given our motivations as described above, our overall objective is to create a system that can correctly and efficiently attribute blame to decisions in a variety of complex, uncertain settings. Importantly, this system should be able to learn models (more specifically probabilistic distributions over decision-making scenarios, and utility functions) from data, but also have certain aspects manually specified by the user, if so desired. It should capture precisely the important theoretical concepts in the BF, but without sacrificing the tractability, interpretability, or other desirable properties of PSDDs which underlie the individual models that the system creates and uses. Finally, and perhaps most importantly, the system must be applicable to a wide range of decision-making scenarios and be able to make attributions of blame that are both appropriate and in line with human judgement.

1.2 Contributions

As far as the authors have been able to determine, this project is the first (and, at the time of writing, only) attempt to combine an abstract, theoretical definition for blameworthiness with a concrete, computational model applied to decision-making scenarios. Though these two key elements already exist, their combination and the results of our project represent a novel contribution to the increasingly large and important field of AI ethics, while also introducing new themes and ideas to areas such TPMs, decision theory, and explainable AI. The main contributions of the project can be summarised as follows, with each point corresponding approximately to each of the remaining chapters:

- A brief but comprehensive review of the literature on moral decision-making in AI systems
- A justified and correct mapping of the BF to a computational model (PSDDs), including complexity results

- A fully implemented algorithmic pipeline for learning models (including simple utility functions) from data and using them to calculate blameworthiness scores
- A series of experimental results using a variety of datasets and models that demonstrate the applicability of our system
- A discussion of the successes and limitations of our approach, with some promising suggestions for further work

1.3 Overview

The remaining chapters are ordered and structured so as to generally reflect the course of the project. In Chapter 2 we provide key definitions, theorems, and notation for the BF and for PSDDs. We also introduce several important ideas and background assumptions concerning ethical decision-making and utility functions, as well as discussing some related work. Following this, Chapter 3 contains our theoretical mapping between the the two frameworks together with justifications of any assumptions we make, a proof of correctness, and results on the complexity of computing the various key elements needed to generate blameworthiness scores. We then proceed to describe the implementation of this mapping in Chapter 4, which includes the details of both existing and novel resources, and how we combine them to form our final algorithmic pipeline. In Chapter 5 we use this pipeline to learn a selection of models from data, before presenting and evaluating our results. We conclude in Chapter 6 with a summary of the project and a look at possible directions for future work. Finally, note that there are three appendices containing acronyms and nomenclature, code documentation and samples, and datasets with further analysis, respectively.

Chapter 2

Background and Related Work

Here we provide the necessary theoretical background that underlies our work, along with the relevant definitions, notation, and key results. This includes summaries of the different frameworks we employ in our project, and short introductions to other significant ideas. We follow this with a brief survey of related work so as to place our contribution within its proper context.

2.1 Ethics

Before giving the theoretical background for both the BF and PSDDs we devote a small amount of space to a discussion of some of the more high-level assumptions and philosophical issues that surround the ethics of decision-making. This also includes an introduction to the concept of utility functions.

2.1.1 Decision-Making

When attempting to make a moral decision it is common to appeal to some form of normative ethical theory that explains how one should act in a given scenario. Broadly speaking, there are three main types of normative ethical theory, though in practice these are not always mutually exclusive:

- Consequentialist theories [7] are built upon the idea that how right or wrong a decision is is based purely on how good or bad the consequences of that action are. An example of a consequentialist moral principle might be to take the action that leads to the greatest average happiness across all people.

- Deontological theories [8] stress the importance of rights and obligations and can be seen as primarily rule-based. An example of a deontological moral principle might be that one should never kill another person.

- Virtue ethics theories [9] focus less on the morality of specific decisions but on the overall ethical characteristics of the agent making the decision, hence decisions are made according to how closely they align with some set of virtues. An example of a principle of virtue ethics might be to act with courage.

The various merits and details of these theories are well beyond the scope of this project, but it is important to appreciate that here, as in the vast majority of the work on AI ethics, we take a consequentialist stance. However, in the sections that follow we also indicate how our system can be used to include deontological rules.

2.1.2 Utility Functions

In order to encode a consequentialist theory of ethics the notion of a utility function is critical. To be able to measure how right or wrong a decision is (so as to choose the best option), we require a way to measure how good or bad the consequences of that action are. A utility function is the traditional answer to this problem and can be used to represent the preferences of an individual or a group [10; 11].

Formally, we define an agent's utility function $\mathbf{u}$ as a map from the set of all possible worlds $\mathcal{W}$ to the set $\mathbb{R}_{\geq 0}$ such that for any two possible worlds w_1 and w_2, we have $\mathbf{u}(w_1) > \mathbf{u}(w_2)$ if and only if the agent prefers w_1 to w_2, and $\mathbf{u}(w_1) = \mathbf{u}(w_2)$ just in case the agent prefers w_1 and w_2 equally. We also choose here to include a notion of cardinality as opposed to mere ordinality. By this we mean that the magnitude of difference between the utility of two possible worlds can be quantified. Using an ordinal function simply encodes the original preference ranking, so while we might be able to say w_1 is preferred to w_2 and w_2 is preferred to w_3 we do not have any information about the strength of these preferences. In contrast, given the mapping to $\mathbb{R}_{\geq 0}$ in our cardinal function, then if $\mathbf{u}(w_1) = 100$, $\mathbf{u}(w_2) = 60$, and $\mathbf{u}(w_3) = 40$, we can say that w_1 is preferred to w_2 twice as much as w_2 is preferred to w_3.

The idea that humans (or other animals) actually possess a utility function is not entirely uncontroversial, but is widely assumed in many areas of economics, philosophy, and other fields. Whether or not agents have access to their preferences, at the very least it seems plausible that given any two possible worlds an agent will prefer one to the other, or else prefer them both equally, and further that these preferences

are consistent (if the agent prefers w_1 to w_2 and w_2 to w_3 then they prefer w_1 to w_3). More contentious is the proposal that this utility function is cardinal rather than ordinal, though this assumption is necessary for our work and so we do not devote any further space to the issue.

2.2 Blameworthiness

Throughout this project we use the word *blameworthiness* to capture an important part of what can more broadly be described as moral responsibility. There are several other closely related concepts such as guilt and culpability, though we do not attempt to draw lines between the definitions of these words. With that said, we might reasonably think that one can be morally responsible for both good and bad outcomes, yet it seems wrong to say that one can be blamed for a good outcome. This, however, is consistent with the framework presented in [1], in which all actions are viewed as costly with respect to the utility function. Defining an analogous notion of praiseworthiness within this same framework raises other issues that we do not have space to discuss here, but for which we recommend the interested reader refer to the original paper.

In the subsections below we first present an exposition of the parts of the BF that we require for our later contributions, before providing a small worked example to help illustrate the key concepts. Please note that although all of the definitions in this section are taken directly from [1] we have made some slight changes to the notation for the sake of clarity and for ease of integration with the notation of PSDDs, which we present later.

2.2.1 Definitions

The underlying theoretical machinery in [1] is that of the structural equations framework (SEF) from [12], which crucially provides the ability to model counterfactual statements, and thus to define causality, intention, and blameworthiness. In the SEF we model environments by viewing them in terms of variables and the values they take, along with (modifiable) structural equations acting as relations between these variables.

More formally, the set of variables are partitioned into *exogenous* variables $\mathcal{X}$ whose values are determined by factors external to the model in question, and *endogenous* variables $\mathcal{V}$ that are internal to the model and whose values are determined

by those of the exogenous variables. We also have a range function $\mathcal{R}$ that maps every variable to the set of possible values it may take. In any model, there exists one structural equation for each endogenous variable V that takes the form of a function $F_V : \times_{Y \in \mathcal{X} \cup \mathcal{V} \setminus V} \mathcal{R}(Y) \to \mathcal{R}(V)$. This initial set of terms and the corresponding notation allows us to provide the first set of definitions we require.

Definition 2.2.1. A **causal model** M is a pair $(\mathcal{S}, \mathcal{F})$ where $\mathcal{S}$ is a **signature** $(\mathcal{U}, \mathcal{V}, \mathcal{R})$ and $\mathcal{F}$ is a set of **modifiable structural equations** $\{F_V : V \in \mathcal{V}\}$ as described above. A **causal setting** is a pair $(M, \mathbf{X})$ where $\mathbf{X} \in \times_{X \in \mathcal{X}} \mathcal{R}(X)$ is a **context**, a setting of the exogenous variables.

An important caveat to note here is that, as in [12], the authors of [1] restrict their considerations to what are known as *recursive* models. In such models there is a total ordering $\prec$ on the variables in $\mathcal{V}$ such that in any causal setting, if $V_i \prec V_j$ then the value of V_i is independent of the value of V_j. In other words, the value of V_1 is directly determined by the values of the variables in $\mathcal{X}$, the value of V_2 is directly determined by the values of the variables in $\mathcal{X}$ and the value of V_1, and so on. It can then immediately be seen that given a context $\mathbf{X}$, the values of all variables in $\mathcal{V}$ are uniquely determined. This leads us to the next definition.

Definition 2.2.2. A **primitive event** is an equation of the form $V = v$ for some $V \in \mathcal{V}$ where $v \in \mathcal{R}(V)$, and a **causal formula** is denoted $[\mathcal{Y} \leftarrow \mathbf{Y}]\varphi$ where $\mathcal{Y} \subseteq \mathcal{V}$, $\mathbf{Y} \in \times_{Y \in \mathcal{Y}} \mathcal{R}(Y)$, and φ is a Boolean formula of primitive events. In natural language the causal formula says that if the variables in $\mathcal{Y}$ were set to values $\mathbf{Y}$ then φ would hold. For a causal formula ψ we write $(M, \mathbf{X}) \vDash \psi$ if ψ is made true in causal setting $(M, \mathbf{X})$.

As their first step towards a formal definition of blameworthiness, the authors of [1] introduce the notion of an agent's epistemic state $(\mathbf{p}, \mathcal{K}, \mathbf{u})$ where $\mathcal{K}$ is a set of causal settings, and $\mathbf{p}$ is a probability distribution over this set, representing the agent's uncertainty about both the structure of the causal model M and the particular context $\mathbf{X}$. The third element is a utility function $\mathbf{u} : \mathcal{W} \to \mathbb{R}_{\geq 0}$ on the set of worlds, where a world w is defined as a setting of values to all variables in $\mathcal{V}$. Due to the recursive nature of the models we consider, we may thus write $w_{M,\mathbf{X}}$ for the unique world determined by the causal setting $(M, \mathbf{X})$. As their second step, a particular variable $A \in \mathcal{V}$ is identified in order to capture an action of the agent. We are now able to present a definition for the first key element in computing blameworthiness.

Definition 2.2.3. We measure **how much more likely it is that φ will result from performing a than from performing a'** using:

$$\delta_{a,a',\varphi} = max\left(\left[\sum_{(M,\mathbf{X})\in[\![[A=a]\varphi]\!]} \mathbf{p}(M,\mathbf{X}) - \sum_{(M,\mathbf{X})\in[\![[A=a']\varphi]\!]} \mathbf{p}(M,\mathbf{X})\right], 0\right)$$

where $[\![\psi]\!] = \{(M,\mathbf{X}) \in \mathcal{K} : (M,\mathbf{X}) \vDash \psi\}$ is the set of causal settings in which ψ (a causal formula) is satisfied.

This quantity is simply the likelihood of being in a causal setting where a results in φ minus the likelihood of being in a causal setting where a' results in φ, unless this quantity is negative. In the latter case then a is less likely to result in φ than a', and so we set $\delta_{a,a',\varphi}$ to 0, as we do not wish to consider the agent at all blameworthy for causing φ by performing action a (relative to a') [1].

The second key element in determining blameworthiness is the cost of the actions. In order to define a cost function we identify a set $O \subseteq \mathcal{V}$ of outcome variables whose values are determined by an assignment of values to all other variables. It is these variables that we measure cost with respect to and they can be thought of as the subset of variables encoding the space of possible outcomes we care about. In a given causal setting $(M,\mathbf{X})$, we write $\mathbf{O}_{A\leftarrow a} \in O$ to denote the setting of the outcome variables when action a is performed and $w_{M,O\leftarrow\mathbf{O}_{A\leftarrow a},\mathbf{X}}$ to denote the corresponding world.

Definition 2.2.4. The (expected) **cost of a relative to O** is:

$$c(a) = \sum_{(M,\mathbf{X})\in\mathcal{K}} \mathbf{p}(M,\mathbf{X})\left[\mathbf{u}(w_{M,\mathbf{X}}) - \mathbf{u}(w_{M,O\leftarrow\mathbf{O}_{A\leftarrow a},\mathbf{X}})\right]$$

In [1] actions are always costly, and so $\mathbf{u}(w_{M,\mathbf{X}})$ describes the utility of the world determined by the causal setting $(M,\mathbf{X})$ when no action is performed by the agent. Note also that we take the expectation over the agent's epistemic state represented by the distribution $\mathbf{p}$ over $\mathcal{K}$.

The costs of an action are an important consideration in attributing blame. For example, we would not typically blame an agent for running over a rabbit in their car if the only other option would have been to sacrifice their own life by swerving off the road into a tree, because the cost (to the agent) involved in this latter action is significantly higher than the cost of the rabbit's death. The final step taken is to combine the two elements from the previous two definitions into a single definition of blameworthiness that takes into account both the probability of the outcome in question

[1] Here it is implicitly assumed that we do not assign negative blame.

occurring given each action, and the relative costs of each action. In order to do this we introduce one last quantity N which measures how important we view the costs of actions as being when attributing blame. Specifically, the larger N becomes the less we care about the cost of actions.

Definition 2.2.5. The **degree of blameworthiness of** a **for** φ **relative to** a' (given a cost function c and a cost importance measure N) is:

$$db_N(a,a',\varphi) = \delta_{a,a',\varphi}\frac{N - max(c(a') - c(a),0)}{N}$$

The overall **degree of blameworthiness of** a **for** φ is then:

$$db_N(a,\varphi) = max_{a' \in \mathcal{R}(A)\setminus a} db_N(a,a',\varphi))$$

It can be seen in our definition above that as $N \to \infty$ then $db_N(a,a',\varphi) \to \delta_{a,a',\varphi}$; the blameworthiness of action a for an outcome φ relative to action a' simply becomes our measure of how much more likely it is that φ will result from performing a than from performing a', with no regard to the cost of each action. Note that we assume that blame is always non-negative and so we require that $N > max_{a \in A} c(a)$ to ensure that the numerator of the fraction in the equation above is always positive. As is mentioned by the original authors, we also assume that the value of N varies according to the decision-making scenario in question.

2.2.2 An Example

It is instructive to study a small example. Consider the following decision-making scenario in which a woman is walking to work and is not sure if it will rain (she thinks the probability it will is 0.5). If she goes back to collect her umbrella there is a probability (again, 0.5) she will be late to work. However, more important to her than being on time to work is whether she arrives at work dry. We can encode this using the BF as follows:

- $\mathcal{X} = \{R\}$ where $R = 1$ if it rains and $R = 0$ otherwise (we may also write this as simply R and $\neg R$)
- $\mathcal{V} = \{U, W, L\}$ where $U = 1$ if the woman goes back to get her umbrella, $W = 1$ if she arrives to work wet, and $L = 1$ if she arrives to work late (with each variable equal to 0 in the alternative case), hence $O = \{W, L\}$

- M_1 contains the variables above and the structural equations $\mathcal{F}_1$ that encode the case where the woman is late to work due to her going back to collect her umbrella, and M_2 is as above but includes structural equations $\mathcal{F}_2$ that tell us she is not late even though she goes back to collect her umbrella
- $\mathcal{K} = \{(M_1, \neg R), (M_1, R), (M_2, \neg R), (M_2, R)\}$ and $\mathbf{p}$ is such that $\mathbf{p}(M, \mathbf{X}) = 0.25$ for all $(M, \mathbf{X}) \in \mathcal{K}$
- $\mathbf{u}$ is such that in any world w, if the woman is on time to work she receives utility 2 and if she arrives to work dry she receives utility 3 (thus her theoretical maximum utility is 5 when she arrives to work on time and dry, and her theoretical minimum is 0 when she arrives late and wet from the rain)

Now suppose we want calculate how blameworthy the woman is for being late to work because she went back to get her umbrella (as opposed to not going back). In other words we compute $db_N(U, \neg U, L = 1)$. The first element we need is how much more likely it is that she will be late due to going back for the umbrella, captured by $\delta_{U,\neg U,L=1}$. Note that $[\![[U = 1](L = 1)]\!] = \{(M_1, \neg R), (M_1, R)\}$ and $[\![[U = 0](L = 1)]\!] = \varnothing$. Thus we have:

$$\begin{aligned}
\delta_{U,\neg U,L=1} &= max\left(\left[\sum_{(M,\mathbf{X})\in\{(M_1,\neg R),(M_1,R)\}} \mathbf{p}(M,\mathbf{X}) - \sum_{(M,\mathbf{X})\in\varnothing} \mathbf{p}(M,\mathbf{X})\right], 0\right) \\
&= max\big([(0.25 + 0.25) - 0], 0\big) \\
&= max(0.5, 0) \\
&= 0.5
\end{aligned}$$

The second element we require are the costs $c(U)$ and $c(\neg U)$ for performing the action of collecting the umbrella or not. To calculate these we first make the assumption that in each of the causal settings where $R = 1$ then the default action is $U = 1$, and each of the causal settings where $R = 0$ then the default action is $U = 0$. This means that $\mathbf{u}(w_{M,\mathbf{X}})$ is well-defined for each causal setting $(M, \mathbf{X}) \in \mathcal{K}$. Hence we have:

$$\begin{aligned}
c(U) &= \sum_{(M,\mathbf{X})\in\mathcal{K}} \mathbf{p}(M,\mathbf{X})\big[\mathbf{u}(w_{M,\mathbf{X}}) - \mathbf{u}(w_{M,O\leftarrow\mathbf{O}_{U\leftarrow 1},\mathbf{X}})\big] \\
&= 0.25[5-3] + 0.25[3-3] + 0.25[5-5] + 0.25[5-5] \\
&= 0.5 + 0 + 0 + 0 \\
&= 0.5
\end{aligned}$$

$$
\begin{aligned}
c(\neg U) &= \sum_{(M,\mathbf{X})\in\mathcal{K}} \mathbf{p}(M,\mathbf{X})\left[\mathbf{u}(w_{M,\mathbf{X}}) - \mathbf{u}(w_{M,O\leftarrow\mathbf{O}_{U\leftarrow 0},\mathbf{X}})\right] \\
&= 0.25[5-5] + 0.25[3-2] + 0.25[5-5] + 0.25[5-2] \\
&= 0 + 0.25 + 0 + 0.75 \\
&= 1
\end{aligned}
$$

Finally, in order to complete the example we choose, somewhat arbitrarily[2], $N = 2$ which gives us the final blameworthiness score:

$$
\begin{aligned}
db_2(U,\neg U, L=1) &= \delta_{U,\neg U,L=1}\frac{2 - max(c(\neg U) - c(U), 0)}{2} \\
&= 0.5\frac{2 - max(1-0.5, 0)}{2} \\
&= 0.375
\end{aligned}
$$

2.3 Tractable Probabilistic Models

Many widely-used PGMs such as Bayesian networks and Markov random fields suffer from the problem of intractable inference which can in turn lead to intractability problems in learning [13]. One solution is to use approximate inference methods such as variational Bayesian methods and Markov chain Monte Carlo. An alternative is to instead impose restrictions upon the structure of the model so as to guarantee tractability. Such models form the growing class of TPMs and vary in the restrictions to their structure and the tractability guarantees that they provide.

In the following subsections we introduce PSDDs, one particular instance of a TPM, and use the same scenario as in Section 2.2.2 to illustrate the form of the model. Finally we justify our choice of PSDDs, highlighting the features that make them particularly well-suited to our purposes compared to other TPMs.

2.3.1 Probabilistic Sentential Decision Diagrams

A PSDD is a tractable representation of a probability distribution over a propositional logic theory (also known as a propositional knowledge base), a set of sentences in propositional logic. We henceforth refer to the propositions in these sentences as variables. PSDDs can be viewed as a natural extension of sentential decision diagrams

[2] However we still have our restriction that $N > max_{a\in A} c(a) = max(c(U), c(\neg U)) = max(0.5, 1) = 1$, so the value is not completely arbitrary.

(SDDs), which represent a theory, by the use of extra parameters that define the probability of any assignment of values to the variables [2]. SDDs in turn are based on vtrees which are binary trees with each leaf corresponding to a different variable.

We now provide formal definitions of each of these three structures. In the next subsection corresponding diagrams are given using our example from Section 2.2.2 in order to help demonstrate the overall construction of PSDDs from SDDs, and SDDs from vtrees. Please note that the first two definitions are taken directly from [14], while the final definition is taken from [2]. As with the earlier definitions, we have made a few slight changes to the notation for the sake of clarity and for ease of integration with the BF later on.

Definition 2.3.1. A **vtree** V for a set of variables $\mathcal{X}$ is a full binary tree (a binary tree in which every non-leaf node has exactly two children) whose leaves are in a one-to-one correspondence with the variables in $\mathcal{X}$.

When learning the structure of PSDDs from data, learning an appropriate vtree is critical as each split in the tree represents an independence assumption between the variables in the left sub-vtree and the right sub-vtree that allows us to factorise the final distribution expressed by the PSDD. In the PSDD learning algorithm used in our work these splits are made by calculating the average pairwise mutual information between the variables (this is explained in greater detail as part of Section 4.1.2).

A second important fact to note is that when basing PSDDs upon vtrees, whether a subset of variables is in a left sub-vtree or a right sub-vtree is also critical, and so in some sense the vtree is asymmetric. This is explained immediately following Definition 2.3.3, which is required in order to appreciate this subtlety. Finally, the structure of the vtree can impact not just the quality of the learnt distribution but the size of the PSDD. This is because how compactly we are able to capture a set of logical constraints depends on how we split the variables in order to construct these constraints.

Definition 2.3.2. S is an **SDD** that is normalised for a vtree V over variables $\mathcal{X}$ if and only if one of the following holds:

- S is a **terminal node** such that $S = \top$ or $S = \bot$
- S is a terminal node such that $S = X$ or $S = \neg X$ and V is a leaf node corresponding to variable X

- S is a **decision node** $(p_1,s_1),...,(p_k,s_k)$ where **primes** $p_1,...,p_k$ are SDDs corresponding to the left sub-vtree of V, **subs** $s_1,...,s_k$ are SDDs corresponding to the right sub-vtree of V, and $p_1,...,p_k$ form a **partition** (the primes are consistent, mutually exclusive, and their disjunction $p_1 \vee ... \vee p_k$ is valid)

We refer to each (p_1,s_1) as an **element** of a decision node. Each terminal node corresponds to its literal or truth symbol and each decision node $(p_1,s_1),...,(p_k,s_k)$ corresponds to the sentence $\bigvee_{i=1}^{k}(p_i \wedge s_i)$. S represents a theory (which can be viewed as a set of logical constraints) in that the root of S evaluates to true if and only if the assignment of values to the variables in $\mathcal{X}$ are consistent with that theory.

Note that in an SDD (and therefore in a PSDD), for any possible assignment of values $\mathbf{X}$ to the variables $\mathcal{X}$ that the SDD ranges over, at each decision node $(p_1,s_1),...,(p_k,s_k)$ at most one prime p_i evaluates to true. In fact, though not strictly necessary, we also make the simplifying assumption that at least one (and thus exactly one) prime evaluates to true for any possible assignment. For such an assignment $\mathbf{X}$ we write $\mathbf{X} \models p_i$. Further, for any decision node corresponding to a node v in the vtree, the variables $\mathcal{X}_l$ under the left sub-vtree and the variables $\mathcal{X}_r$ under the right sub-vtree partition the set of variables $\mathcal{X}$ in the vtree rooted at v, and hence the primes $p_1,...,p_k$ are sentences over $\mathcal{X}_l$ and the subs $s_1,...,s_k$ are sentences over $\mathcal{X}_r$.

Definition 2.3.3. A **PSDD** P is a normalised SDD S (for some vtree V) with the following parameters:

- For each decision node $(p_1,s_1),...,(p_k,s_k)$ and prime p_i a non-negative parameter θ_i such that $\sum_{i=1}^{k}\theta_i = 1$ and $\theta_i = 0$ if and only if $s_i = \bot$
- For each terminal node $\top$ a parameter θ such that $0 < \theta < 1$ (this node is hence denoted as $X : \theta$ where X is the variable of the vtree node that $\top$ is normalised for)

These parameters then describe the probability distribution over the SDD theory as follows. For each node n in P, normalised for some vtree node v in V, we have a distribution $\mathbf{p}_n$ over the set of variables $\mathcal{X}$ in the vtree rooted at v where:

- If n is a terminal node and v has variable X:

n	$\mathbf{p}_n(X)$	$\mathbf{p}_n(\neg X)$
X	1	0
$\neg X$	0	1
$X:\theta$	θ	$1-\theta$
$\bot$	0	0

- If n is a decision node $(p_1, s_1), ..., (p_k, s_k)$ with parameters $\theta_1, ..., \theta_k$ and v has variables $\mathcal{X}_l$ in its left sub-vtree and variables $\mathcal{X}_r$ in its right sub-vtree:

$$\begin{aligned}\mathbf{p}_n(\mathbf{X}_l, \mathbf{X}_r) &= \sum_i \theta_i \mathbf{p}_{p_i}(\mathbf{X}_l)\mathbf{p}_{s_i}(\mathbf{X}_r) \\ &= \theta_j \mathbf{p}_{p_j}(\mathbf{X}_l)\mathbf{p}_{s_j}(\mathbf{X}_r)\end{aligned}$$

  for the single j such that $\mathbf{X}_l \models p_j$

It is important to observe the asymmetry between primes and subs here. In particular it follows that the direction of each branch in the vtree upon which the PSDD is based is not arbitrary and in fact influences the final probability distribution represented by the PSDD, as alluded to above. We also note, as proven in [2], that for any node n, each parameter θ_i or θ has an intuitive semantic meaning both locally (in the distribution $\mathbf{p}_n$ represented by the sub-PSDD rooted at node n) and globally (in the distribution $\mathbf{p}$ represented by the full PSDD). Namely, consider the notion of a context $\gamma_n = p_1 \wedge ... \wedge p_k$, a conjunction of the primes that are traversed from the root of the PSDD P to a node n. Then if γ_n has strictly positive probability we have that $\mathbf{p}_n(\cdot) = \mathbf{p}(\cdot|\gamma_n)$. From this result and Definition 2.3.3 it follows immediately that:

- If n is a terminal node $X : \theta$, then $\theta = \mathbf{p}_n(X) = \mathbf{p}(X|\gamma_n)$
- If n is a decision node $(p_1, s_1), ..., (p_k, s_k)$ then $\theta_i = \mathbf{p}_n(p_i) = \mathbf{p}(p_i|\gamma_n)$ for each $i \in \{1, ..., k\}$

2.3.2 Continuing Our Example

In order to illustrate the definitions from the previous subsection we return to our example from Section 2.2.2. Here we consider four variables R, U, L, and W. We also have two constraints on these variables which exclude some worlds from our set of possible worlds by assigning them a probability of 0:

- $W \leftrightarrow (R \wedge \neg U)$ says that the woman arrives to work wet from the rain if and only if it is raining and she did not go back for her umbrella
- $\neg U \rightarrow \neg L$ says that if the woman does not go back for her umbrella then she will not be late to work

We use the same probabilities as before (a 0.5 probability that it rains and a 0.5 probability that the woman will be late to work if she goes back to collect her umbrella)

and suppose (differently from before[3]) that in the case that it rains she goes back for her umbrella with probability 0.667, and in the case that it doesn't she goes back for her umbrella with probability 0.444. These parameters allow us to specify the full probability distribution, given in Table 2.1.

R	U	L	W	**p**(R,U,L,W)	Excluding Constraint(s)
1	1	1	1	0	$W \leftrightarrow (R \wedge \neg U)$
1	1	1	0	0.167	
1	1	0	1	0	$W \leftrightarrow (R \wedge \neg U)$
1	1	0	0	0.167	
1	0	1	1	0	$\neg U \rightarrow \neg L$
1	0	1	0	0	$W \leftrightarrow (R \wedge \neg U)$, $\neg U \rightarrow \neg L$
1	0	0	1	0.167	
1	0	0	0	0	$W \leftrightarrow (R \wedge \neg U)$
0	1	1	1	0	$W \leftrightarrow (R \wedge \neg U)$
0	1	1	0	0.111	
0	1	0	1	0	$W \leftrightarrow (R \wedge \neg U)$
0	1	0	0	0.111	
0	0	1	1	0	$W \leftrightarrow (R \wedge \neg U)$, $\neg U \rightarrow \neg L$
0	0	1	0	0	$\neg U \rightarrow \neg L$
0	0	0	1	0	$W \leftrightarrow (R \wedge \neg U)$
0	0	0	0	0.278	

Table 2.1: The probability distribution over possible worlds in our example, together with a note of any constraints that force some possible worlds to have probability 0.

To represent this distribution as a PSDD we first form a vtree V, as can be seen in Figure 2.1. In this diagram the numbers attached to each node are simply a labelling convention. We may then encode our two logical constraints, $W \leftrightarrow (R \wedge \neg U)$ and $\neg U \rightarrow \neg L$, into an SDD S normalised for V, which is presented in Figure 2.2.

[3]Note here that we clearly do not assume the perfectly rational behaviour of a classical utility-maximising agent.

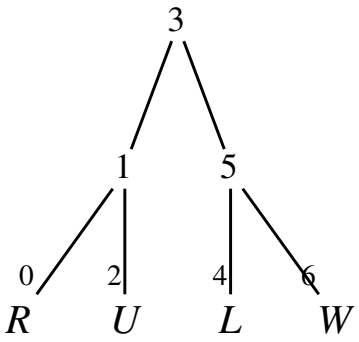

Figure 2.1: A vtree V over the set of variables $\{R, U, L, W\}$ in our example.

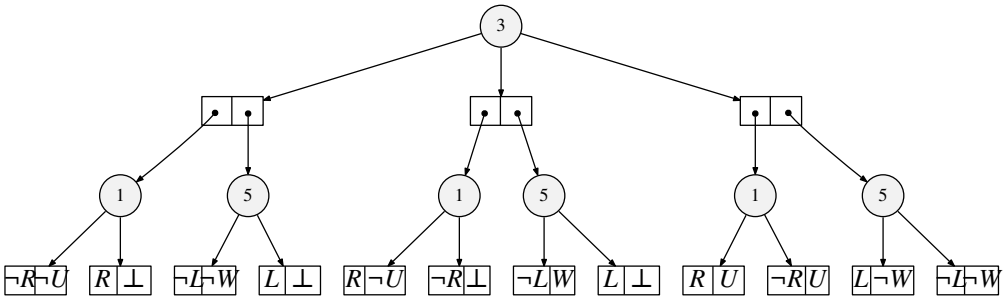

Figure 2.2: An SDD S normalised for the vtree above that represents the logical constraints $W \leftrightarrow (R \wedge \neg U)$ and $\neg U \rightarrow \neg L$ from our example.

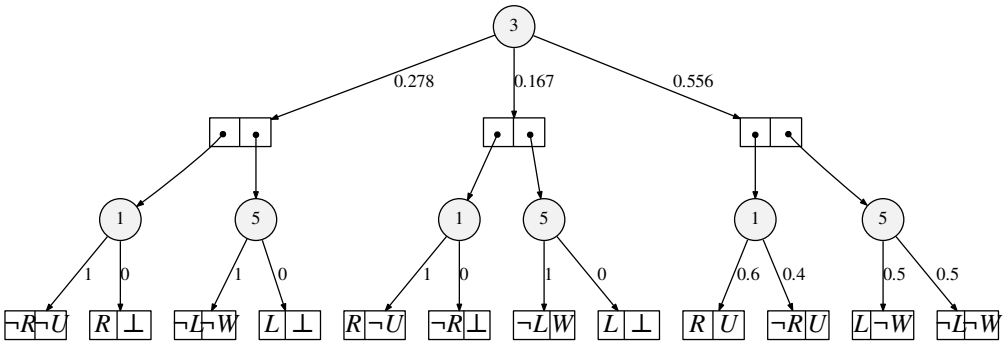

Figure 2.3: A PSDD P encoding all features of the example as defined above.

Here the boxed nodes represent primes and the circular nodes represent decisions, each decision node corresponding to a node in V. For example, the lower-left-most

prime node represents $\neg R \wedge \neg U$ and the lower-left-most decision node (labelled with a 1) represents $(\neg R \wedge \neg U) \vee (\neg R \wedge \bot)$. In fact, because $\neg R \wedge \bot$ is always false, we can reduce the sentence this node represents to simply $\neg R \wedge \neg U$. Hence another way to visualise an SDD is with boxes as conjunctions and circles as disjunctions. The final step is to parameterise S to form a PSDD P that represents the full distribution in the table above, as shown in Figure 2.3.

First observe that as we had no $\top$ terminal nodes in S, none of the terminal nodes in P are directly parameterised. We may also note that our parameters do indeed correspond to the semantics provided in the Section 2.3.1. For example, consider the edge labelled 0.6 which we will denote as θ. Here we have that:

$$\begin{aligned}
\theta &= \mathbf{p}\big(R|(R \wedge U) \vee (\neg R \wedge U)\big) \\
&= \mathbf{p}(R|U) \\
&= \frac{\mathbf{p}(U|R)\mathbf{p}(R)}{\mathbf{p}(U)} \\
&= \frac{\mathbf{p}(U|R)\mathbf{p}(R)}{\mathbf{p}(U|R)\mathbf{p}(R) + \mathbf{p}(U|\neg R)\mathbf{p}(\neg R)} \\
&= \frac{0.667 \times 0.5}{(0.667 \times 0.5) + (0.444 \times 0.5)} \\
&= 0.6
\end{aligned}$$

To conclude this subsection we show how P is evaluated in order to compute the probability of a certain assignment of values to the variables (which can be checked against the original value in Table 2.1). Here it is useful to visualise the diagram differently again, with boxes as products of their two elements and circles as sums of their children weighted by their respective edge labels. Now let us compute the probability that it is raining and that the woman goes back for her umbrella but is not late to work; $R = 1$, $U = 1$, $L = 0$, and $W = 0$. We evaluate P by starting at the leaf nodes and eventually returning the value at the root:

$$\begin{aligned}
\mathbf{p}(1,1,0,0) &= 0.278\big([1(0 \times 0) + 0(1 \times 0)] \times [1(1 \times 1) + 0(0 \times 0)]\big) \\
&\quad + 0.167\big([1(1 \times 0) + 0(0 \times 0)] \times [1(1 \times 0) + 0(0 \times 0)]\big) \\
&\quad + 0.556\big([0.6(1 \times 1) + 0.4(0 \times 1)] \times [0.5(0 \times 1) + 0.5(1 \times 1)]\big) \\
&= 0.278(0 \times 1) + 0.167(0 \times 0) + 0.556(0.6 \times 0.5) \\
&= (0.278 \times 0) + (0.167 \times 0) + (0.556 \times 0.3) \\
&= 0.167
\end{aligned}$$

2.3.3 Model Features

As highlighted in [2] we can think of PSDDs both in terms of knowledge representation models such as binary decision diagrams (BDDs) [15] and their various extensions including SDDs, or as members of the class of TPMs. Here we concentrate on the latter perspective. PSDDs can be reduced to sum-product network (SPNs) which are deep generative models whose creation was motivated by the desire to directly represent the partition function of a graphical model (the evaluation of which is generally intractable) [16]. SPNs in turn can be viewed as a type of arithmetic circuit (AC), one of the earliest proposed TPMs [17].

There are many subtle variations within the class of TPMs, and PSDDs are one of the more restrictive formalisms. However, these extra restrictions result in guarantees of many of the desirable properties outlined in this section. Using the terminology from [18], PSDDs possess the following two characteristics which together allow for generally tractable probabilistic reasoning:

- *Decomposability* (for every element (p_i, s_i) of a decision node, the set of variables of p_i and the set of variables of s_i are disjoint)
- *Determinism* (for any assignment of values to the variables each decision node has at most one element (p_i, s_i) that evaluates to true)

PSDDs support tractable evaluation in that computing the probability $\mathbf{p}(\mathbf{X})$ of any assignment of values $\mathbf{X}$ to the variables $\mathcal{X}$ can be done using a single bottom-up pass through the graph [2], equivalent to how our final calculation in Section 2.3.2 was made. This clearly takes time linear in the size of the graph, which we write as $O(|P|)$. Similarly, we can compute the probability of a partial assignment of values to the variables by marginalising the unset variables to give $\mathbf{p}(\mathbf{Y})$ where $\mathcal{Y} \subseteq \mathcal{X}$.

In practice this is done by simply setting any literal corresponding to an unassigned variable to the value 1, and thus we have tractable marginal inference. Combining the two facts above gives us tractable conditional inference as of course $\mathbf{p}(\mathbf{Y}|\mathbf{Z}) = \frac{\mathbf{p}(\mathbf{Y},\mathbf{Z})}{\mathbf{p}(\mathbf{Z})}$. Both of these latter tasks can also be performed in time $O(|P|)$. Given our need in the BF to compute probabilities of potentially large numbers of possible worlds, the tractability of these querying tasks will be crucial in the tractability of our final system.

The property of determinism ensures that computing the most probable evidence (MPE) is also tractable in PSDDs [18], where in general we have $MPE(\mathbf{Z}) = argmax_{\mathbf{Y}}\mathbf{p}(\mathbf{Y}|\mathbf{Z})$ such that $\mathcal{Y}$ and $\mathcal{Z}$ partition $\mathcal{X}$[4]. In practice the MPE is found by

[4]Note that we may have $\mathcal{Z} = \varnothing$.

setting any evidence variables to their assigned values, then using a Viterbi-style backtracking algorithm over the PSDD in which decision nodes are interpreted as representing maxes rather than sums. More details on the implementation of such an algorithm (which also runs in time $O(|P|)$) can be found in Section 4.2.2. Calculating the MPE will play an important role in allowing our models to not only represent decision-making scenarios but also to make optimal decisions within them and to efficiently find the most likely outcomes of decisions.

There are two primary ways in which PSDDs can be learnt. The first is to encode the underlying logical constraints as an SDD and then to learn the parameters from data (as demonstrated in the original paper on PSDDs, for example). Given a complete dataset (one in which all data points correspond to a full instantiation of the variables) the maximum likelihood estimates of all parameters can be calculated in closed form. Moreover, this can be done in time $O(|P|N)$ where $|P|$ is the size of the PSDD and N is the number of distinct datapoints in the dataset [2].

The second way to learn a PSDD is also to learn its structure from data, as initially proposed in [19]. The algorithms in the LearnPSDD package from this work allow one to learn PSDDs in both constrained and unconstrained probability spaces and to produce models that are competitive with the state-of-the-art both in terms of accuracy and size (which is used as a proxy for tractability). This is important if we want our system to learn a probability distribution and utility function from human-generated datasets.

Unlike other TPMs, it is straightforward to encode logical constraints into PSDDs. Firstly, this is theoretically desirable as it directly enforces sparsity in the model which in turn can lead to increased accuracy and decreased size. Secondly it is practical, as there are many situations in which we might want to specify underlying constraints such as laws, rules, or restrictions, or to incorporate background knowledge within the probability distribution. This means that we construct PSDDs using the same structural equations from the SEF, which can be encoded as propositional sentences and are in turn used to specify decision-making scenarios in the BF. Returning to Section 2.1.1, we can also draw a parallel between these logical constraints and deontological rules, and between learnt distributions over decision-making scenarios (which encode preferences) and the utility functions used in consequentialist ethical theories. Using PSDDs allows us to account for and combine both of these approaches.

Related to the fact that PSDDs can be used to encode logical constraints is the notion of explainability [20]. As well as the fact that we can query PSDDs over propo-

sitional sentences, which in turn can be used to represent many structured concepts, the parameters in the PSDD are also semantically meaningful (as explained in Section 2.3.1). This allows one to inspect and interpret parameters both locally and globally in order to more easily provide explanations or identify anomalies.

2.4 Related Work

The BF has its roots in an earlier attempt to define responsibility and blame by Chockler and Halpern [21]. This work is notable in that it is the first to treat responsibility and blame as admitting of degrees that can be directly computed. Similarly to the BF it also relies on the SEF from [12] and the notion of counterfactuals, although it differs in that it does not take into account any notion of utility.

Another influence on the BF is the model of intention inference proposed by Kleiman-Weiner et al. in [22]. It uses inverse rational planning to make quantifiable predictions about the moral permissibility of actions, which are then compared against judgements collected from humans. The models used take the form of influence diagrams (IDs), a variation on Bayesian networks, although their relatively small size does not give rise to tractability concerns. In a related paper [23] by the same lead author, a theoretical model of moral learning is proposed in order to explain the learning and cognitive development of moral theories in children. Here again, as in our own work, the model relies critically on the combination of utility functions and probabilistic inference.

There has been relatively little work done on using TPMs for decision-making, the two noteworthy exceptions being decision circuits (DCs) and sum-product-max networks (SPMNs), both of which are reducible to each other. DCs are a generalisation of ACs and can be used to tractably evaluate IDs, although there are currently no algorithms for learning them directly from data [24]. Similarly, SPMNs are a variation on SPNs that also include utility nodes and max nodes with children that correspond to decisions [25], although unlike DCs SPMNs can be learnt from data. However, neither of these models guarantees tractable MPE calculation or supports the direct encoding of logical principles as in PSDDs.

In [26] Choi et al. employ PSDDs to learn distributions over preference rankings, in a work not dissimilar to our own. The main distinction is that the variables in this distribution are the positions of items within preference lists, as opposed to the items themselves. Their model also does not take account of different preferences resulting from different contexts.

An important part of learning a model of moral decision-making is in learning a utility function. This topic is a relatively large area that is typically known as inverse reinforcement learning (IRL) [27] or Bayesian inverse planning [28]. Here the general problem is to learn an agent's utility function given (possibly partial) observations of the agent's behaviour in a variety of settings for which one has an existing model. In the vast majority of the literature this task is framed in terms of Markov decision processes (MDPs) and reward functions.

Although the core ideas also apply to our work, many of the algorithms that have been developed in this vein are not particularly compatible with PSDDs, or PGMs in general for that matter. Instead, we make some simplifying assumptions about the nature of the utility function (as explained in the following chapter) and use techniques more similar to those presented in [29]. This paper also includes ideas such as those discussed more generally and recently by Evans et al. in [30], concerning the problem of learning utility functions from behaviour that is inconsistent or subject to biases.

An excellent high-level overview of strategies in creating moral decision-making frameworks for AI is given by Conitzer et al. in [31]. They look at how game-theoretic approaches and machine learning can each be used in making moral decisions and make some tentative suggestions as to how they might be combined to this end in future work. Greene et al. also consider the broad problem of how to embed ethical principles into decision-making AI systems, with a particular focus on decisions that are made alongside other intelligent agents, be they human or artificial [32]. Their work is largely based on the multi-agent setting and as such they also bring in ideas from the important topics of computational social choice and preference aggregation. Despite their relevance these subjects are beyond the scope of our current work, although this is something we may cover in future research (see Section 6.2 for a discussion of possible extensions to this project).

Aside from the more theoretical and broad discussions referred to above, there have also been concrete attempts to build in notions of moral responsibility to particular tools. Both Dehghani et al. and Mao and Gratch use natural language processing to learn rule-based systems that are designed to capture human responses to moral decision-making scenarios and then to either make decisions or pass judgements accordingly [33; 34]. However, neither of these works attempts to use a formal definition of blameworthiness, and their reliance on data consisting of (simplified) natural language descriptions of moral decision-making scenarios is neither necessary nor possible for many cases we might want to model. Further, the rule-based nature of these

tools may result in less flexibility and robustness than a data-driven approach such as the learning of PSDDs.

Some of the most closely related work to this project has been conducted recently with the aid of MIT's Moral Machine website [35] and the data it has been used to gather. On the website participants are invited to submit moral judgements on what an autonomous vehicle with a brake failure should do when faced with two bad outcomes (such as running over some pedestrians or crashing and thus killing those inside the vehicle).

Kim et al. build on the theory in [23] by developing a computational model of moral decision-making whose predictions they test against Moral Machine data [36]. A key feature of the model is its hierarchical Bayesian nature; decisions are dependent on the individual's moral principles which are in turn dependent on the group that the individual belongs to. The learning of such moral principles (which take the form of weights on abstract concepts) is also something we attempt to achieve using PSDDs, although by modelling more complex decision-making scenarios we also have the option of learning context-relative principles (for example, one should never hit someone unless it is in self-defence).

In [37] a more straightforward attempt is taken in that the authors develop a method of aggregating the preferences of all participants in order to make a given decision (once again, this relies on the field of computational social choice). However, due to the large numbers of such orderings, tractability issues arise and so sampling must be used. Further, directly learning preferences like this does not allow us to specify or guarantee any restrictions. The use of PSDDs in our system solves both of these issues.

Chapter 3

Theoretical Framework

The primary theoretical contribution of the project comes in the form of a mapping from the BF to the PSDD framework (both of which were described in the previous chapter) that we then use as the basis for our implemented system. After presenting this mapping we provide a justification of our various assumptions and choices, a proof of correctness, and several complexity results.

3.1 Mapping

The first thing we do is distinguish between the cases in which we do and do not model outcome variables. In both cases we have exogenous (or random) variables $\mathcal{X}$, but in the former our endogenous variables $\mathcal{V}$ are partitioned into decision variables $\mathcal{D}$ (to which each action variable A belongs) and outcome variables O, and in the latter we have $\mathcal{V} = \mathcal{D} = O$. Observe that in this second case it is trivially true that $O = \mathcal{D} \cup O$, allowing us to use the same definitions and notation for both cases in the rest of the mapping.

The range function $\mathcal{R}$ is defined by the scenario we model, though in practice we one-hot encode the variables and so the range of each is simply $\{0, 1\}$. As a brief aside we note that the ability to include logical constraints in a PSDD means we can prevent the learning of models in which more than one of the one-hot variables is active at any time, and our learnt distributions face absolutely no decrease in quality. A subset (possibly empty) of the structural equations in $\mathcal{F}$ are implicitly encoded within the structure of the SDD underlying the PSDD and consist of the propositional formulae that remain true in every causal model M.

The remaining equations are those that vary depending on the causal model. Each

possible setting of the variables in $\mathcal{D} \cup O$ given a context $\mathbf{X}$ corresponds to a set of structural equations that combine with those encoded by the SDD to determine the values of these variables given $\mathbf{X}$. In other words, the structural equations encoded in the SDD do not necessarily constrain the SDD to one satisfying assignment given a setting of the random variables, but to a set of possible satisfying assignments. Hence in general the probability of a causal formula $\psi = [\mathcal{Y} \leftarrow \mathbf{Y}]\varphi$ is real-valued in the range $[0,1]$ instead of simply equal to 1 or 0 (true or false). Therefore we no longer write $(M,\mathbf{X}) \models \psi$ and so we do not use $[\![\psi]\!]$ in our definitions either.

Instead of our distribution $\mathbf{p}$ ranging over the set of causal settings $\mathcal{K}$, $\mathbf{p}$ ranges over all variables instead, $\mathbf{p} : \times_{Y \in \mathcal{X} \cup \mathcal{D} \cup O} \mathcal{R}(Y) \rightarrow [0,1]$. To simplify notation (and remain faithful to our later implementation) we view a Boolean formula of primitive events (possibly resulting from decision A) as a function $\varphi : \times_{Y \in O \cup \mathcal{D} \setminus A} \mathcal{R}(Y) \rightarrow \{0,1\}$ that returns 1 if the original formula is true or 0 otherwise. Hence for the probability of φ occurring given that action A is performed we use $\sum_{\mathbf{Y} \in \times_{Y \in O \cup \mathcal{D} \setminus A} \mathcal{R}(Y)} \varphi(\mathbf{Y})\mathbf{p}(\mathbf{Y}|a)$ instead of $\sum_{(M,\mathbf{X}) \in [\![[A=a]\varphi]\!]} \mathbf{p}(M,\mathbf{X})$. In natural language, both of these quantities represent the sum of the probabilities (across all possible contexts) of all the outcomes satisfying φ, given that decision $A = a$ is made (see Section 3.3 for the proof). This is what we then use to calculate $\delta_{a,a',\varphi}$. Here we are implicitly summing over the probabilities of the contexts $\mathbf{X}$ given in the model. In order not to re-learn a separate model for each scenario we also allow the user of our system the option of specifying a particular current distribution on contexts $\mathbf{q}$. In this case the expression above becomes $\sum_{\mathbf{X} \in \times_{Y \in \mathcal{X}} \mathcal{R}(Y)} \sum_{\mathbf{Y} \in \times_{Y \in O \cup \mathcal{D} \setminus A} \mathcal{R}(Y)} \varphi(\mathbf{Y})\mathbf{p}(\mathbf{Y}|\mathbf{X},a)\mathbf{q}(\mathbf{X})$, which reduces to the standard case when $\mathbf{q} = \mathbf{p}$.

We now consider the utility function $\mathbf{u}$, the output of which we assume is normalised to the range $[0,1]$. In our implementation we give the option for the user to input an existing utility function or to learn one from the decision-making data. We avoid unnecessary extra notation here by defining the utility function in terms of contexts $\mathbf{X}$, decisions $\mathbf{D}$, and outcomes $\mathbf{O} = (O_1, \ldots O_n)$ instead of worlds w as in the BF. When learning a utility function we allow the user to specify whether or not the function should be context-relative, in other words whether we should have $\mathbf{u}(\mathbf{O})$ or $\mathbf{u}(\mathbf{O}|\mathbf{X})$. We also ask whether the function should be linear in the outcome variables (as long as $\mathcal{D} \neq O$), in which case the final utility is $\mathbf{u}(\mathbf{O}|\mathbf{X}) = \sum_i \mathbf{u}_i(O_i|\mathbf{X})$ or $\mathbf{u}(\mathbf{O}) = \sum_i \mathbf{u}_i(O_i)$ respectively (where we assume that each $\mathbf{u}_i(O_i) \geq 0$).

For each of these possible combinations the key assumption we make (before normalisation) is that the probability of a certain decision given a context is linearly pro-

portional to the expected utility resulting from that decision. Specifically, for each decision $\mathbf{D}$ and context $\mathbf{X}$ we make the assumptions given in Table 3.1[1]. Our not including assumptions in two of the cases is explained and justified in Section 3.2.2.

Outcome Variables	Linear	Context-Relative	Assumption
✓	✓	✓	$\mathbf{p}(\mathbf{D}\|\mathbf{X}) \propto \sum_i \mathbf{u}_i(O_i\|\mathbf{X})\mathbf{p}(O_i\|\mathbf{D},\mathbf{X})$
✓	✓	✗	$\mathbf{p}(\mathbf{D}\|\mathbf{X}) \propto \sum_i \mathbf{u}_i(O_i)\mathbf{p}(O_i\|\mathbf{D},\mathbf{X})$
✓	✗	✓	$\mathbf{p}(\mathbf{D}\|\mathbf{X}) \propto \sum_{\mathbf{O}} \mathbf{u}(\mathbf{O}\|\mathbf{X})\mathbf{p}(\mathbf{O}\|\mathbf{D},\mathbf{X})$
✓	✗	✗	$\mathbf{p}(\mathbf{D}\|\mathbf{X}) \propto \sum_{\mathbf{O}} \mathbf{u}(\mathbf{O})\mathbf{p}(\mathbf{O}\|\mathbf{D},\mathbf{X})$
✗	✓	✓	N/A
✗	✓	✗	N/A
✗	✗	✓	$\mathbf{p}(\mathbf{D}\|\mathbf{X}) \propto \mathbf{u}(\mathbf{O}\|\mathbf{X}) = \mathbf{u}(\mathbf{D}\|\mathbf{X})$
✗	✗	✗	$\mathbf{p}(\mathbf{D}\|\mathbf{X}) \propto \mathbf{u}(\mathbf{O}) = \mathbf{u}(\mathbf{D})$

Table 3.1: The six different instantiations of our assumption that the probability of a decision being made in a context is linearly proportional to the expected utility of that decision in that context.

Our slightly different approach to the definition of a utility function means we also need to adapt the cost function given in the BF. As actions do not deterministically lead to outcomes in our work, we cannot use $\mathbf{O}_{A\leftarrow a}$ to represent the specific outcome when decision a is made (in some context). For our purposes it suffices to use $c(a) = -\sum_{\mathbf{O},\mathbf{X}} \mathbf{u}(\mathbf{O}|\mathbf{X})\mathbf{p}(\mathbf{O},\mathbf{X}|a)$ or $c(a) = -\sum_{\mathbf{O}} \mathbf{u}(\mathbf{O})\mathbf{p}(\mathbf{O}|a)$ (depending on whether $\mathbf{u}$ is context-relative or not). This is just the negative expected utility over all contexts given that decision $A = a$ is made. A proof of equivalence of this cost function to the one in the BF (with respect to the task of determining blameworthiness scores) is given in Section 3.3.

Again, we also give the user the option of updating the distribution over contexts so that the current model can be re-used. In which case we have either $c(a) = -\sum_{\mathbf{O},\mathbf{X}} \mathbf{u}(\mathbf{O}|\mathbf{X})\mathbf{p}(\mathbf{O}|\mathbf{X},a)\mathbf{q}(\mathbf{X})$ or $c(a) = -\sum_{\mathbf{O},\mathbf{X}} \mathbf{u}(\mathbf{O})\mathbf{p}(\mathbf{O}|\mathbf{X},a)\mathbf{q}(\mathbf{X})$, both of which reduce to the previous expressions when $\mathbf{q} = \mathbf{p}$. Given c, both $db_N(a,a',\varphi)$ and $db_N(a,\varphi)$ are defined as in the BF, although we instead require that $N > -min_{a\in A}c(a)$ (the equivalence of this condition to the original is also shown in Section 3.3).

[1]Please note that for reasons of clarity and brevity we write $\sum_{\mathbf{O}}$ instead of $\sum_{\mathbf{Y}\in\times_{Y\in O}\mathcal{R}(Y)}$ in the table as well as in the remainder of this section, where we also write $\sum_{\mathbf{X}}$ instead of $\sum_{\mathbf{Y}\in\times_{Y\in\mathcal{X}}\mathcal{R}(Y)}$.

3.2 Assumptions and Choices

3.2.1 Representing Scenarios

Our first assumption is that outcome variables may not always be modelled, or in other words we have simply $O = \mathcal{D}$. The practical justification behind this is that we want to learn our models from decision-making data, and while we assume that this includes descriptions of the context and the decision(s) made we do not necessarily want to assume that whomever or whatever recorded this data was also able to record all the outcomes and repercussions of the decision(s). The more theoretical reason is that sometimes it makes sense to think of outcomes as being deterministic consequences of decisions, and thus the decisions being an end in themselves.

Secondly, instead of using a different SDD for each model M, the SDD represents the set of all possible models. The PSDD formed by parameterising the SDD then corresponds to the probability distribution over $\mathcal{K}$, as it is the structural equations that define the values of the endogenous variables in terms of the exogenous variables. Hence by using PSDDs we compact all this information into a single model. The critical assumption here is that the signature $\mathcal{S}$ remains the same in all models (although the structural equations $\mathcal{F}$ may vary). In simpler terms, we assume that the variables and the values that they may take remain the same in all models, but that the ways in which they are related, and the likelihoods of them taking certain values may change.

In general, given that the PSDD represents an agent's uncertain view of a particular decision-making scenario we do not think it too restrictive to keep the elements of this scenario the same across the potential eventualities as long as the way these elements interact may differ. Indeed, to learn a PSDD from decision-making data requires that the datapoints measure the same variables each time.

3.2.2 Learning Utility Functions

Normalising our utility function to the range $[0,1]$ has no effect on our calculations as we only use functions with bounded ranges. Hence, if there are negative utilities given, we can simply subtract the minimum utility value before dividing by the maximum utility (unless all utilities are the same, in which case we set them all equal to 1). This is a positive affine transformation and so preserves both the ordinality and relative cardinality of all utility values. We do not allow the possibility of utility functions that are linear in decision variables because in this situation the utility-maximising choice

would simply be to make as many decisions as possible (setting as many decision variables to 1 as possible) which clearly does not make sense in the vast majority of decision-making scenarios.

The option of linearity in the outcome variables is appropriate for cases in which the occurrence of another outcome variable either never decreases utility (in which case the variables should be encoded as 1 if they occur and 0 otherwise) or never increases utility (in which case the variables should be encoded as 0 if they occur and 1 otherwise). In other words the utility function is simply a vector of weights and the total utility of an outcome is the dot product of this vector with the vector of outcome variables.

For example, consider a decision-making scenario in which one is attempting to rescue a group of people and each outcome variable corresponds to a particular person being saved. Then the total utility can be measured as the sum of the outcome variables, each weighted by the utility of saving that respective person. However, there are also cases where this is not appropriate. Perhaps, in the case above, the people are being rescued by a life raft but if too many people are brought on board, the raft will sink.

Similarly, in many scenarios, how good a certain outcome $\mathbf{O}$ is depends on the context $\mathbf{X}$. Swerving off the road and crashing one's car in order to avoid a pedestrian could typically be a decision that leads to the highest expected utility, but perhaps in a context in which the pedestrian is recklessly crossing the road on a red light it is better to make a different decision. However, if we model outcomes as context-relative, then when learning a utility function from decision-making data, without further information we can only learn how good decisions are relative to other decisions within the same context.

To illustrate this point, consider decision-making data with two contexts, $\mathbf{X}_1$ and $\mathbf{X}_2$, each occurring five times. Now suppose that an agent makes decisions $\mathbf{D}_{1a}$ four times and decisions $\mathbf{D}_{1b}$ once in context $\mathbf{X}_1$ and similarly for decisions $\mathbf{D}_{2a}$ and $\mathbf{D}_{2b}$ in context $\mathbf{X}_2$. Then (assuming the agent is rational) we can infer that the expected utility from $\mathbf{D}_{1a}$ in $\mathbf{X}_1$ is greater than that from $\mathbf{D}_{1b}$ and likewise for $\mathbf{D}_{2a}$ and $\mathbf{D}_{2b}$ in $\mathbf{X}_2$. However, it could be that these decisions result in expected utilities of 4 versus 1 and 400 versus 100 respectively; we have no way of comparing utilities across different contexts. When using context-relative utility functions in our system we therefore normalise utility values over contexts instead of overall, so that the maximum utility in any context is 1.

The final assumption to be justified, and perhaps the most significant, is that the

probability of a certain decision given a context is linearly proportional to the expected utility resulting from that decision. We begin by breaking this assumption down into two claims. The first is that there exists such a proportionality relationship (linear or otherwise) between these two quantities. This, we believe, is relatively uncontroversial and can be restated as the simple principle that an agent is more likely to choose a decision that leads to a higher (expected) utility than one that leads to a lower (expected) utility. If we view decisions as guided by preferences which are in turn based on a utility function, then it follows that the decisions should, on average, be consistent with and representative of the utility function.

The second claim is that this relationship is in fact linear. This simplification is far less critical to our work than the first claim, and its validity rests on how we should model the decision-maker from whom the data has been gathered. Clearly a rational, utility-maximising agent who is informed and unimpaired would only ever make one decision (the one with the highest expected utility), but this idealisation is unlikely to be reflected in real-world data and so we must decide how to account for this. We may also need to create models from data generated by multiple agents whose preferences and decisions may well not be the same.

A *naive* agent or one who is impaired would be more likely to make worse decisions (that result in below-average expected utility across all decisions). This would be reflected by a non-linear relationship in which the expected utility from decisions made more often would in fact be relatively much higher than that from decisions made less often. Conversely, a *sophisticated* agent or one with little or no impairment would be more likely to make better decisions (ones that result in above-average expected utility across all decisions). This would again be reflected by a non-linear relationship, but here the variation in decisions made is accounted for by the decisions having relatively close expected utilities and so the agent has less incentive to choose the utility-maximising decision. By using a linear relationship instead we simplify our calculations and avoid making any assumption about the agent's naivety or sophistication.

This can perhaps best be seen diagrammatically, as in Figure 3.1. Consider a scenario in which an agent makes decision D_1 with probability 0.5, D_2 with probability 0.3, and D_3 with probability 0.2. In the case of a standard agent, simplifying our notation from earlier, we have $\mathbf{p}(D) \propto \mathbf{u}(D)$ and so the decisions have expected utilities 1, 0.6, and 0.4 respectively (recall that we normalise the function to have maximum value 1).

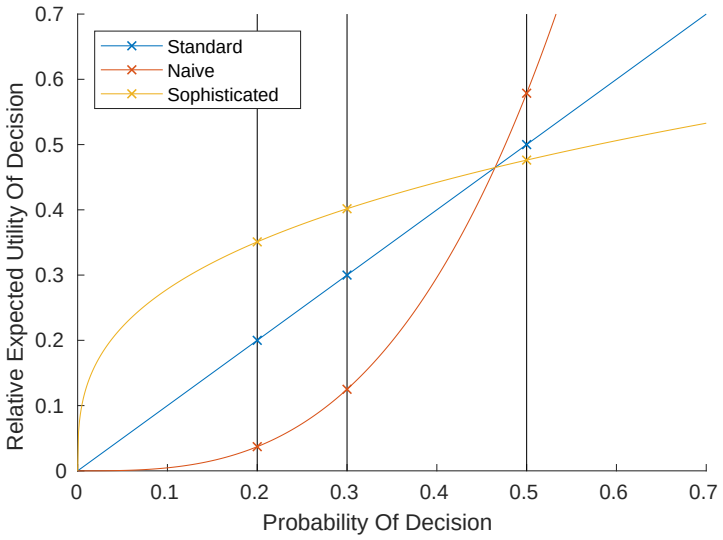

Figure 3.1: Examples of possible relationships between the probability of a decision and the (relative) expected utility resulting from that decision.

In the case of a naive agent then the probability of choosing a decision with lower expected utility is higher. We model this using a concave function (on [0,1]), such as $(\frac{\mathbf{p}(D)}{0.6})^3 \propto \mathbf{u}(D)$, in which case the decisions have expected utilities 1, 0.216, and 0.064 respectively. In the case of a sophisticated agent on the other hand, we use a convex function $0.6\mathbf{p}(D)^{\frac{1}{3}} \propto \mathbf{u}(D)$ and obtain expected utilities 1, 0.843, and 0.737. An important point to conclude with is that having learnt a linearly proportional utility function, we can then change it to reflect these other assumptions by simply applying a function such as those above and then re-normalising afterwards. Thus our choice of a linear relationship imposes no real restrictions on our work.

3.3 Proof of Correctness

In this section we offer proof that our eventual implementation with PSDDs using the mapping above computes the same blameworthiness scores as in the BF. We do this by supplying two lemmas corresponding to the two key elements of the final blameworthiness formula, $\delta_{a,a',\varphi}$ and $c(a)$. Using these two results our final proof of correctness follows immediately.

Lemma 3.3.1. *The mapping in Section 3.1 computes the same value for* $\delta_{a,a',\varphi}$ *as in the BF.*

Proof. Recall that in the BF we have:

$$\delta_{a,a',\varphi} = max\Big(\Big[\sum_{(M,\mathbf{X})\in[\![[A=a]\varphi]\!]} \mathbf{p}(M,\mathbf{X}) - \sum_{(M,\mathbf{X})\in[\![[A=a']\varphi]\!]} \mathbf{p}(M,\mathbf{X})\Big], 0\Big)$$

And in our system we use:

$$\begin{aligned}\delta_{a,a',\varphi} &= max\Big(\Big[\sum_{\mathbf{Y}\in\times_{Y\in O\cup\mathcal{D}\setminus A}\mathcal{R}(Y)} \varphi(\mathbf{Y})\mathbf{p}(\mathbf{Y}|a) - \sum_{\mathbf{Y}\in\times_{Y\in O\cup\mathcal{D}\setminus A}\mathcal{R}(Y)} \varphi(\mathbf{Y})\mathbf{p}(\mathbf{Y}|a')\Big], 0\Big)\\ &= max\Big(\sum_{\mathbf{Y}\in\times_{Y\in O\cup\mathcal{D}\setminus A}\mathcal{R}(Y)} \varphi(\mathbf{Y})\big[\mathbf{p}(\mathbf{Y}|a) - \mathbf{p}(\mathbf{Y}|a')\big], 0\Big)\end{aligned}$$

Observe that the two probability distributions are defined differently; the former over the set of all possible causal models $\mathcal{K}$ and the latter over the values of all variables in the model $\times_{Y\in\mathcal{X}\cup\mathcal{D}\cup O}\mathcal{R}(Y)$. To clarify our working below we use $\mathbf{p}_{BF}$ for the original distribution. To further clarify matters, when summing over all possible assignments $\mathbf{Y}$ in $\times_{Y\in O\cup\mathcal{D}\setminus A}\mathcal{R}(Y)$ without restriction we simply write $\mathbf{Y}$. The proof of this lemma therefore reduces to showing that:

$$\sum_{(M,\mathbf{X})\in[\![[A=a]\varphi]\!]} \mathbf{p}_{BF}(M,\mathbf{X}) = \sum_{\mathbf{Y}} \varphi(\mathbf{Y})\mathbf{p}(\mathbf{Y}|a)$$

Using our assumption that the signature $\mathcal{S}$ of any causal model $M = (\mathcal{S},\mathcal{F})$ does not change, we can view $\mathbf{p}_{BF}(M,\mathbf{X})$ as $\mathbf{p}_{BF}(\mathcal{F},\mathbf{X})$. Each set of structural equations $\mathcal{F}$ together with a context $\mathbf{X}$ deterministically leads to a unique complete assignment $\mathbf{V}$ of the endogenous variables, which we write as $(\mathcal{F},\mathbf{X}) \models \mathbf{V}$, though there may be many such sets of equations that lead to the same assignment. Hence, for any (possibly partial) assignment $\mathbf{Y}$ of the endogenous variables we have:

$$\mathbf{p}(\mathbf{X},\mathbf{Y}) = \sum_{\mathcal{F}\in\{\mathcal{F}:(\mathcal{F},\mathbf{X})\models\mathbf{Y}\}} \mathbf{p}_{BF}(\mathcal{F},\mathbf{X})$$

In the notation from the BF $[\![[A=a]\varphi]\!]$ represents the set of all causal settings $(M,\mathbf{X})$ where $A=a$ causes φ. Here a cause is viewed as an implication: if $A=a$ is true then φ is true. Intuitively, if $A=a$ is part of $\mathbf{X}$ already then the question of whether $(M,\mathbf{X}) \in [\![[A=a]\varphi]\!]$ reduces to the question of whether $(M,\mathbf{X}) \in [\![\varphi]\!]$. Finally, recall that we view φ as function such that $\varphi(\mathbf{Y}) = 1$ if $\mathbf{Y}$ satisfies φ and $\varphi(\mathbf{Y}) = 0$ otherwise. With these two observations and the identity immediately above we may now complete

the proof as follows:

$$\begin{aligned}
\sum_{(M,\mathbf{X})\in[\![[A=a]\varphi]\!]} \mathbf{p}_{BF}(M,\mathbf{X}) &= \sum_{(\mathcal{F},\mathbf{X})\in[\![[A=a]\varphi]\!]} \mathbf{p}_{BF}(\mathcal{F},\mathbf{X}) \\
&= \sum_{(\mathcal{F},\mathbf{X})\in[\![\varphi]\!]} \mathbf{p}_{BF}(\mathcal{F},\mathbf{X}|a) \\
&= \sum_{\mathbf{Y}\in\{\mathbf{Y}:\varphi(\mathbf{Y})=1\}} \sum_{\mathbf{X}} \sum_{\mathcal{F}\in\{\mathcal{F}:(\mathcal{F},\mathbf{X})\models\mathbf{Y}\}} \mathbf{p}_{BF}(\mathcal{F},\mathbf{X}|a) \\
&= \sum_{\mathbf{Y}\in\{\mathbf{Y}:\varphi(\mathbf{Y})=1\}} \sum_{\mathbf{X}} \mathbf{p}(\mathbf{X},\mathbf{Y}|a) \\
&= \sum_{\mathbf{Y}\in\{\mathbf{Y}:\varphi(\mathbf{Y})=1\}} \mathbf{p}(\mathbf{Y}|a) \\
&= \sum_{\mathbf{Y}\in\{\mathbf{Y}:\varphi(\mathbf{Y})=1\}} \varphi(\mathbf{Y})\mathbf{p}(\mathbf{Y}|a) \\
&= \sum_{\mathbf{Y}\in\{\mathbf{Y}:\varphi(\mathbf{Y})=1\}} \varphi(\mathbf{Y})\mathbf{p}(\mathbf{Y}|a) + 0 \\
&= \sum_{\mathbf{Y}\in\{\mathbf{Y}:\varphi(\mathbf{Y})=1\}} \varphi(\mathbf{Y})\mathbf{p}(\mathbf{Y}|a) + \sum_{\mathbf{Y}\in\{\mathbf{Y}:\varphi(\mathbf{Y})=0\}} \varphi(\mathbf{Y})\mathbf{p}(\mathbf{Y}|a) \\
&= \sum_{\mathbf{Y}\in\{\mathbf{Y}:\varphi(\mathbf{Y})=1\}\cup\{\mathbf{Y}:\varphi(\mathbf{Y})=0\}} \varphi(\mathbf{Y})\mathbf{p}(\mathbf{Y}|a) \\
&= \sum_{\mathbf{Y}} \varphi(\mathbf{Y})\mathbf{p}(\mathbf{Y}|a)
\end{aligned}$$

□

Lemma 3.3.2. *The mapping in Section 3.1 computes the same value for $c(a') - c(a)$ as in the BF.*

Proof. Following the convention in the previous proof we refer to the original cost function as c_{BF} and our system's cost function as c. Recall that:

$$c_{BF}(a) = \sum_{(M,\mathbf{X})\in\mathcal{K}} \mathbf{p}_{BF}(M,\mathbf{X})\left[\mathbf{u}(w_{M,\mathbf{X}}) - \mathbf{u}(w_{M,O\leftarrow\mathbf{O}_{A\leftarrow a},\mathbf{X}})\right]$$

In our mapping we presented two possible cost functions depending on whether the utility function was context-relative or not. In this proof we assume the latter, but it can easily be extended to the context-relative case. In particular we have:

$$\begin{aligned}
c(a) &= -\sum_{\mathbf{O}\in\times_{Y\in O}\mathcal{R}(Y)} \mathbf{u}(\mathbf{O})\mathbf{p}(\mathbf{O}|a) \\
&= -\sum_{\mathbf{O}} \mathbf{u}(\mathbf{O})\mathbf{p}(\mathbf{O}|a)
\end{aligned}$$

where our notation is simplified as before. First, observe that:

$$\begin{aligned}
c_{BF}(a') - c_{BF}(a) &= \sum_{(M,\mathbf{X})\in\mathcal{K}} \mathbf{p}_{BF}(M,\mathbf{X})\left[\mathbf{u}(w_{M,\mathbf{X}}) - \mathbf{u}(w_{M,O\leftarrow\mathbf{O}_{A\leftarrow a'},\mathbf{X}})\right] \\
&\quad - \sum_{(M,\mathbf{X})\in\mathcal{K}} \mathbf{p}_{BF}(M,\mathbf{X})\left[\mathbf{u}(w_{M,\mathbf{X}}) - \mathbf{u}(w_{M,O\leftarrow\mathbf{O}_{A\leftarrow a},\mathbf{X}})\right] \\
&= \sum_{(M,\mathbf{X})\in\mathcal{K}} \mathbf{p}_{BF}(M,\mathbf{X})\mathbf{u}(w_{M,\mathbf{X}}) - \sum_{(M,\mathbf{X})\in\mathcal{K}} \mathbf{p}_{BF}(M,\mathbf{X})\mathbf{u}(w_{M,O\leftarrow\mathbf{O}_{A\leftarrow a'},\mathbf{X}}) \\
&\quad - \sum_{(M,\mathbf{X})\in\mathcal{K}} \mathbf{p}_{BF}(M,\mathbf{X})\mathbf{u}(w_{M,\mathbf{X}}) + \sum_{(M,\mathbf{X})\in\mathcal{K}} \mathbf{p}_{BF}(M,\mathbf{X})\mathbf{u}(w_{M,O\leftarrow\mathbf{O}_{A\leftarrow a},\mathbf{X}}) \\
&= \left[- \sum_{(M,\mathbf{X})\in\mathcal{K}} \mathbf{p}_{BF}(M,\mathbf{X})\mathbf{u}(w_{M,O\leftarrow\mathbf{O}_{A\leftarrow a'},\mathbf{X}}) \right] \\
&\quad - \left[- \sum_{(M,\mathbf{X})\in\mathcal{K}} \mathbf{p}_{BF}(M,\mathbf{X})\mathbf{u}(w_{M,O\leftarrow\mathbf{O}_{A\leftarrow a},\mathbf{X}}) \right]
\end{aligned}$$

As before, from now on we refer to $(\mathcal{F},\mathbf{X})$ instead of $(M,\mathbf{X})$, using our assumption that the signature $\mathcal{S}$ does not change. Hence our proof of the statement that $c_{BF}(a') - c_{BF}(a) = c(a') - c(a)$ reduces to our proving the following identity:

$$\sum_{(\mathcal{F},\mathbf{X})\in\mathcal{K}} \mathbf{p}_{BF}(\mathcal{F},\mathbf{X})\mathbf{u}(w_{\mathcal{F},O\leftarrow\mathbf{O}_{A\leftarrow a},\mathbf{X}}) = \sum_{\mathbf{O}} \mathbf{u}(\mathbf{O})\mathbf{p}(\mathbf{O}|a)$$

In the BF a world w is a complete assignment of values to the endogenous variables, but here (as in fact is implicitly assumed in the original work) $\mathbf{u}$ is a function of the values of the outcome variables $\mathbf{O} \in \times_{Y\in O}\mathcal{R}(Y)$ (which form a subset of the endogenous variables). To clarify notation let us thus write $\mathbf{u}(w_{\mathcal{F},O\leftarrow\mathbf{O}_{A\leftarrow a},\mathbf{X}})$ as $\mathbf{u}(\mathbf{O}_{\mathcal{F},\mathbf{X},a})$, and if $A = a$ is already part of $\mathbf{X}$ this simplifies to $\mathbf{u}(\mathbf{O}_{\mathcal{F},\mathbf{X}})$. Finally, recall our earlier identity between $\mathbf{p}_{BF}$ and $\mathbf{p}$, now applied to the case where $\mathbf{Y} = \mathbf{O}$:

$$\mathbf{p}(\mathbf{X},\mathbf{O}) = \sum_{\mathcal{F}\in\{\mathcal{F}:(\mathcal{F},\mathbf{X})\models\mathbf{O}\}} \mathbf{p}_{BF}(\mathcal{F},\mathbf{X})$$

Proceeding in much the same way as the previous proof, we now conclude:

$$\begin{aligned}
\sum_{(\mathcal{F},\mathbf{X})\in\mathcal{K}} \mathbf{p}_{BF}(\mathcal{F},\mathbf{X})\mathbf{u}(\mathbf{O}_{\mathcal{F},\mathbf{X},a}) &= \sum_{(\mathcal{F},\mathbf{X})\in\mathcal{K}} \mathbf{p}_{BF}(\mathcal{F},\mathbf{X}|a)\mathbf{u}(\mathbf{O}_{\mathcal{F},\mathbf{X}}) \\
&= \sum_{\mathbf{O}}\sum_{\mathbf{X}} \sum_{\mathcal{F}\in\{\mathcal{F}:(\mathcal{F},\mathbf{X})\models\mathbf{O}\}} \mathbf{p}_{BF}(\mathcal{F},\mathbf{X}|a)\mathbf{u}(\mathbf{O}) \\
&= \sum_{\mathbf{O}}\sum_{\mathbf{X}} \mathbf{p}(\mathbf{X},\mathbf{O}|a)\mathbf{u}(\mathbf{O}) \\
&= \sum_{\mathbf{O}} \mathbf{u}(\mathbf{O})\mathbf{p}(\mathbf{O}|a)
\end{aligned}$$

□

Theorem 3.3.3. *The mapping in Section 3.1 computes the same blameworthiness scores as in the BF.*

Proof. Recall that:

$$db_N(a,a',\varphi) = \delta_{a,a',\varphi}\frac{N - max(c(a') - c(a),0)}{N}$$

From the previous two lemmas we have that the quantities $\delta_{a,a',\varphi}$ and $c(a') - c(a)$ computed by our system have the same values as in the BF. Further, the original restriction $N > max_{a\in A}c_{BF}(a)$ on N was put in place to ensure that the numerator of $db_N(a,a',\varphi)$ is never negative. By inspection, the maximum value of of $c(a') - c(a)$ occurs when $c(a') = 0$ (as our costs are never positive) and when $-c(a)$ is at a maximum. Thus our new limit of $N > -min_{a\in A}c(a) = max_{a\in A} - c(a)$ imposes precisely the same restriction. Hence the value of $db_N(a,a',\varphi)$ computed by our system is the same as in the BF. As we have:

$$db_N(a,\varphi) = max_{a'\in\mathcal{R}(A)\setminus a}db_N(a,a',\varphi))$$

then we may also deduce that the the value of $db_N(a,\varphi)$ computed by our system is the same as in the BF, and thus that the mapping is correct. □

3.4 Complexity Results

Given that one of our original concerns was tractability, we conclude this chapter with several computational complexity results for our mapping. Basic results were given in the original paper, but only in terms of the computations being polynomial in $|M|$, $|\mathcal{K}|$, and $|\mathcal{R}(A)|$ [1]. Here we provide more detailed results that are specific to our mapping and to the properties of PSDDs. We begin by noting several key quantities:

- $O(|P|)$ is the time taken to evaluate the PSDD P. This is linear in the size of the PSDD ($|P|$, typically measured as the number of parameters in the PSDD) although we do not attempt to provide bounds on this latter quantity here save for the obvious fact that it is smaller than the tabular representation, $O(2^{|\mathcal{X}|+|\mathcal{D}|+|O|})$

- $O(U)$ is the time taken to evaluate the utility function. This could be constant if the function is stored as a data structure with constant time look-up (as is the case in our implementation, for example) but for our general complexity results we do not make this assumption

- $O(|\varphi|)$ is the time taken to evaluate the Boolean function φ representing the combination of primitive events we wish to consider when attributing blame. This is linear in $|\varphi|$ which measures the number of Boolean connectives in φ

- $O(Q)$ is the time taken to evaluate the alternative distribution $\mathbf{q}$ over contexts, if it is given. Again, this is constant time in our implementation (though of $O(2^{|\mathcal{X}|})$ space complexity) but we do not make any assumption about this in our results here

Aside from whether or not the user specifies an alternative distribution $\mathbf{q}$, the complexity of calculating blameworthiness scores also depends on whether the utility function is context-relative or not. If it is, then we sum over all contexts (as well as all outcomes) when calculating the costs of actions, leading to a factor of $O(2^{|\mathcal{X}|})$ increase in complexity. Finally, note that we assume here that the PSDD and utility function are given in advance and so we do not consider the computational cost of learning. A summary of our results is as follows:

- When we are given $\mathbf{q}$ (whether or not $\mathbf{u}$ is context-relative has no effect in this case)

Term	Time Complexity
$\delta_{a,a',\varphi}$	$O(2^{\lvert\mathcal{X}\rvert+\lvert\mathcal{D}\rvert+\lvert\mathcal{O}\rvert}\lvert\varphi\rvert\lvert P\rvert Q)$
$c(a)$	$O(2^{\lvert\mathcal{X}\rvert+\lvert\mathcal{O}\rvert}U\lvert P\rvert Q)$
$db_N(a,a',\varphi)$	$O(2^{\lvert\mathcal{X}\rvert+\lvert\mathcal{O}\rvert}\lvert P\rvert Q(2^{\lvert\mathcal{D}\rvert}\lvert\varphi\rvert+U))$
$db_N(a,\varphi)$	$O(\lvert\mathcal{R}(A)\rvert 2^{\lvert\mathcal{X}\rvert+\lvert\mathcal{O}\rvert}\lvert P\rvert Q(2^{\lvert\mathcal{D}\rvert}\lvert\varphi\rvert+U))$

- When we are not given $\mathbf{q}$ but $\mathbf{u}$ is context-relative

Term	Time Complexity
$\delta_{a,a',\varphi}$	$O(2^{\lvert\mathcal{D}\rvert+\lvert\mathcal{O}\rvert}\lvert\varphi\rvert\lvert P\rvert)$
$c(a)$	$O(2^{\lvert\mathcal{X}\rvert+\lvert\mathcal{O}\rvert}U\lvert P\rvert)$
$db_N(a,a',\varphi)$	$O(2^{\lvert\mathcal{O}\rvert}\lvert P\rvert(2^{\lvert\mathcal{X}\rvert}U+2^{\lvert\mathcal{D}\rvert}\lvert\varphi\rvert))$
$db_N(a,\varphi)$	$O(\lvert\mathcal{R}(A)\rvert 2^{\lvert\mathcal{O}\rvert}\lvert P\rvert(2^{\lvert\mathcal{X}\rvert}U+2^{\lvert\mathcal{D}\rvert}\lvert\varphi\rvert))$

- When we are neither given $\mathbf{q}$ nor is $\mathbf{u}$ context-relative

Term	Time Complexity
$\delta_{a,a',\varphi}$	$O(2^{\lvert\mathcal{D}\rvert+\lvert O\rvert}\lvert\varphi\rvert\lvert P\rvert)$
$c(a)$	$O(2^{\lvert O\rvert}U\lvert P\rvert)$
$db_N(a,a',\varphi)$	$O(2^{\lvert O\rvert}\lvert P\rvert(U+2^{\lvert\mathcal{D}\rvert}\lvert\varphi\rvert))$
$db_N(a,\varphi)$	$O(\lvert\mathcal{R}(A)\rvert 2^{\lvert O\rvert}\lvert P\rvert(U+2^{\lvert\mathcal{D}\rvert}\lvert\varphi\rvert))$

We observe all of the final time complexities for calculating cost, the probability of an outcome given an action, and hence blameworthiness are exponential in at least some subset of the variables. This is in fact equivalent to the result stated in the original paper, where these complexities are described as being polynomial in the size of $\mathcal{K}$, which is $O(2^{|\mathcal{X}|+|\mathcal{D}|+|O|})$.

In practice however, $\mathcal{K}$ is the set of all actually possible worlds, those with non-zero probability of occurring. This is likely to be much smaller than the total set of worlds (where a world in this sense corresponds to an assignment of values to all the binary variables, giving us the exponential complexity described above). Using PSDDs allows us to exploit this fact in ways that other models cannot, as we can logically constrain the model to have zero probability on any impossible world. Thus, in summing over possible assignments to the variables when calculating blameworthiness we can ignore a great many of the terms and speed up computation dramatically.

Chapter 4

Implementation

Here we describe the more technical contribution of the project; an algorithmic pipeline that implements the theoretical work in the preceding chapter. This pipeline consists of existing and novel resources which are both discussed in detail before a high-level overview of the full structure is given.

4.1 Existing Resources

Aside from standard Python packages there are two existing resources that our work builds upon, each of which is used to create vtrees, SDDs, or PSDDs given different kinds of input. These are included under the `resources` directory of our final package, although are called from within the pipeline and so do not need be used manually. With that said, the features of the pipeline are better understood with some explanation of how the existing resources work and how we utilise them.

4.1.1 The SDD Package

Fundamental to the majority of the software and implementations using SDDs or PSDDs is the SDD package and library [38], an open-source system for creating and managing SDDs. The primary feature within this package that we employ is the SDD API, and in particular SDD managers, for compiling logical formulae into SDDs. In concrete terms an SDD manager is a C program with three broad sections:

- First the SDD is initialised based on a vtree which can be created at the same time or read from a file. The variables in the SDD are also specified here

- The bulk of the program is then used to construct the SDD using a series of logical operations (disjunction, conjunction, and negation) that are sequentially applied to larger and larger sub-SDDs over the set of variables. Once completed the full SDD can be manipulated and queried according to the purposes of the user (this is not required for our work)

- Finally, the manager can be used to save SDDs and vtrees to files, as well as corresponding Dot graphs for visualisation, and a record of any queries and results

A sample SDD Manager (corresponding to our example given in Section 2.3.2) is given in Section B.2.1 of the appendix for reference. The functionality we rely on here is the construction and outputting of SDDs, which can in turn be parameterised to form a PSDD for our model. However, as can be seen from the small example provided, the syntax and use of SDD Managers is somewhat complicated, and so as part of the novel resources detailed in Section 4.2 we include a parser that converts logical formulae written using a simple infix notation into a corresponding SDD Manager and thus to an SDD.

4.1.2 LearnPSDD

The most important existing resource in our implementation is LearnPSDD, a recently developed set of algorithms that can be used to learn the structure of PSDDs from data [19]. It also includes a variety of methods for learning vtrees and for converting SDDs to PSDDs. We now briefly describe these three functionalities and how we make use of them in our work.

Each split in a vtree corresponds to an assumption of independence between the variables in each of the two branches. Because of this, learning the structure of the vtree is critically important to the likelihood of the eventual PSDD. In order to quantify the level of independence between sets of variables the `learnVtree` algorithm chooses the split that minimises the average pairwise mutual information between the variables, given by:

$$APMI(\mathcal{X},\mathcal{Y}) = \frac{1}{|\mathcal{X}||\mathcal{Y}|}\sum_{X\in\mathcal{X}}\sum_{Y\in\mathcal{Y}}\mathbf{p}(X,Y)log\frac{\mathbf{p}(X,Y)}{\mathbf{p}(X)\mathbf{p}(Y)}$$

The construction by splitting can either be done bottom-up (joining sub-vtrees one at a time) or top-down (splitting up sub-vtrees one at a time), although the former leads to slightly better performance in practice and so this is what we use in our pipeline.

To convert an SDD to a PSDD we use `sdd2psdd` which calculates the maximum likelihood estimates of the parameters (the formulae for which are given in [2]) using training data, and possibly validation or test data. This algorithm also requires the vtree associated with the SDD, which is typically another output of the SDD manager. Finally, we may specify the smoothing method, if any, to use. Of the available options we choose Laplace smoothing with parameter 1, recommended by the authors of the software as a fairly reliable and robust setting [39].

The third and final algorithm we use is `learnPsdd`, which takes as input a vtree and training data (with the option of also including validation or test data) and outputs a PSDD, both the structure and the parameters of which are learnt from data. The learning process involves iteratively performing two operations, split and clone, on an initialised PSDD until a time or size limit is reached, or until the log likelihood of the model converges. Split works by breaking up elements (a conjunction of a prime and a sub) and forming two new, mutually exclusive primes based on possible assignments to the original, then joining them both to the same sub as before. Clone copies a node and joins it to a subset of its parents.

The key point with both of these operations is that they change the structure of the induced distribution but not the logical base encoded by the underlying SDD. Which operation is chosen at each step in the algorithm is determined by a score function that measures the increase in log likelihood over the increase in the size of the model. As in `sdd2psdd` we can also specify a smoothing parameter, and in our work we again choose Laplace smoothing with parameter 1. Finally, `learnPsdd` can take an existing PSDD as input and be used to refit the structure and parameters if required, which is a feature we offer to the user of our pipeline after converting an SDD to a PSDD[1].

4.2 Novel Resources

As well as the existing resources detailed above, our implementation is also required to be able to perform many tasks for which there was no existing software. These tasks can be broken up into four broad groups, each of which relates to a separate file in our packaged implementation and contains a selection of functions. Here we provide a brief overview of our requirements and implemented solutions. The full details of the

[1] A small idiosyncrasy with the LearnPSDD package means that the original likelihood scores reported when using `sdd2psdd` are wildly inaccurate, and so the PSDD must be refitted (even if the result is the same as the original) to obtain correct values.

package and definitions of the functions can be found in Section B.1 of the appendix.

4.2.1 Building Models

In order to maximise the cohesiveness and convenience of the pipeline we needed a way to run the algorithms that construct the building blocks of our models (SDD managers, vtrees, SDDs, and PSDDs) from within the final program. When creating a new model in the pipeline, the user specifies a name which in turn leads to the creation of new sub-directories of this same name in the `data`, `models`, and `output` directories of the package. Any other parts of the model that are constructed or inputted by the user are then stored in the relevant sub-directory and can be accessed and re-used later.

The first group of functions simply set up the LearnPSDD package (stored in the `resources` directory) and run `learnVtree`, `sdd2psdd`, or `learnPsdd` via instructions to the command line. They take as input the name of the model, any datasets that the user has entered, and any required constituents such as vtrees or SDDs that have either been learnt already or inputted by the user.

While the input taken by the learning algorithms above is easily specified by the user, the construction of SDDs from logical constraints using the SDD package is far less straightforward. To solve this problem we created a function that parses a list of logical constraints and creates an SDD manager from them, which can in turn be compiled and run in order to output the SDD and vtree. This list of constraints can be written using a simple prefix notation for logical connectives and either letters or numbers denoting variables, depending on the preference of the user. For example, $(A \wedge B) \leftrightarrow C$ can be written as `=(&(A,B),C)` or `=(&(1,2),3)`. The pipeline provides instructions and an interface for entering these constraints via the command line during its execution, and then stores the entries in a file for parsing or editing later.

4.2.2 Performing Inference

The software resources currently available for PSDDs do not extend to performing manual inference or executing user-inputted queries, but these tasks are key to our ability to calculate blameworthiness scores. We therefore developed a series of functions to this end that make use of the tractability properties of PSDDs. This included several helper functions including a parser that reads a PSDD in from the file format outputted by LearnPSDD and creates a corresponding data structure, as well as functions for printing out summaries of the PSDD and its nodes which we use in our

pipeline as feedback to the user after the model has been completed.

There are two main inference functions. The first evaluates the PSDD bottom-up with partial (possibly complete) evidence and returns the evidence's probability. When non-complete evidence is provided (meaning not every variable is assigned a value) the algorithm automatically marginalises out the unspecified variables by assigning value 1 to any nodes associated with those variables. This evaluation of the PSDD follows the same pattern as our calculations at the end of Section 2.3.2, and the pseudo-code on which we based our implementation can be found in [2].

The second inference function is a Viterbi-style algorithm that calculates the MPE of the distribution represented by the PSDD given partial evidence (possibly none). Again, the PSDD is evaluated bottom-up, but this time with max nodes replacing the decision nodes, which select the maximising element. Each of the children with maximal value is stored and after the bottom-up evaluation is complete the variables are set to their correct values in a single top-down pass through the PSDD, following the maximising branch at each decision node. Finally, the MPE algorithm outputs both the most likely assignment of values to the variables not in the original evidence, along with the probability of this assignment. Pseudo-code for MPE inference using a very similar model (selective SPNs) is provided in [40] and was an influence on our implementation.

For our later calculations we also require the variables in the PSDD to be partitioned into random, decision, and output sets. This choice is made via the command line during the execution of the pipeline, but to avoid re-specifying them each time a user works with the model, we provide functions for saving the choice to a text file which can then be read in and applied to the PSDD data structure any time the model is re-used.

We conclude this subsection by noting that the inference algorithms and the splitting up of variables into the three classes allow us to perform a variety of useful queries aside from calculating blameworthiness scores. In particular the PSDD can be used to make decisions by providing evidence as a setting of the random variables and then taking the MPE assignment of the decision variables. This can also be done when some random variables are not specified, and so are marginalised out as unknowns, and when some outcome variables are specified; say if we wanted to find the most likely setting of the decision variables that would lead to a particular outcome. These are not required for calculating blameworthiness scores, but could potentially be useful in related future work.

4.2.3 Utilities from Probabilities

Once the PSDD data structure has been created and the variables defined the utility function may be learnt. Alongside the learning function we include helper functions that normalise a utility function to the range $[0,1]$, and a function that calculates the expected utility within a context, possibly given a partial assignment to the decision variables. As explained in Section 3.2.2, the form that the learnt utility function takes depends on three things: whether there are outcome variables in the model, whether the function is linear in the outcome variables, and whether the function is context-relative. The implemented function for learning utility values therefore proceeds on a case-by-case basis.

When there are no outcome variables, learning is straightforward as we have either $\mathbf{p}(\mathbf{D}|\mathbf{X}) \propto \mathbf{u}(\mathbf{D}|\mathbf{X})$ or $\mathbf{p}(\mathbf{D}) \propto \mathbf{u}(\mathbf{D})$ depending on whether the function is context-relative or not[2]. To learn the utility values we simply loop over the possible arguments of the utility function and calculate the relevant probabilities. Finally we normalise the function, either by dividing by the highest utility value in each context for the context-relative case, or by the highest overall utility value otherwise.

The process is slightly more complicated and more computationally expensive when there are outcome variables involved. Here we have four cases, as the function may or may not be linear in the outcome variables as well as being context-relative or not. We illustrate the calculations made by our algorithm on just one of these cases, but the other three follow precisely the same approach. In particular we consider a utility function that is linear, but not context-relative, as we believe this captures the majority of standard cases. Recall that, according to our proportionality assumption, for each context $\mathbf{X}$ and each set of decisions $\mathbf{D}$ we have:

$$\mathbf{p}(\mathbf{D}|\mathbf{X}) \propto \sum_i \mathbf{u}_i(O_i)\mathbf{p}(O_i|\mathbf{D},\mathbf{X})$$

where each outcome variable O_i has a weight $\mathbf{u}_i(O_i) \geq 0$.

In a scenario with l possible contexts $\mathbf{X}$, m possible decisions $\mathbf{D}$, and n outcome variables O_i we represent the above equation[3] for all $\mathbf{X}$ and $\mathbf{D}$ in matrix form as follows:

[2] In the latter case we drop the conditional context $\mathbf{X}$ in the probability term as we calculate $\mathbf{u}(\mathbf{D})$ by taking an average over contexts weighted by $\mathbf{p}(\mathbf{X})$ and so we have $\sum_{\mathbf{X}} \mathbf{p}(\mathbf{D}|\mathbf{X})\mathbf{p}(\mathbf{X}) = \mathbf{p}(\mathbf{D}) \propto \mathbf{u}(\mathbf{D})$.

[3] Note that we assume equality here, as opposed to linear proportionality, and normalise the utility values after learning.

$$\begin{bmatrix} \mathbf{p}(O_1|\mathbf{D}_1,\mathbf{X}_1) & \dots & \mathbf{p}(O_n|\mathbf{D}_1,\mathbf{X}_1) \\ \vdots & \ddots & \vdots \\ \mathbf{p}(O_1|\mathbf{D}_m,\mathbf{X}_1) & \dots & \mathbf{p}(O_n|\mathbf{D}_m,\mathbf{X}_1) \\ \vdots & \ddots & \vdots \\ \mathbf{p}(O_1|\mathbf{D}_1,\mathbf{X}_l) & \dots & \mathbf{p}(O_n|\mathbf{D}_1,\mathbf{X}_l) \\ \vdots & \ddots & \vdots \\ \mathbf{p}(O_1|\mathbf{D}_m,\mathbf{X}_l) & \dots & \mathbf{p}(O_n|\mathbf{D}_m,\mathbf{X}_l) \end{bmatrix} \begin{bmatrix} \mathbf{u}_1(O_1) \\ \vdots \\ \mathbf{u}_n(O_n) \end{bmatrix} = \begin{bmatrix} \mathbf{p}(\mathbf{D}_1|\mathbf{X}_1) \\ \vdots \\ \mathbf{p}(\mathbf{D}_m|\mathbf{X}_1) \\ \vdots \\ \mathbf{p}(\mathbf{D}_1|\mathbf{X}_l) \\ \vdots \\ \mathbf{p}(\mathbf{D}_m|\mathbf{X}_l) \end{bmatrix}$$

Given our restriction that each weight $\mathbf{u}_i(O_i)$ is non-negative the obvious choice here is to use a linear program to find the column vector of weights. However in practice this leads to extreme solutions where many weights are zero and others are relatively large. To counteract this we instead chose to use non-negative linear regression with L2 regularisation, which is equivalent to solving a quadratic program as opposed to a linear one. This is a relatively simple method of learning but one well-suited to our requirements. Writing the equation above as $\mathbf{Ax} = \mathbf{b}$, we compute the following vector:

$$argmin_{\mathbf{x}}\left(||\mathbf{Ax}-\mathbf{b}||_2^2+\lambda||\mathbf{x}||_2^2\right)$$

where λ is the regularisation constant and $||\cdot||_2^2$ is the square of the Euclidean norm. In a brief series of experiments we found that our results were relatively insensitive to our choice of λ, so long as it was non-negative, with the best results in general arising from setting $\lambda = \frac{n}{2}$. This is therefore the standard value we chose to use in our implementation and across our experiments.

Before continuing we note briefly that depending on the decision-making scenario there may be some contexts in which an outcome variable is independent of any possible decision made by the agent. In terms of our matrix above we would have a sub-column containing only a constant value: $\mathbf{p}(O_i|\mathbf{D}_j,\mathbf{X}_k)$ would be the same for all j. Intuitively we do not want our learning process to infer anything about the utility of outcome O_i in context $\mathbf{X}_k$ based on this data. Hence in our implementation we ignore such situations when calculating the utility values (by setting constant sub-columns to zero vectors) and maintain a count for the probability of each outcome variable to measure in how many contexts it can take different values. Once the utility values have been learnt they are then normalised by dividing by their respective counts to give the actual values.

After the utility function has been computed the non-zero values are stored in a dictionary along with the weights, in the case of a linear function. As can be seen from

the size of the full matrix, when the number of variables becomes large this learning process can become quite computationally expensive, and so we also include a function that writes the stored utility function to a text file that can be read in quickly and added to the PSDD data structure whenever the model is re-used. This ability to read in utility functions also allows the user to specify their own function during the model creation stage of the pipeline if they so desire.

A second method we used to maximise the tractability of our implementation was to exploit the assumption that the only assignments of values to variables with zero probability are those that are logically impossible as defined by any constraints on the model. This holds true in our system due to the smoothing we use in the learning process, and so in many problems we are able to exponentially decrease the number of calculations made, as we can safely ignore zero rows in the above probability matrix and vector. This of course has the side-effect of meaning that utilities for scenarios with zero probability are undefined, but this seems eminently reasonable to us and has no impact on further calculations.

4.2.4 Computing Blameworthiness

The functions created to compute blameworthiness are among the simplest, with each corresponding to the quantities needed in the final equation: $\sum_{\mathbf{Y}} \varphi(\mathbf{Y})\mathbf{p}(\mathbf{Y}|a)$, $\delta_{a,a',\varphi}$, $c(a)$, $db_N(a,a',\varphi)$, and finally $db_N(a,\varphi)$. These are all implemented to reflect the formulae defined in our mapping from Section 3.1. The only noticeable departure from the original theoretical versions is that the functions that calculate quantities $\delta_{a,a',\varphi}$ and $db_N(a,a',\varphi)$ take an additional variable as input to accommodate for the fact that we one-hot encode any decision variables that can take more than two values, as sometimes we may want to compare between, say $A = a$ and $A = a'$ as opposed to simply $A = a$ and $A \neq a$. In concrete terms we have `db_n(A1, a1, A2, a2, phi, psdd, N, q)` instead of `db_n(A, a1, a2, phi, psdd, N, q)`, where the uppercase `A` arguments represent (binary) variables and the lowercase `a` arguments are their values.

Also included within this group of functions is a parser that reads in a probability distribution over the contexts of the model and stores it as a data structure for use in computing blameworthiness. This is required when the user chooses to specify a particular distribution $\mathbf{q}$ in order to calculate blame in a scenario that is not representative of the learnt distribution $\mathbf{p}$ over contexts. Using a distribution over a smaller number of contexts can also be used as a method to increase the speed of our computations as

once again we may safely ignore any scenarios with zero probability.

4.3 Algorithmic Pipeline

The underlying motivation behind our pipeline was that a user should be able to go from any stage of creating a model (from simply possessing the data, to perhaps having already encoded some constraints into an SDD, or having derived a utility function) to generating blameworthiness scores as conveniently and as straightforwardly as possible. With this in mind our final file in the packaged implementation runs from the command line and prompts the user for a series of inputs including: data; existing PSDDs, SDDs, or vtrees; logical constraints on the data; utility function specifications; variable descriptions; and finally the decisions, outcomes, and other details needed to compute a particular blameworthiness score. These inputs and any outputs from the pipeline are stored in newly created sub-directories of the `data`, `models`, and `output` directories inside the package. Thus each model and its results can be easily accessed and re-used if needed. The full structure of the pipeline can be seen in Figure 4.1 and the package is available to download at [41].

In this report we provide a high-level overview of the pipeline from start to end and justify the structural decisions made. To illustrate the use of the pipeline we also include the command line input and output as a sample of typical usage in Section B.2.2 of the appendix. We encourage the reader to download and experiment with the pipeline for themselves using the example data and information we offer within the package distribution, if they so desire.

The pipeline can be broken down into two main sections, the first concerning the construction of the model and the second used for generating blameworthiness scores. The model creation stage can similarly be broken down into two large subsections. The first of these subsections is captured fully in the left half of Figure 4.1 and corresponds to building a PSDD. The order of the prompts to the user in this subsection (in other words the order of the decisions the user makes, denoted by diamond nodes in the control flow diagram) can be thought of as working backwards through the algorithmic process of creating a PSDD.

Firstly, if the user already has a model then they may safely skip both initial subsections and proceed directly to generating blameworthiness scores. Likewise, if they already have a PSDD then they may skip ahead to the second subsection involved in creating the model, after inputting the path to the PSDD file. Note here that user inputs

are denoted in the control flow diagram by parallelogram nodes and processes by rectangles. If the user does not have a PSDD to input, then the next closest stage is to have an existing SDD, and so this is the next check made in the pipeline. If the answer is affirmative the program requests a path to the SDD file and to the data (along with the vtree associated with the SDD) and generates a PSDD using `sdd2psdd`. If not, then we must first either learn or accept as input a vtree that is required for either `sdd2psdd` or `learnPsdd`.

Following this the user has the option to restrict the PSDD to be learnt by inputting a series of logical constraints. If they choose to do so, then the program records and stores these constraints before using them and the vtree to create an SDD manager. An SDD is then constructed by the manager and converted to a PSDD by `sdd2psdd`. If no constraints are specified then the structure of the PSDD is also learnt from data and the vtree using `learnPsdd`. After any of these routes to generating a PSDD has been completed the user has the final option of whether or not to refit their model. This is only necessary if the original PSDD was learnt without smoothing (as explained in Section 4.2.3), but can also be used to increase the likelihood or decrease the size of the model, especially if it was originally parameterised using `sdd2psdd` (see also Footnote 1 in Section 4.1.2).

The second subsection involved in creating a model concerns the utility function and can be seen in the right half of Figure 4.1, consisting of all nodes above *q?*. In order to set up a utility function for the model we first need to know how the variables are partitioned (into the classes random, decision, and outcome) and so we request this from the user. The user may then specify a particular utility function they wish to use (in the form of a text file in which any assignment with non-zero utility forms a row, with the final element being the utility value) or choose to learn one using the probability distribution learnt from data that has now been encoded in the PSDD.

When learning a utility function the user decides whether the function should be linear in the outcome variables (assuming they exist) and whether it should be context-relative, giving us the six cases shown previously in Table 3.1. In general we expect these choices to depend on the particular scenario the user wishes to model. Following this the utility function is learnt as described in Section 4.2.3, saved to an external file in the relevant sub-directory, and thus the model is complete.

The second main section of the pipeline can be repeated as many times as the user wishes within the same execution of the program and generates blameworthiness scores, saving them as sentences in natural language to a file in the `output` directory.

This section of the pipeline can again be seen in the right half of Figure 4.1 and consists of all remaining nodes. We begin by asking the user whether they would like to specify a particular distribution **q** over contexts in which to compute the blameworthiness score. This potential input also takes the form of a text file in which each context with non-zero probability is recorded as a separate line of comma-separated variable values with the final element being the probability of the context.

The idea behind this feature is that the user can learn a general model and then use the same utility function and outcome probabilities (given a context and decision) in a variety of specific instances. The distribution **q** can also be thought of as an update to the decision-making agent's epistemic state (or perhaps the epistemic state they ought to have) with regard to the worlds they believe possible and their relative probabilities. In the general case we simply use the distribution already learnt and encoded in the PSDD; this further option merely grants the user some extra flexibility and the chance to avoid having to learn multiple similar models.

The next step is for the user to input the decision that was made and the decision(s) they wish to compare it to. In the case of comparing two specific actions these are entered identically as the names of the decision variables and the corresponding values. In the case of computing the general blameworthiness $db_N(a,\varphi)$ of an action a for outcome φ we ask the user to specify all the other one-hot encoded variables that represent the decision variable A in question so that we take the maximum from the correct set of values.

Recall that an outcome φ is a Boolean formula of primitive events, which are in turn assignments of a value to a variable. We ask the user for this function in terms of the basic Python syntax for Boolean connectives, which we then convert directly into a lambda expression for use in our calculations. For example, if φ is given by $(B = 1) \wedge (E = 0)$ the user should input `(o[2] == 1) & (o[5] == 0)`. The variable naming convention seen here is also explained to the user during the program with an example.

The final input that may be entered is a value for N which measures cost importance when determining blameworthiness. We offer two standard options to the user: $N = 1$ (the maximum possible utility for any assignment) and $N = 1.1 \times (-min_{a \in A} c(a))$, as we require that $N > -min_{a \in A} c(a)$. If the user wishes to enter their own value for N we also provide them with the lower bound $-min_{a \in A} c(a)$ to help inform their choice. At this stage the blameworthiness score is generated and saved to an output file. The user may then generate other blameworthiness scores by returning to the start of the second

section or decline this last prompt, in which case the program terminates.

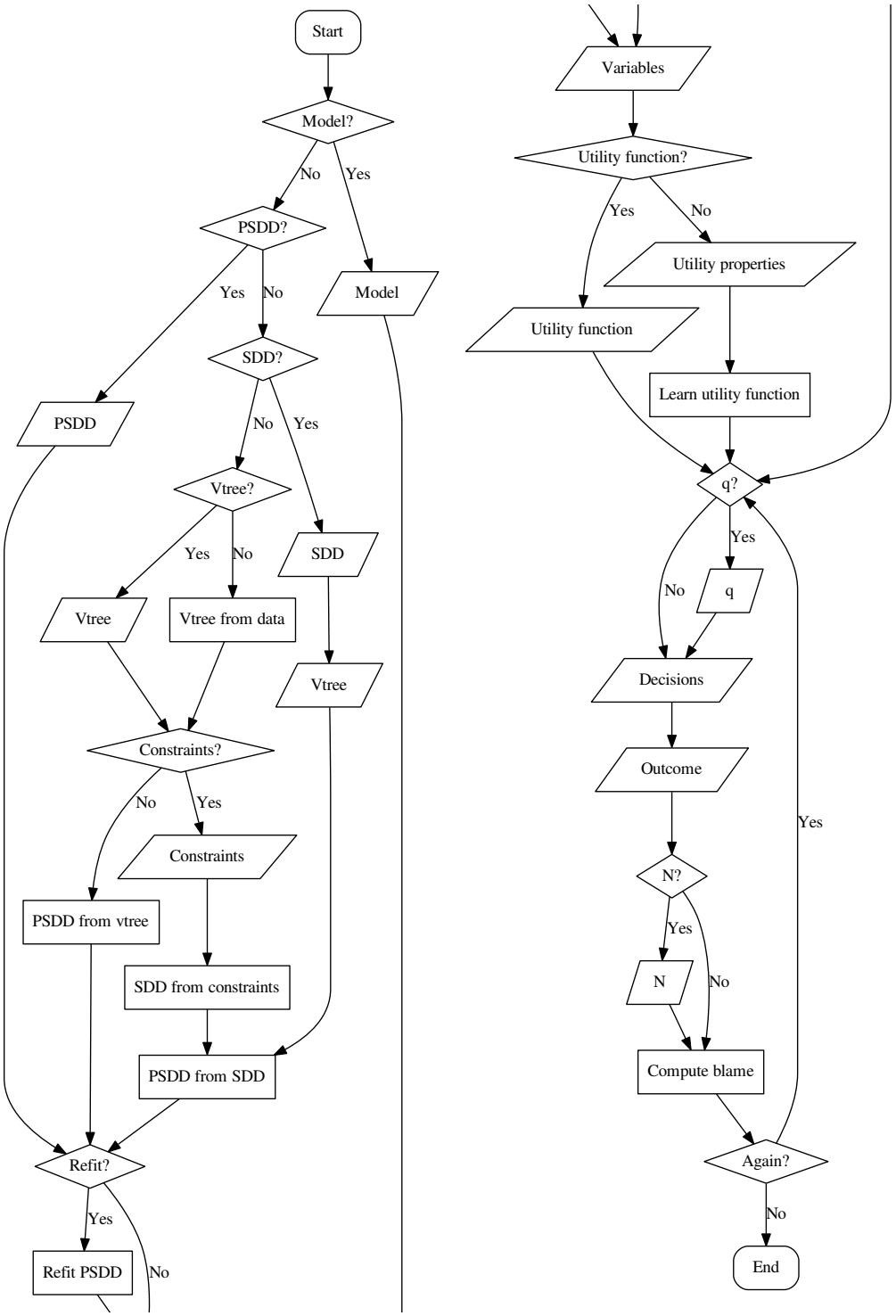

Figure 4.1: The control flow of our pipeline, split into two halves. Rounded rectangles are start and end points, diamonds represent decisions, parallelograms correspond to inputs from the user, and rectangles are processes undertaken by the program.

Chapter 5

Experiments and Results

With our implementation complete we proceed to learn several models using a selection of datasets from varying domains in order to test our hypotheses. In particular we wish to answer three questions in each case:

- Does our system learn the correct overall probability distribution?
- Does our system capture the correct utility function?
- Does our system produce reasonable blameworthiness scores?

Each of our three experiments follows approximately the same structure, although due to the different natures of the datasets we use slightly different methods to attempt to answer these questions in the different cases, as is explained in our analysis.

5.1 Lung Cancer Staging

In our first experiment we use a synthetic dataset, albeit one generated using a model constructed from expert knowledge. More specifically we use the lung cancer staging ID given in [42] (and shown for reference in Figure 5.1, without the distribution tables for the random variables), an early and influential example of using IDs to analyse real world decision-making. IDs are similar to our work in that they are compact probabilistic models of decision-making scenarios that use random variables (which can also function as outcome variables), decision variables, and a utility function. With that said there are some key differences including, but not limited to: the fact there is currently no way to learn IDs from data; inference in IDs is generally intractable [13]; the utility function is encoded by nodes in the ID (and thus always presumed

known); and that typically IDs are used as tools to make decisions rather than to model distributions over decisions.

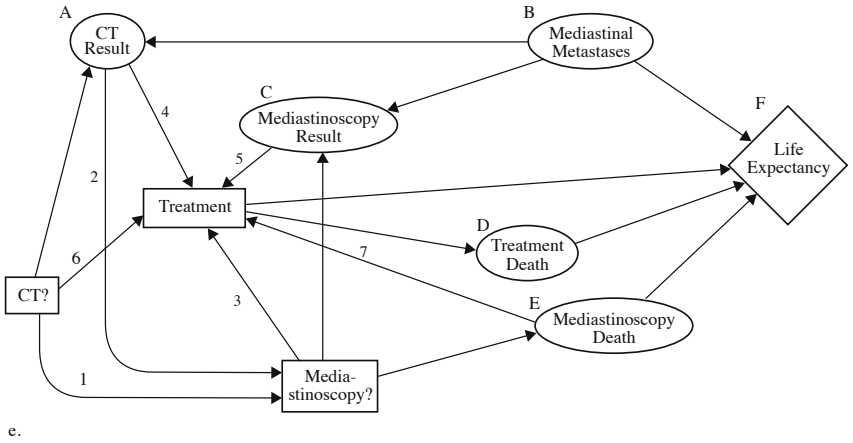

Figure 5.1: The ID used to generate the synthetic data in our first experiment (this image was copied from [42]). Elliptical nodes correspond to random variables, rectangular nodes to decisions, and the diamond node to utility.

Because of this last feature, decisions are not made probabilistically but by the user of the ID, or by solving the ID in order to maximise utility, and so we have no distribution of decisions to learn from. We therefore generate the data from the ID assuming that the overall decision strategy recommended in the original paper is followed with some high probability at each point. This initial experiment thus functions as something of a test to see how well our model can recover a decision-making strategy from data. In the lung cancer staging ID decisions are made sequentially, but our models do not naturally capture this and so we introduce extra random variables to represent how previous decisions have been made (for instance, if a CT scan was not conducted then we have $CT_{N/A} = 1$ in the context). Using this extension we can still capture the entire decision-making process using one model.

5.1.1 Data

We now outline the data generation process. This follows the lung cancer staging procedure in which a thoractomy (T) is the usual treatment unless the patient has mediastinal metastases (MM), in which case a thoractomy will not result in greater life expectancy than the lower risk option of radiation therapy, which is the preferred treatment in this case. The first decision to be made is whether a CT scan should be

performed to test for mediastinal metastases (the prior probability of which is 0.46). This is typically standard procedure and so we have $CT = 1$ with probability 0.9 (a description of the variables and a summary of the dataset is given in Table 5.1).

The following decision is whether to perform a mediastinoscopy (M). If the CT scan results are positive for mediastinal metastases then a mediastinoscopy is usually recommended in order to provide a second check, but if the CT scan result is negative then a mediastinoscopy is not seen as worth the extra risk involved in the operation, and so we proceed directly to performing a thoractomy as treatment. Hence in the case of a negative CT scan we have $M = 1$ with probability 0.2 and in the case of a positive CT scan we have $M = 1$ with probability 0.7. When a CT scan is not performed we have $M = 1$ with probability 0.9.

Finally, we model the decision of whether to perform a thoractomy as an outcome based on the results of the tests. Here a thoractomy is performed if the last diagnostic test is negative (when both a CT scan and a mediastinoscopy are performed, the CT scan always precedes the mediastinoscopy), or in the case where no diagnostic tests were made a thoractomy is performed with probability 0.6. We remind the reader that these are not actual statistics from medical practice but an artificial distribution over decisions that approximately follows the strategy recommended in [42].

We encode the specificity and sensitivity of both the CT scan and mediastinoscopy using random variables, and whether or not the patient survives the mediastinoscopy or either of the treatments using non-deterministic outcome variables (S_{DP} and S_T). We also provide a list of logical constraints on the distribution to ensure that it follows the process described above. For example, we require that the one-hot encoded variables (those that record the result, if any, of each of the two diagnostic tests) are mutually exclusive, that thoractomies are performed in particular cases only, that the patient can only die from a mediastinoscopy if one was performed, and so on.

Utilities in this problem are measured by life expectancy in years and depend on the treatment given and whether the patient has mediastinal metastases. It is worth remarking briefly that in the original paper the differences in overall life expectancies are extremely similar for most strategies (within a few days of each other), so similar that in a related paper the authors chose to instead measure utility by the cost of the operations involved [43]. The existence of potential disagreement in strategies over such a small difference in utility is evidence that the decision-making agent (the medical practitioner) is what we referred to in Section 3.2.2 as *sophisticated*. This is something we might reasonably expect and hope to see in such a domain.

Number of data points	100,000
Number of variables	12
Random variables	Mediastinal Metastases (MM), CT Positive (CT_+), CT Negative (CT_-), No CT ($CT_{N/A}$), Mediastinoscopy Positive (M_+), Mediastinoscopy Negative (M_-), No Mediastinoscopy ($M_{N/A}$)
Decision variables	Perform CT (CT), Perform Mediastinoscopy (M)
Outcome variables	Perform Thoractomy (T), Diagnosis Procedures Survived (S_{DP}), Treatment Survived (S_T)
Constraints	$(CT_+ \vee CT_-) \leftrightarrow CT$, $CT_{N/A} \leftrightarrow \neg CT$, $(M_+ \vee M_-) \leftrightarrow M$, $M_{N/A} \leftrightarrow \neg M$, $M_- \rightarrow T$, $M_+ \rightarrow \neg T$, $(CT_- \wedge \neg M) \rightarrow T$, $(CT_+ \wedge \neg M) \rightarrow \neg T$, $\neg S_{DP} \rightarrow M$, $\neg(CT_+ \wedge CT_-)$, $\neg(M_+ \wedge M_-)$, $\neg S_{DP} \rightarrow \neg S_T$
Model count	52
Utilities recorded?	Yes (Life Expectancy)

Table 5.1: A summary of the lung cancer staging data used in our first experiment.

5.1.2 Results

To answer our first question we measure the overall log likelihood of the models learnt by our system on training, validation, and test datasets (formed by shuffling the data and partitioning them into sets sized according to the ratio 70:15:15). A full comparison of log likelihood scores across a range of similar models and learning techniques is beyond the scope of this project, although to provide some evidence of the competitiveness of PSDDs we include the log likelihood scores of an SPN learnt from the same data as a benchmark.

We follow a similar pattern in our remaining experiments, each time using Tachyon (an open source library for SPNs [44]) to produce an SPN using the same training, validation, and test sets of our data, with the standard learning parameters as given in the documentation example[1]. We also compare the sizes (measured by the number of nodes) and the log likelihoods of PSDDs learnt with and without logical constraints in order to demonstrate the effectiveness of the former approach (and the superiority of both PSDDs over the SPN). The log likelihoods across each of the three partitions

[1] In particular we use ten epochs and a mini-batch size of 100 with bounds [8,12] and [2,4] on the number of children of the sum and product nodes respectively. Further details of these parameters can be found in the original documentation.

of the dataset for each of these models is presented in Table 5.2. We can immediately observe that including the constraints aids LearnPSDD in creating a model that is both smaller and more accurate, especially when compared to the SPN.

Model	Training	Validation	Test	Size
PSDD With Constraints	-2.047	-2.046	-2.063	134
PSDD Without Constraints	-2.550	-2.549	-2.564	436
SPN (Using Tachyon)	-3.139	-3.143	-3.158	1430

Table 5.2: Log likelihoods and sizes of the two PSDDs and the SPN learnt from the lung cancer staging data.

We continue our analysis (using our model based on the PSDD with constraints from now on) in regard to our first question by assessing whether the model is able to recover the decision-making strategy used to generate the data. In order to do this we compute the probabilities with which certain decisions are made in the model across a variety of contexts and compare these to our original probabilities. This comparison can be seen in Figure 5.2 and is, in general, encouraging. At most points of the staging procedure the model learns a very similar distribution over decisions to our original strategy, and in all cases the correct decision is made the majority of times.

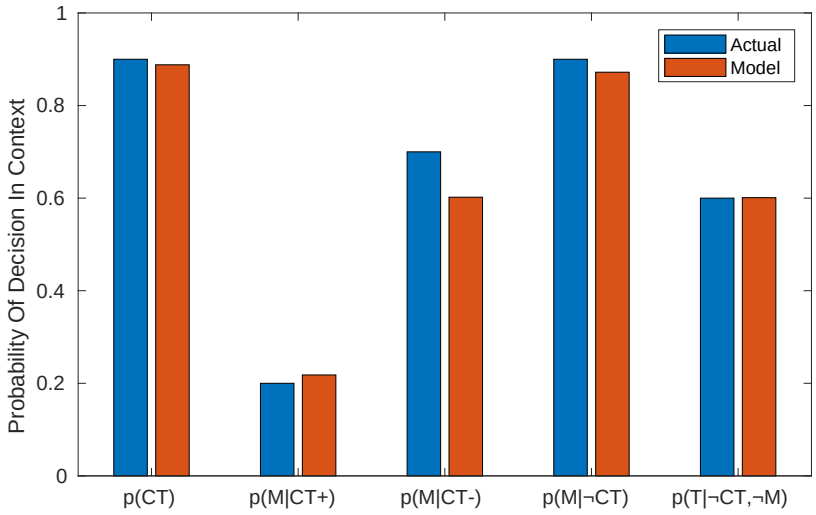

Figure 5.2: A comparison between the five probability values specified in our data generation process and the corresponding values learnt by our system from this data.

Answering our second question is slightly more difficult as the given utilities (which are measured by life expectancy and are context-relative and non-linear) are not neces-

sarily such that our decisions are linearly proportional to the expected utility resulting from them. This is a result of the somewhat arbitrary way we chose the probabilities involved in following the optimal (at least according to the analysis in [42]) strategy. With that said, our strategy was chosen so as to maximise expected utility in the majority of cases. Thus when comparing the given life expectancies with the utility function we learn for comparison, we would still hope to see the same ordinality of utility values, even if not the same cardinality.

The mean and standard deviations of the utility values learnt by the model for each of the possible overall outcomes are presented in comparison with the normalised life expectancies in Table 5.3. We observe that the utilities follow the same ordinal relationship as the original life expectancies apart from in the case where the patient has mediastinal metastases (and so either treatment gives the same life expectancy) and the case where the patient does not have mediastinal metastases and is given radiation therapy.

However we also note that the standard deviation in the case of this latter value is relatively large. This is likely to be a result of the fact that the probability of this scenario occurring is relatively small (approximately 0.06) when following our diagnosis strategy, and so the estimate may be improved by more data. That the mean learnt utility value in the scenario in which a patient dies is dramatically lower than any of the scenarios in which they survive is a positive result, as is that the successful performing of a thoractomy when the patient does not have mediastinal metastases (the optimal scenario) has maximal utility.

Context/Outcome	Normalised L.E.	Mean Utility	Standard Deviation
Die	0	0.134	0.323
Mediastinal Metastases	0.404	0.977	0.073
Radiation Therapy	0.593	0.927	0.163
Thoractomy	1.000	1.000	0.000

Table 5.3: A comparison between the life expectancies given in [42] and the utility values learnt by our system based on the decision strategy outlined earlier.

In attempting to answer our final question of whether the system makes reasonable attributions of blame we return to using the utility values given in the original data. We then break our question into two parts: does the system attribute no blame in the correct cases, and does the system attribute more blame in the cases we would expect it

to (and less in others)? Needless to say it is very difficult (perhaps even impossible, at least without an extensive survey of human opinions) to produce an appropriate metric for how correct our attributions of blame are, but we suggest that these two criteria are the most fundamental and capture the core of what we want to evaluate. It is also worth remembering that tuning some parameters (such as the cost importance measure N, or the regularisation constant λ when learning utility functions, for example) can help to produce more intuitive blameworthiness scores for different models.

Regarding the first part of the question, the system should produce a blameworthiness score of zero in cases where performing the action being judged is *less* likely to result in the outcome we are concerned with than the action(s) we are comparing it to. In our model the chance of the patient dying in the diagnostic process ($\neg S_{DP}$) is increased if a mediastinoscopy (M) is performed. Hence the blameworthiness for such a death due to *not* performing a mediastinoscopy should be zero.Using the notation from Section 2.2.1, we expect to find that $db_N(\neg M, M, \neg S_{DP}) = 0$, and this, in fact, is exactly what we do find (these outputs are left as an example in our distribution package and can be found in the relevant sub-directory). We also made similar checks for other situations in which we would expect there to be no blame and found all the scores to be zero[2].

Moving on to the second part of the question, we should see the system producing higher blameworthiness scores when a negative outcome is more likely to occur (assuming the actions in question have relatively similar costs). For example, in the case where the patient does not have mediastinal metastases then the best treatment is a thoractomy, but a thoractomy will not be performed if the last diagnosis test performed is positive. The specificity of a mediastinoscopy is 0.995 and higher than that of a CT scan, which is 0.81. Hence a CT scan is more likely to produce a false positive and thus (assuming no mediastinoscopy is performed as a second check) lead to the wrong treatment.

In the case where only one diagnostic procedure is performed we should thus expect to see a higher degree of blame attributed to the decision to conduct a CT scan as opposed to a mediastinoscopy. Note that even though a mediastinoscopy has a higher cost involved (in that the patient is more likely to die if it is performed), it should not be enough to outweigh the test's accuracy in this circumstance. This is exactly the result we obtain; with $N = 1$ (the maximum possible utility and one of our two standard

[2]Details of these other examples are omitted here for reasons of space and are excluded from the output file in the distribution as they were performed in earlier stages of our experiment as tests.

values offered to the user of our pipeline, as explained in Section 4.3) we have blameworthiness score 0.013098 for the CT scan and 0.000222 for the mediastinoscopy. Note that both of these numbers are low due to both the tests still being fairly accurate. As before, we conducted a selection of other successful tests but omit their descriptions and results here for reasons of space.

5.2 Teamwork Management

The second experiment we ran was on a recently collected dataset of human decision-making in teamwork management [45]. This data was recorded from over 1000 participants as they played a game (named Agile Manager [46]) that simulates task allocation processes in a variety of management environments. In each level of the game the player has different tasks to allocate to a selection of virtual workers that have different attributes and capabilities. The tasks vary in difficulty, value, and time requirements, and the player gains feedback from the virtual workers as tasks are completed.

At the end of the level the player receives a score where points are deducted from the maximum of 100% depending on how late or low-quality the work that they managed was. Finally, the player is asked to record their emotional response to the result of the game in terms of scores corresponding to six basic emotions. This last feature makes the dataset somewhat unique in its combination of scope and scale, although in our experiment we simplify the complexity of the emotional response scores by viewing utility as measured only by the happiness score reported by the user. We also remove all data entries where no input was given to any of the emotional responses, as we assume here that the user chose not to fill them in for reasons other than that they were completely emotionless. Screenshots from the game are provided in Figure 5.3 for illustrative purposes.

5.2.1 Data

All of the details above make for a fairly complicated decision-making environment, but we simplify matters somewhat by only considering the self-declared management strategy of the player as our decisions. Within the game this is recorded by five checkboxes at the end of the level that are not mutually exclusive, giving 32 possible overall strategies. These strategy choices concern methods of task allocation and are as follows: load-balancing (keeping each worker's workload roughly even), uniform (as-

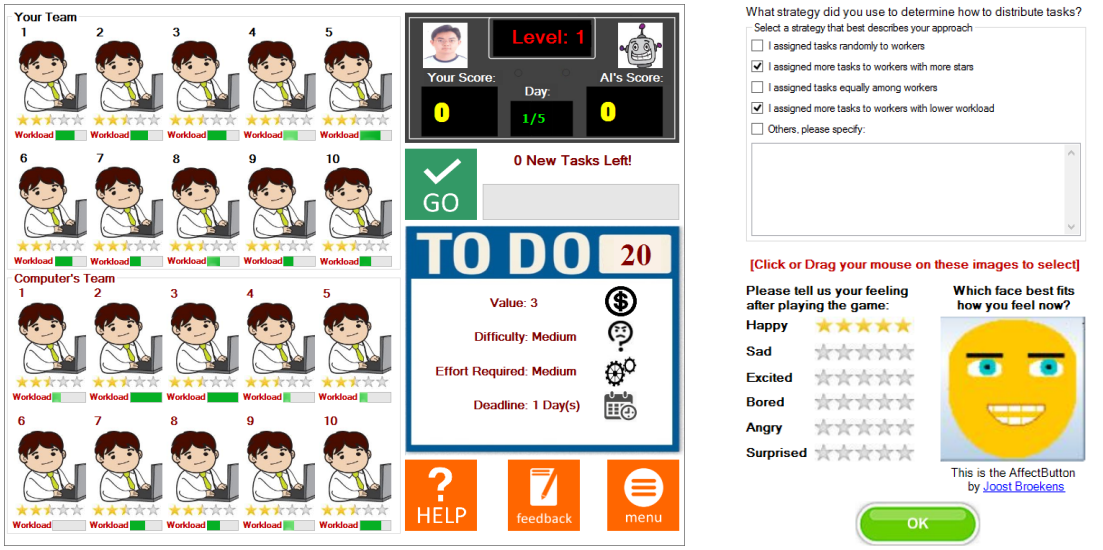

Figure 5.3: Two screenshots from Agile Manager taken just before a game starts (on the left) and just after it ends (on the right). These images were taken directly from [47].

signing tasks uniformly to workers), skill-based (assigning tasks by how likely the worker is to complete the task well and on time), random, and other (when the player used a strategy that was not listed).

We also convert the score features so that we may model positive outcome variables. The maximum score loss for any player due to low quality was below 50% and similarly for score lost due to lateness. Hence we convert this to a fraction and subtract it from 0.5 before multiplying by a factor of ten and rounding in order to get a rating out of five for both quality and timeliness (with five being the best and one the worst). These outcome variables are then one-hot encoded into ten binary variables for use in the model. The context variable representing which of the six game levels is taking place is encoded into six binary variables. In this experiment there are no constraints on the scenario other than those that force the one-hot encoded variables for the game level, quality rating, and timeliness rating to be mutually exclusive. A summary of the dataset we used is provided in Table 5.4 with further details on how this was adapted from the original given in Section C.3 of the appendix.

5.2.2 Results

As in our first experiment we begin by measuring the overall log likelihoods of both constrained and unconstrained PSDDs across training, validation, and test sets (in the same size ratios, 70:15:15) using an SPN learnt with Tachyon as a benchmark. These

Number of data points	7446
Number of variables	21
Random variables	Level 1 (L_1), ... , Level 6 (L_6)
Decision variables	Other (O), Load-balancing (L), Uniform (U), Skill-based (S), Random (R)
Outcome variables	Timeliness 1 (T_1), ... , Timeliness 5 (T_5), Quality 1 (Q_1), ... , Quality 5 (Q_5)
Constraints	$\bigvee_{i\in\{1,...,6\}} L_i$, $L_i \rightarrow \neg\bigvee_{j\in\{1,...,6\}\setminus i} L_j$ for all $i \in \{1,...,6\}$, $\bigvee_{i\in\{1,...,5\}} T_i$, $T_i \rightarrow \neg\bigvee_{j\in\{1,...,5\}\setminus i} T_j$ for all $i \in \{1,...,5\}$, $\bigvee_{i\in\{1,...,5\}} Q_i$, $Q_i \rightarrow \neg\bigvee_{j\in\{1,...,5\}\setminus i} Q_j$ for all $i \in \{1,...,5\}$
Model count	4800
Utilities recorded?	Yes (Self-reported Happiness Score)

Table 5.4: A summary of the teamwork management data used in our second experiment.

results are presented in Table 5.5 below. Once again, the constrained PSDD is smaller and more accurate, although the differences in the PSDD likelihoods are less pronounced here. We also note that the size of the SPN is approximately ten times that of the constrained PSDD, as was the case in our previous experiment as well. In the rest of our analysis we use the constrained PSDD as the basis for our model.

Model	Training	Validation	Test	Size
PSDD With Constraints	-5.541	-5.507	-5.457	370
PSDD Without Constraints	-5.637	-5.619	-5.556	931
SPN (Using Tachyon)	-7.734	-7.708	-7.658	3550

Table 5.5: Log likelihoods and sizes of the two PSDDs and the SPN learnt from the teamwork management data.

As well as using the overall log likelihood of the model to answer our first question, we investigate how often the model would employ each of the 32 possible strategies (where a strategy is represented by an assignment of values to the binary indicator decision variables) compared to the average participant. In our analysis during preprocessing the data we observed that the proportion of times a certain strategy is used is approximately the same in each of the six game levels (as can be seen in Figure

5.4) and so we make our comparison using the overall model predictions across all contexts.

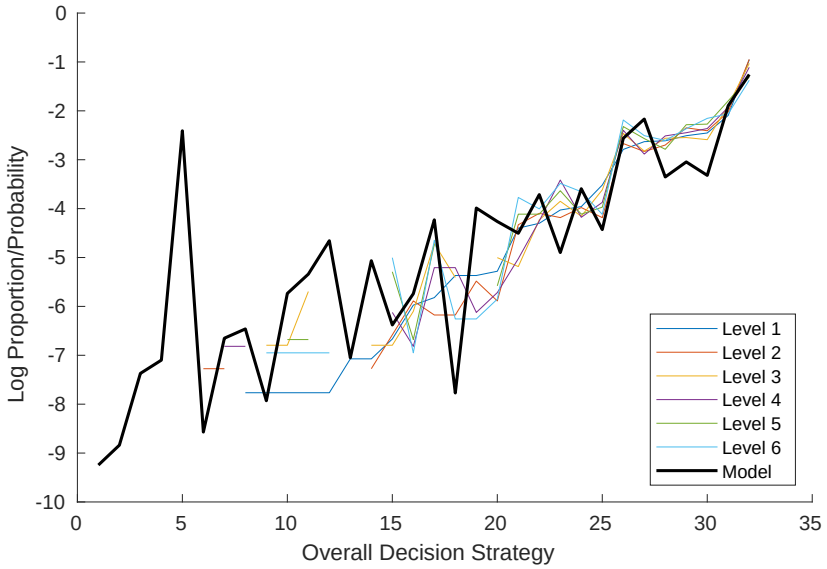

Figure 5.4: The log probability assigned to each possible decision strategy across all contexts by our model, compared to the log proportion of times each strategy was used in each level of the game. Strategies are sorted in ascending order by their proportion of use in level 1 and gaps in each plot represent strategies never used in that game level (thus having log proportion negative infinity).

In general the probabilities according to the model are relatively similar to the actual proportions in the data, though noisier. The discrepancies are particularly noticeable for decisions that were made very rarely (perhaps only once or twice in the entire dataset) or not at all. This is not hugely surprising as we might reasonably expect the model to have difficulty learning the exact probability of a decision given so few examples in the training data, and the lack of gaps in the black line compared to the others is a result of smoothing while learning so as to make the model more robust.

Moving on to consider utility in this problem, we provide two results. Firstly, given that the data is generated by human decisions and includes self-reported utility scores based on the results of these decisions, we investigate our assumption introduced in Section 3.2.2 that the fraction of times a decision is made is (linearly) proportional to the expected utility based on that decision. In order to do this we split the data up based on the context (game level) and produce a scatter plot of the proportion of times a set of decisions is made (a particular assignment to each of the five decision variables)

against the average utility (happiness score) of that decision. This plot can be seen in Figure 5.5.

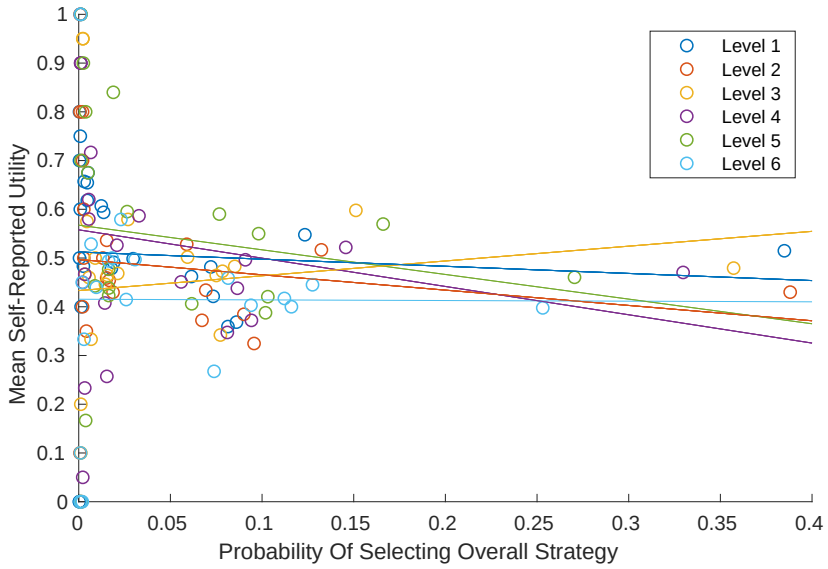

Figure 5.5: Each point represents a decision strategy in a level of the game; we compare the proportion of times it is used against the average self-reported utility that results from it. Each line is a least-squares best fit to the points in that level.

Our first observation is that there are three main clusters of points: strategies used very infrequently with a wide range of utilities, those used somewhat infrequently (about a tenth of times) with typically average utility, and those used very frequently (over a quarter of the time) that converge to very similar average utilities of around 0.5. Our second observation is that overall there is certainly no obvious positive linear correlation as our original assumption would imply. The reasons for this departure from the relationship specified by our assumption could be any one or combination of the following:

- Players do not play enough rounds of the game to find out which strategies reliably lead to higher scores and thus (presumably) higher utilities
- Players do not accurately self-report their strategies
- Players' strategies have relatively little impact on their overall utility based on the result of the game

We recall here that our assumption essentially comes down to supposing that people more often make decisions that result in greater utilities (for themselves, at least). The

eminent plausibility of this statement, along with the relatively high likelihood of at least one of the factors in the list above means we do not have enough evidence here to refute the statement, although certainly further empirical work would be required in order to demonstrate its truth. See Section 6.2.3 for further discussion along these lines.

In contrast to our previous experiment, the lack of a proportionality relationship seen in Figure 5.5 and the wide range of utilities resulting from strategies used a similar proportion of times indicates that in this scenario, our decision-making agents are what we referred to in Section 3.2.2 as *naive*. This naivety (or the noisiness of the decision-making process) would also explain each of the possible factors in the list above that would mean our proportionality assumption is violated. Upon reflection, naivety in such a domain is not hugely surprising, as we would not necessarily expect our agents to be particularly invested in or competent at playing the small game that they were asked to use as part of an experiment.

In our second result in answering our second question we learn a utility function based on the distribution of the data and assess its reasonableness. We can expect this function to be linear, as we assume that utility is positively correlated with the total score that the player achieves which in turn is measured by the sum of how well the player does with respect to quality and timeliness. However, how much utility a player gains from certain outcomes may vary across contexts, as some game levels may be harder than others and so a less good outcome in such a level might still result in a relatively high utility. In our analysis we thus inspect the average weights given to the outcome variables across the six game levels, which can be seen in the right half of Figure 5.6.

If our learnt utility function is correct it should place higher weights on the outcome variables corresponding to higher ratings, which as we can see is true for timeliness, but not quite true for quality as the five-out-of-five rating is in fact weighted only third highest. On further investigation we found that the learnt utility weights are in fact almost identical to the distribution of the outcomes in the data (which can be seen in the left half of Figure 5.6). Because our utility weights were learnt on the basis that players more often play strategies that will lead to better expected outcomes, the similarity between these two graphs adds further weight to our suggestion that in fact the self-reported strategies of players have very little to do with the final outcome.

To conclude our analysis of this experiment we come to assessing how reasonable the attributions of blame made by the model are, here using the original utility function

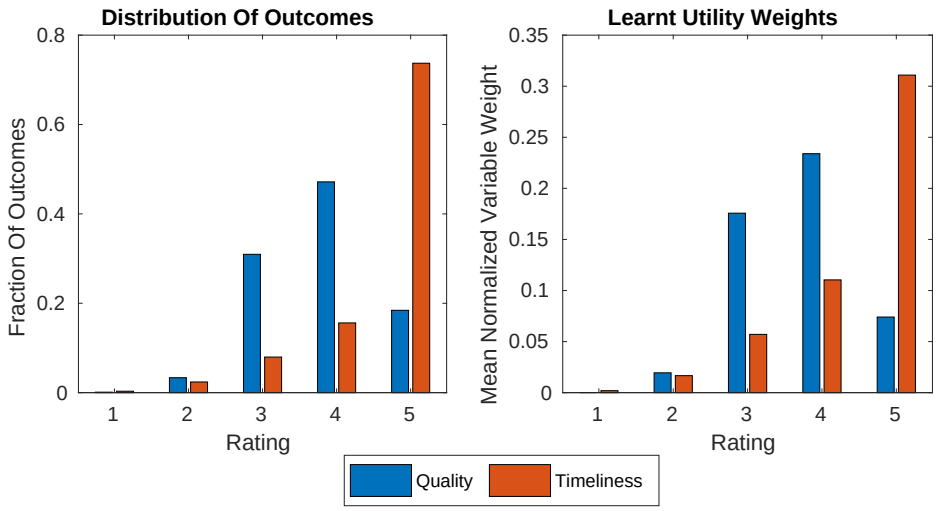

Figure 5.6: A comparison of the learnt utility weights for each of the outcome variables (to the right) and the proportion of times each outcome occurs in the data (to the left).

based on the average self-reported happiness scores of the players. We take the same approach as in the previous experiment and first examine cases in which the blameworthiness score should be zero, and then compare cases that should have lower or higher scores with respect to one another. In all of the blameworthiness scores below we use the cost importance measure $N = 1$.

As our first example we considered a particular level of the game (level one, to be precise) by choosing a different distribution $\mathbf{q}$ when generating our scores. Here a player is less likely to receive a low rating for quality (say, only Q_1 or Q_2) if they employ a skill-based strategy where tasks are more frequently allocated to better workers (S). Hence we should expect to find that $db_N(S, \neg S, Q_1 \vee Q_2) = 0$. Once again, our expectations are confirmed and this is exactly the result generated by our system. As before, the particular examples given here are stored in the outcomes directory of the distribution package, and though other tests were made we omit them here to save space.

Concerning our second set of tests for the reasonableness of our attributions of blame, we looked at the timeliness rating outcomes. A player is less likely to obtain the top timeliness rating (T_5) if they do not use a strategy that uniformly allocates tasks (U) compared to their not using a random strategy of allocation (R). Thus if our model is correct, there should be more blame attributed in the former case than in the latter. Using our notation from earlier we expect to find that $db_N(\neg U, \neg T_5) > db_N(\neg R, \neg T_5)$.

In actual fact we have $db_N(\neg U, \neg T_5) = 0.001585$ and $db_N(\neg R, \neg T_5) = 0$ (as it turns out that in general a player should avoid using a random strategy completely if they wish to obtain the top timeliness rating), thus confirming our hypothesis.

5.3 Trolley Problems

After testing our system on both synthetic and human-generated datasets we devised our own experiment using a small-scale survey to gather data about hypothetical moral decision-making scenarios. The full survey document including the consent form, instructions to participants, and questions can be found in Section C.4 of the appendix. Here we offer a slightly more compact overview of the experiment and justify our major design choices. The scenarios took the form of variants on the famous trolley problem [48] in which one is posed with the question of whether or not to flip a switch that would divert a speeding trolley (or train, in our case) onto a side track, killing one person tied up there but saving the lives of five others tied up on the main track.

We extended this idea, as is not uncommon in the literature, by introducing a series of different characters (groups of people) that might be on either track: one person, five people, 100 people, one's pet, one's best friend, and one's family. We also added two further decision options, pushing whoever is on the side track into the way of the train in order to save whoever is on the main track, and sacrificing oneself by jumping in front of the train, saving both characters in the process. The survey then took the form of asking each participant which of the four actions they would perform (the fourth being to do nothing) given each possible permutation of the six characters on the main and side tracks. Here we assumed that a character could not appear on both tracks in the same scenario, and so we had 30 possible options. The general setup can be seen in Figure 5.7, with locations A and B denoting the locations of people on the main track and side track respectively (a key for the symbols used is given in the original experiment description contained in the appendix).

Last of all, we added a probabilistic element to our scenarios whereby the switch only works with probability 0.6 (and so with probability 0.4 the train continues on its path and hits whoever is at location A on the main track), and pushing the character at location B onto the main track in order to stop the train succeeds with probability 0.8 (in the other case they are not pushed onto the main track and so survive, while the character already on the main track dies). Our reasoning behind this stemmed from the fact that people are generally more averse to actively pushing someone than to flipping

a switch [49], and people are certainly more averse to sacrificing themselves than doing either of the former. However, depending on how much one values the character on the main track's life, one might be prepared to perform a less desirable action in order to increase their chance of survival. We chose the values 0.6 and 0.8 so as to get a wider spread of data points (as opposed to all the participants making exactly the same choice in each scenario).

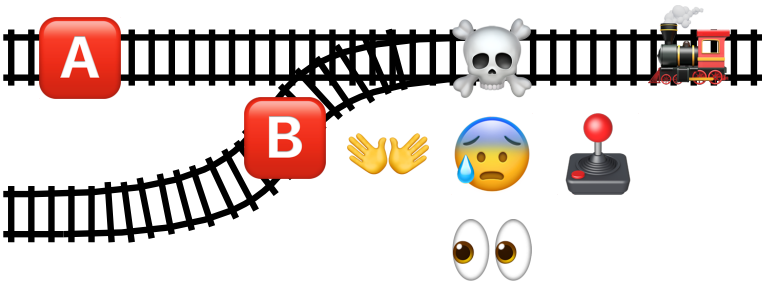

Figure 5.7: A diagram given to participants showing the layout of the experimental scenario and the four possible options available, with locations A and B instantiated by two particular characters depending on the context.

The scenarios were by their nature fairly rigidly constrained but also involved a degree of uncertainty and so were well-suited to our PSDD-based models. Further, while we did not ask participants directly for any utility assessments, the simplicity of the setup meant that a utility function that was both linear in the outcome variables and not context-relative was plausible. Making this assumption allowed us to reduce the amount of computation involved, and also to more easily make comparisons between the weights that the participants in fact assigned to the outcome variables (which we measured by proxy as the survival rate of each character, as explained in Section 5.3.2) and the weights learnt by our model.

5.3.1 Data

In total we asked twelve participants 30 questions each, with a question corresponding to one of the possible scenarios. For example, if there were five people on the main track (at location A) and their best friend on the side track (location B), a participant might respond by saying that they would do nothing, meaning that the five people would die and their best friend would survive. In order to encode the context, decisions, and outcomes into data points we used a binary indicator variable for each character

with regard to each location, giving us twelve context variables. Each of the four decision options is also represented by a single binary variable. Finally we used binary outcome variables corresponding to the survival of each of the six characters as well as one representing the survival of the participant in the hypothetical scenario.

Number of data points	360
Number of variables	23
Random variables	One Person On Track A (A_1), ... , Family On Track A (A_{Fa}), One Person On Track B (B_1), ... , Family On Track B (B_{Fa})
Decision variables	Do Nothing (N), Flip Switch (F), Push B (P), Sacrifice Oneself (S)
Outcome variables	One Person Lives (L_1), ... , Family Lives (L_{Fa}), You Live (L_Y)
Constraints	$\bigvee_{i\in\{1,...,Fa\}} A_i$, $\bigvee_{i\in\{1,...,Fa\}} B_i$, $\neg(A_i \wedge B_i)$ for all $i \in \{1,...,Fa\}$, $A_i \rightarrow \neg\bigvee_{j\in\{1,...,Fa\}\setminus i} A_j$ for all $i \in \{1,...,Fa\}$, $B_i \rightarrow \neg\bigvee_{j\in\{1,...,Fa\}\setminus i} B_j$ for all $i \in \{1,...,Fa\}$, $\bigvee_{D\in\{N,F,P,S\}} D$, $D \rightarrow \neg\bigvee_{D'\in\{N,F,P,S\}\setminus D} D'$, $(A_i \wedge N) \rightarrow \neg L_i$ for all $i \in \{1,...,Fa\}$, $(B_i \wedge N) \rightarrow L_i$ for all $i \in \{1,...,Fa\}$, $L_i \rightarrow (A_i \vee B_i)$ for all $i \in \{1,...,Fa\}$, $(S \wedge (A_i \vee B_i)) \rightarrow L_i$ for all $i \in \{1,...,Fa\}$, $L_Y \leftrightarrow \neg S$, $(L_i \wedge (P \vee F)) \rightarrow \neg\bigvee_{j\in\{1,...,Fa\}\setminus i} L_j$ for all $i \in \{1,...,Fa\}$, $(\neg L_i \wedge (P \vee F)) \rightarrow \bigvee_{j\in\{1,...,Fa\}\setminus i} L_j$ for all $i \in \{1,...,Fa\}$
Model count	180
Utilities recorded?	No

Table 5.6: A summary of the trolley problem data used in our third experiment.

Using our example from above, we would have $A_5 = 1$, $B_{Fr} = 1$, $N = 1$, $L_{Fr} = 1$, $L_Y = 1$, and all other variables equal to zero (an explanation of the variable names can be found in Table 5.6). Note here that we implicitly assume that a decision cannot result in a character living if that character was not on either track in the scenario. This was a choice we made in order to help the system to learn exactly which outcomes were the desired ones resulting from a particular decision, but is also a consequence

of our modelling outcomes as positive if they occur and neutral otherwise. Table 5.6 summarises the dataset we gathered, with further notes provided in Section C.4 of the appendix.

5.3.2 Results

Our first step is again to report the log likelihood scores and sizes for each of the three models we learn and for each of the three parts of the dataset, as can be seen in Table 5.7. These are calculated using the same shuffling and splitting procedures on the data and the same learning algorithms and parameters as in both of our previous experiments. We also see the same pattern with regard to the accuracy and sizes of each of the three models, although here the SPN struggles far more than in either of our previous experiments. This is likely to be due to the fact that our overall dataset for this experiment was particularly small.

Model	Training	Validation	Test	Size
PSDD With Constraints	-4.440	-4.510	-4.785	368
PSDD Without Constraints	-6.189	-6.014	-6.529	511
SPN (Using Tachyon)	-15.513	-16.043	-15.765	3207

Table 5.7: Log likelihoods and sizes of the two PSDDs and the SPN learnt from the trolley problems data.

We also wanted to see how well our model (based on the constrained PSDD) serves as a representation of the aggregated decision preferences of participants. In order to do so we calculated how likely the model would be to make particular decisions in each of the 30 scenarios and compared this with the average across participants in the survey. A summary of the predictions made by our model in each of the 30 contexts is presented in Section C.4 of the appendix, along with the average decisions made by participants. For reasons of space we focus here on a representative subset of this decision-making data. Namely, we compare our model's predictions to the actual decisions made in the five possible scenarios in which the best friend character is on the main track (see Figure 5.8).

In general the model's predictions are similar to the answers given in the survey, although the effect of smoothing our distribution during learning is noticeable, especially due to the fact that the model was learnt with relatively few data points. Despite this handicap the most likely decision in any of the 30 contexts according to the model

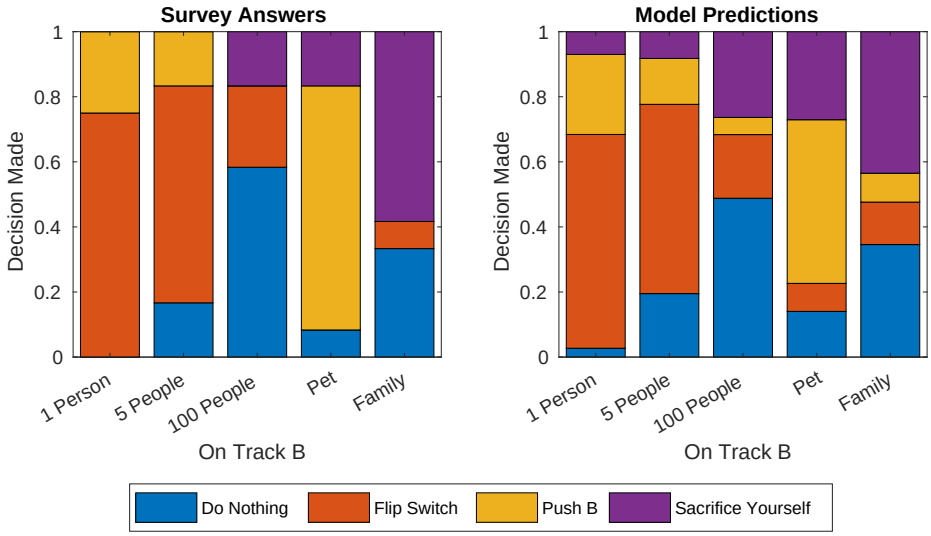

Figure 5.8: A comparison of the decisions made by participants and the predictions of our model in each of the five scenarios in which the best friend character is on the main track (A).

is in fact the majority decision in the survey, with the ranking of other decisions in each context also highly accurate. Hence, along with the likelihood scores given in Table 5.7, it seems fair to say that the model serves as a good representation of the aggregated decision-making preferences of the participants.

Unlike our other two experiments the data we gathered in our survey does not explicitly contain any utility information, meaning our system was forced to learn a utility function by using the probability distribution encoded by the PSDD[3]. Within the decision-making scenarios we presented, it seemed fairly plausible that the decisions made by participants were guided by weights that they assigned to the lives of each of the six characters and to their own life. Given that each of these is captured by a particular outcome variable we chose to construct a utility function that was linear in said variables. In other words, in each scenario we expect the participant to maximise their utility by saving the characters whose utility weights sum to the highest value.

Due to the symmetric nature of the set of contexts (each pair of characters appears together exactly twice, with their positions reversed in the second context) we also chose to make the utility function insensitive to context. This also seems quite reasonable, as we would not expect how much one values the life of a particular character to depend on which track that character was on, or whether they were on a track at all.

[3]We remind the reader that the theoretical and implementation details of this process are to be found in Sections 3.2.2 and 4.2.3 respectively.

In summary then, we used our system to learn a utility function that was linear in the outcome variables but not context-relative.

With no existing utility function to compare our learnt function to we chose to interpret the survival rates of each character as the approximate weight assigned to their lives by the participants. While the survival rate is a non-deterministic function of the decisions made in each context (due to the probabilistic nature of the outcomes resulting from the decisions), we assume that over the 360 trials in the experiment these rates average out enough for us to make a meaningful comparison between them and the weights learnt by our model. A visual representation of this comparison can be seen in Figure 5.9.

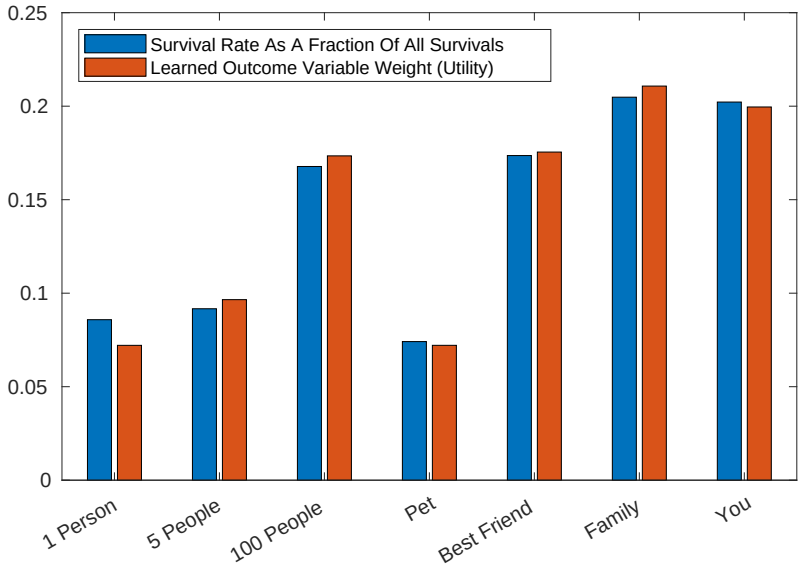

Figure 5.9: A comparison between the average survival rates of the seven characters (including the participants in the survey), normalised to sum to one, and the corresponding utility function weights learnt by our system.

It is immediately obvious that our system has captured the correct utility function to a high degree of accuracy. With that said, our assumption about using survival rates as a proxy for real utility weights does lend itself to favourable comparison with a utility function learnt from a probability distribution over contexts, decisions, and outcomes (which therefore includes survival rates). Given the setup of the experiment however, this assumption seems justified and furthermore to be in line with how most of the participants answered the survey. Indeed the relative utilities resulting from the survival of each of the characters (or at least the ordering) does appear plausible upon

consideration of how one might expect the average participant to respond.

As in the two experiments above we conclude by investigating the attributions of blame that the model makes, however this time we use our utility function learnt from data. Because of the symmetric nature of the 30 different contexts (as mentioned above), the probability of a particular character surviving as a result of a particular action across all contexts is just the same as the probability of that character not surviving. Hence in what follows we use our system's feature of being able to accept particular distributions $\mathbf{q}$ over the contexts in which we wish to attribute blame, allowing us to focus only on particular scenarios.

We begin with a simple check (one of many we made) that our system produces blameworthiness scores of zero in the appropriate cases of this experiment. Clearly in any of the possible contexts one should not be blamed at all for the the death of the character on the main track for flipping the switch (F) as opposed to doing nothing (N), because in the latter case they will die with certainty, but not in the former. Note that this is not to say one would not be blameworthy when compared to all other actions as one could, for example, have sacrificed oneself instead, saving all other lives with certainty. Choosing a scenario arbitrarily to illustrate this point, with one person on the side track and five people on the main track, we have $db_N(F,N,\neg L_5) = 0$ and $db_N(F,\neg L_5) = 0.307142$ (with our measure of cost importance $N = 0.7619$, 1.1 times the negative minimum cost of any action and the second of our two standard values offered to the user of our pipeline, as explained in Section 4.3)[4].

Before concluding both this subsection and chapter with our final example, it is proper to mention that our system did struggle with a few of the checks of the kind above in this experiment. On further inspection this was due to the accuracy of the probability estimates and in particular resulted from our focusing on only one particular context at a time. In such checks we essentially relied on the accuracy of probability estimates learnt from a tiny twelve datapoints (one for each participant), and so it is unsurprising that such estimates led our calculations astray. These problems did not seem to impact our other examples (including the one below) in which we used multiple contexts or larger datasets. Overall, the few incorrect scores we generated were a symptom of the size of our dataset in this experiment, which was necessarily limited by time and resources, rather than our implementation itself, but it is nonetheless important to point out these failures (we consider the limitations of our work more fully

[4]Please note that subscript N here is cost importance, and the standard N refers to the *do nothing* variable.

in Section 6.1.2).

The second example that we present is slightly more complex. Consider the scenario in which one is standing by the side of the tracks but is not wearing one's glasses. There is a large crowd of a hundred or so people on the main track, but one is unable to tell from this distance if the five or so people on the side track are strangers or one's family. Clearly the more likely it is that the family is on the side track, the more responsible one is for their deaths ($\neg L_{Fa}$) if one, say, flips the switch (F) to divert the train. Conversely, we would also expect there to be *less* blame attributed for the deaths of the 100 people ($\neg L_{100}$) say, if one did nothing (N), the more likely it is that the family is on the side track. This is because the cost of performing any other action would be much higher if they were (as they would be more likely to die).

We compare cases where there is a 0.3 probability that the family is on the track against a 0.6 probability and for all calculations use the cost importance measure $N = 1$. Because of this, not only would we expect the blame for the death of the family to be higher when pulling the switch in the latter case, we would expect the value to be approximately twice as high as in the former case. Once again our hypothesis is vindicated and we compute values $db_N(F, \neg L_{Fa}) = 0.263996$ and $db_N(F, \neg L_{Fa}) = 0.553846$ respectively. Similarly, when considering blame for the deaths of the 100 people due to doing nothing, we find that $db_N(N, \neg L_{100}) = 0.152738$ in the former case and that $db_N(N, \neg L_{100}) = 0.109729$ in the latter case (when the cost of performing another action is higher). As in our other experiments, these results are recorded in the relevant sub-directory of our package distribution.

Chapter 6

Conclusion and Future Work

We conclude with a summary of the project that gives the primary successes and limitations of our work, before suggesting several possible extensions and promising directions for future research.

6.1 Summary

In summarising we return to consider our original motivation and objectives as described in Section 1.1 and highlight the successes and limitations of our work with respect to these, alongside other issues we came across while completing the project. Finally, we offer some concluding remarks before proceeding to consider possible future work.

6.1.1 Successes

On a theoretical level, we were able to successfully map the BF to a concrete instance of a probabilistic model. Further, we were able to adhere almost exactly to the specification of decision-making scenarios used in this original work (see Section 3.2.1 for minor exceptions). At the same time we exploit many of the desirable properties of our chosen model, the PSDD. For example, the individual models our system creates are tractable with respect to our blameworthiness calculations, interpretable with respect to the decision-making scenarios they represent, and can be both learnt from data and incorporate given logical restrictions.

These results were supported by the successful implementation of our mapping and its combination with a host of other algorithmic procedures in order to form an

overall pipeline. This pipeline is flexible in its usage, allowing the user to input and create various kinds of model, and to specify constraints or parameters in order to suit their particular needs. Moreover, it has a streamlined and user-friendly interface that stores outputs and inputs in a systematic and accessible manner. While it is far from cutting-edge software it effectively and efficiently fulfills our original specification and needs.

Finally, we were able to demonstrate the applicability of our system to three decision-making scenarios, each based on a different domain, using a mixture of synthetic and human-generated data. We showed that our models are typically accurate representations of the probabilistic distributions over contexts, decisions, and outcomes that they are learnt from. Our learnt utility functions, while simple in nature and limited in scope, are still able to capture much of what we expect to find, and in some scenarios (as in Section 5.3.2) are able to match human preferences with high accuracy using very little data. Using these two elements we also successfully generated several examples of blameworthiness scores that are, prima facie, in line with our intuitions and are an encouraging result.

6.1.2 Limitations

Perhaps the most obvious limitation in our project was its scale, particularly with regard to the range and complexity of decision-making scenarios to which we applied our final system, and the depth of our evaluation thereafter. While this is somewhat understandable given the scope and nature of the project, a true assessment of how suitable our system is would require its testing in many more cases. In particular, the models we learnt in our experiments were relatively small, the largest containing only 23 variables. Tractability concerns at this scale are less important, and so ideally we would have liked to experiment using data from more complicated environments in order to better evaluate our system's performance in this regard. Further, while we attempted to use data from a range of settings there are countless other examples that would be amenable to the kinds of analysis we performed.

Potentially the most problematic (and certainly one of the least well-grounded) assumptions in the project is that there exists a (linear) proportionality relationship between the probability of an agent making a certain decision in a context, and the expected utility resulting from the decision in that context. We were unable to provide evidence for this other than appealing to intuition and had neither the time nor

resources to dig deeper into the theory surrounding the issue. Consequently, the utility functions learnt by our system were somewhat idealistic in nature, and our learning process was relatively unsophisticated and so potentially unable to capture the subtleties and complexities of a real world utility function. This links to our point above about the scale of our experiments and the scenarios therein. We consider this limitation and possible future related work in Section 6.2.3.

One of the inherent limitations resulting from our choice of PSDDs is that they are not particularly well-placed to capture dynamic environments or sequential decision-making scenarios. In general, our models represent scenarios where decisions are made simultaneously based on a fixed context, which then produce outcomes. While this is certainly broad enough to capture a great many decision-making processes, it is also clear that not all such processes fit neatly into this format. As we saw in our first experiment (see Section 5.1) it is sometimes possible to add extra variables to deal with these limitations, but in general this approach seems to us rather ad hoc and may also lead to unnecessarily large models. We offer one suggestion for an alternative approach in Section 6.2.1.

6.1.3 Concluding Remarks

In this project we brought together two existing resources, one theoretical (the BF) and one more technical (PSDDs), in order to create a system that is able to attribute blame to decisions in a variety of settings. We showed the correctness of these attributions both theoretically, with respect to the original framework, and also empirically as part of our three experiments. Further, we demonstrated the efficiency of the blameworthiness attributions by successfully implementing our system as an algorithmic pipeline that utilises a selection of existing and novel resources. This efficiency is supported by the complexity results we presented.

While the applicability of our work was not explored here in as much depth as we would have liked, we view our results as something of a proof of concept; a more modest achievement but one in keeping with the scope and nature of the project, as well as our original objectives. In particular, we hope that our work here goes some way towards bridging the gap between the existing philosophical work on moral responsibility and the existing technical work on decision-making in AI systems. With that said, there are certainly other aspects of the project that warrant further investigation and perhaps revision. Particular examples of these are presented in Section 6.2 and

seem to us to be promising and interesting opportunities for future research.

6.2 Future Work

We leave the reader with a short introduction to some further ideas and avenues that we suggest could make fruitful research opportunities. At least some of these we hope to pursue ourselves.

6.2.1 Complex Decisions and Alternative Models

As explained within Section 6.1.2, one limitation of our system is that in using PSDDs as our underlying model, it can be difficult to represent more complex decisions. However, in any alternative we would also like to retain our method of variable encoding (so as to be able to apply the BF), the relatively intuitive nature of our existing models, and their ability to be learnt from data. One way to satisfy all of these concerns may be to use factored MDPs (FMDPs), an extension of the traditional MDP architecture that can provide compact representations of large problems by factoring variables into independent subsets [50]. This is in contrast to the standard approach of enumerating the entire possible state space based on these variables (which is exponentially larger). Factoring the problem also allows for simpler and more compact transition and reward functions, further contributing to the model's overall level of interpretability.

Even a tentative look at how to map the BF to this markedly different architecture is beyond the scope of this project, but it seems to be a particularly promising avenue for future work. FMDPs contain all of the primary elements required for such a mapping, although it is not quite clear yet how the definitions in the BF extend to planning and to utility functions that are necessarily sequential in their evaluation. Other advantages of using FMDPs are that they can also be learnt from data (as originally shown in [51]) and that much of the recent and important work in IRL can be far more easily applied to them. Indeed, MDPs are the general model of choice for reinforcement learning as a whole; an extremely rich area of great relevance to many of the themes and ideas in our work. We hope that this project might serve as something of an example for more complex mappings and models in future.

6.2.2 Moral Principles and Formal Methods

A point that was briefly raised in Section 2.3.3 is the natural ability of the PSDDs in our system to capture elements of both consequentialist and deontological ethical theories, or in terms more directly related to the model itself, both distributions over preferences and rule-based elements. Other than the fact that this is trivially true if one restricts the distribution before learning (for example, in the trolley problems experiment we could have encoded logical constraints such that any human life should be prioritised over the life of a pet), we did not develop this idea further.

What seems to be more interesting than the above example is whether it is possible to extract moral principles from models that are learnt without such constraints. While it is of course unlikely that the whole of the decision-making data used for learning would be consistent with anything but the most trivial of rules, it may instead be possible to verify that certain rules are followed with some high probability, and perhaps even use automated reasoning procedures to deduce and thus predict further ethical principles to be applied to new scenarios, or those in which there was little original data.

As an extremely simple example, if a self-driving car has learnt (with high probability, say) to prioritise the safety of A over B, and B over C, but at no point has it ever encountered a situation concerning the safety of A with respect to C, we would hope that some form of transitivity relation would be adhered to. There has already been some existing work done along similar lines to these (see, for example, [52] and further papers by the same authors), though little to none of it involves learning from data, which is an important part of our project. An integration of these two approaches is therefore something we might hope to work towards in future.

6.2.3 Realistic Utility Functions and Human Biases

There is convincing and increasingly widely-accepted evidence that human decision-makers are not the traditional utility-maximising agents of classical economic and decision theory (perhaps the most influential criticism of the more traditional view comes from Kahneman and Tversky [53]). Further, humans are in general subject to bias and ignorance when they make decisions. Thus, if our system is to learn from decision-making data generated by humans it must also take these factors into account, as our ideal attributor of blame would be immune to such irrationality [30]. The issues here are certainly ones we were aware of when pursuing this project, but chose not to in-

vestigate in any great detail so as to simplify our work and remain within the bounds of our time and resources. In future however, we believe these concerns to be worthy of further research and are hopeful that they might be integrated into the work we have presented here.

Following on from Section 6.1.2, we return to the question of how realistic our assumptions about utility functions (and therefore the utility values that our system learns) are. As was already mentioned, a full survey of the relevant theoretical literature surrounding this question was beyond the scope and far-removed from the general domain of our work here, though it is clearly relevant. In future, however, it would benefit our understanding to examine exactly how, or indeed whether, our assumptions are in fact validated across a variety of decision-making domains. This would also tie in closely to the assessment of our generated blameworthiness scores by comparing them to human judgements. Such an approach is of course more easily described than done, but does seem necessary due to the nature of the data our system uses and of our overall objectives in this project.

Appendix A

Acronyms and Nomenclature

This appendix contains a list of acronyms and a summary of all nomenclature used in this report, the former presented alphabetically and the latter approximately following the structure of Chapter 2.

A.1 Acronyms

AC	Arithmetic Circuit
AI	Artificial Intelligence
BF	Blameworthiness Framework
DC	Decision Circuit
FMDP	Factored Markov Decision Process
ID	Influence Diagram
MDP	Markov Decision Process
MPE	Most Probable Evidence
PGM	Probabilistic Graphical Model
PSDD	Probabilistic Sentential Decision Diagram
SDD	Sentential Decision Diagram
SEF	Structural Equations Framework
SPMN	Sum-Product-Max Network
SPN	Sum-Product Network
TPM	Tractable Probabilistic Model

A.2 Nomenclature

$\mathcal{X}$	exogenous variables *or* random variables
$\mathcal{V}$	endogenous variables
O	outcome variables
$\mathcal{D}$	decision variables
$\mathcal{R}$	range function
$\mathcal{S}$	signature
$\mathcal{F}$	structural equations
F_V	structural equation corresponding to variable V
w	world
$\mathbf{X}$	context
M	causal model
$(M, \mathbf{X})$	causal setting
$\mathcal{K}$	possible causal settings
$\mathcal{Y}$	variables
$\mathbf{Y}$	assignment of values to variables in $\mathcal{Y}$
Y	variable
y	variable value
A	action variable
φ	primitive event
$[\mathcal{Y} \leftarrow \mathbf{Y}]\varphi$	setting the values of the variables in $\mathcal{Y}$ to $\mathbf{Y}$ would cause φ
ψ	causal formula
$[\![\psi]\!]$	causal settings satisfying ψ
$\mathbf{p}$	probability distribution
$\mathbf{u}$	utility function
$\delta_{a,a',\varphi}$	probability of φ resulting from performing a rather than a'
c	cost function
$\mathbf{O}_{A \leftarrow a}$	setting of the outcome variables when action a is performed
N	cost importance measure
$db_N(a, a', \varphi)$	degree of blameworthiness of a for φ relative to a' (with respect to N)
$db_N(a, \varphi)$	degree of blameworthiness of a for φ (with respect to N)

Appendix B

Code Documentation and Samples

All code used in the project is available online [41]. Here we present summaries and key details of each of the separate files, along with some small code samples referenced earlier in the report.

B.1 Documentation

Fully commented code is available within the package online. Below we succinctly describe the purpose of each function and class in the five primary files that we wrote. As well as these files the package also contains the following directories:

- `data` stores the training, validation, and test datasets for each model
- `models` stores the constituents of each model, including vtrees, SDDs, PSDDs, constraints, utility functions, and alternative probability distributions
- `output` stores a text file that records the blameworthiness results of the most recent execution of the pipeline for each model
- `resources` stores the LearnPSDD and SDD library packages
- `tests` stores several un-commented and un-optimised example files used to generate results of the form reported in our experiments

B.1.1 model.py

This file contains:

- functions for learning vtrees, SDDs, and PSDDs using exisiting resources

- other helper functions

`setup_learnpsdd(PWD)` sets up the LearnPSDD package within the present working directory `PWD`.
`set_names(NAME)` creates global variable names for the constituents of a model with name `NAME`.
`psdd_from_sdd(NAME,DATA,PWD)` creates a PSDD by parameterising an existing SDD via `sdd2psdd` using data, the availability of which is recorded using an array of indicator variables `DATA`.
`psdd_from_vtree(NAME,DATA,PWD)` learns an (unconstrained) PSDD from an existing vtree and data using `learnPsdd`.
`vtree_from_data(NAME,DATA,PWD)` learns a vtree from given data using `learnVtree` by minimizing average pairwise mutual infomation between branches (via Blossom).
`sdd_from_constraints(NAME,PWD)` uses the SDD package to construct an SDD from logical constraints.
`create_sdd_manager(NAME)` creates an SDD manager based on logical constraints that are stored in an existing text file (the path of which is determined by `NAME`).
`re_fit(NAME,DATA,PWD)` optionally refits and replaces a PSDD based on an existing one while preserving any logical constraints.

B.1.2 psdd.py

This file contains:

- class definitions for PSDDs and the nodes they contain
- a function for loading PSDDs from PSDD files outputted by LearnPSDD
- functions for evaluating PSDDs and performing MPE inference
- other helper functions

`Node` is the class for nodes in a PSDD with attributes `node_id`, `node_type`, `variable`, `children`, `coefficient`, and `value`.
`Psdd` is the class for a PSDD with attributes `nodes`, `variables`, `random`, `decision`, `outcome`, `root`, `utility`, and `weights`.
`create_psdd(psdd_file)` reads in a `psdd_file` generated by LearnPSDD and creates a corresponding data structure for computation.

`evaluate(psdd,evidence)` returns the probability of `evidence` (a partial assignment of values to the variables) in a `psdd`.
`eval_psdd(id,psdd)` takes a `psdd` and a node `id` and recursively computes the value of a sub-PSDD rooted at that node.
`mpe(psdd,evidence)` computes the MPE of a `psdd` given partial `evidence`, returning the most probable assignment of variables (that are not in the evidence) and the overall probability of the total assignment.
`max_psdd(id,psdd,max_branches)` recursively evaluates a `psdd` with max nodes instead of sums and stores the maximizing children in `max_branches`.
`set_variables(id,psdd,max_branches)` recursively sets the variables of a `psdd` by following the branches stored in `max_branches`, starting at node `id`.
`split_variables(psdd,decision_variables,outcome_variables=[])` splits the variables of a `psdd` into `decision_variables`, `outcome_variables` (if they exist), and random variables.
`merge(dict1,dict2)` merges two dictionaries `dict1` and `dict2`.
`save_variables(PSDD,NAME,PWD)` saves the allocation of variables in a `PSDD` to a separate file (named using `NAME` and stored in `PWD`) for later use.
`add_variables(PSDD,variables_file)` sets the variables in a `PSDD` according to an external `variables_file`.
`print_psdd(psdd)` prints out the constituents of a particular instance of the `psdd` data structure.
`print_node(node)` prints out the constituents of a particular instance of the `node` data structure.

B.1.3 utility.py

This file contains:

- functions for learning and normalising utility functions
- a function for calculating the expected utility of decisions
- other helper functions

`learn_utility(psdd,linear,context_relative,utility_file=None)` either accepts and normalises a utility function from a `utility_file` if given or calculates one from data if not, depending on whether or not the function is `linear` and whether or not it is `context_relative`.

`normalise(u,psdd)` accepts a (context-relative) utility function `u` and normalises it to the range [0,1].
`expected_utility(causal_setting,decisions,psdd)` takes a `causal_setting` and `decisions` and calculates the expected utility from the decisions in that setting.
`save_utility(PSDD,NAME,PWD)` saves the utility function of a `PSDD` to a separate file (named using `NAME` and stored in `PWD`) for later use.
`add_utility(PSDD,utility_file)` sets the utility function of a `PSDD` according to an external `utility_file`.

B.1.4 blameworthiness.py

This file contains:

- functions for calculating the probabilities of events, the costs of actions, and blameworthiness scores
- other helper functions

`prob(A,a,phi,psdd,q)` calculates the probability that decision `a` (a possible value of variable `A`) results in outcome `phi` (a Boolean formula), using distribution over contexts `q` (which may be different to that encoded by the `psdd`).
`delta(A1,a1,A2,a2,phi,psdd,q)` measures how much more likely it is that `phi` will result from performing `a1` than from performing `a2`.
`cost(A,a,psdd,q)` computes the cost of performing an action (in other words, making a decision) `a`.
`form_q(psdd,prob_file=None)` forms a new distribution over contexts from a `prob_file` if given, or if not uses the distribution already encoded in a `psdd`.
`db_n(A1,a1,A2,a2,phi,psdd,N,q)` computes the blameworthiness of action `a1` for outcome `phi` relative to action `a2` using the measure of cost importance `N`.
`db(A,a,B,phi,psdd,N,q)` computes the general blameworthiness of action `a` for outcome `phi`.

B.1.5 pipeline.py

This file contains:

- functions for creating new models and sub-directories in which to store them

- functions for taking various inputs from the user including vtrees, SDDs, PSDDs, constraints, data, and functions
- functions for conducting and storing the results of blameworthiness queries
- functions for printing out results and information to the user of the pipeline

`get_model(PWD)` gets the name of an existing model within the relevant sub-directory of `PWD`.

`get_name()` gets a name for a new model as input from the user and creates the relevant sub-directories.

`get_psdd(NAME,PWD)` either accepts a PSDD file as input (and copies it to the relevant models folder in `PWD`, renaming it according to the given `NAME`) or asks about an SDD file.

`get_sdd(NAME,DATA,PWD)` either accepts an SDD file as input (and copies it to the relevant models folder in `PWD`, renaming it according to the given `NAME`) or asks about constraints.

`get_constraints(NAME,DATA,PWD)` either accepts constraints and learns an SDD then a PSDD, or learns a PSDD straight from a vtree.

`get_vtree(NAME,DATA,PWD)` either accepts a vtree or asks about data to learn one from (the availability of which is recorded using an array of indicator variables `DATA`).

`get_data(NAME,PWD)` accepts paths to data (a training dataset, and possibly validation and test datasets as well) from the user.

`fit_again(NAME,PWD,psdd_first=False,DATA=None)` optionally re-learns a PSDD based on an existing one while preserving any logical constraints (with default parameters indicating that the PSDD has not been learnt first before an SDD, and that the availability of possible datasets is currently unknown).

`define_variables()` asks the user which variables represent decisions or outcomes.

`get_utility()` accepts a utility function or provides a specification for learning one.

`utility_props()` asks the user for the properties of a utility function (whether it is linear and whether it is context-relative).

`get_q(PSDD)` checks whether the probability distribution over contexts is different (from the one encoded in the `PSDD`) in this scenario.

`get_decisions()` asks the user for the decisions to consider when attributing blame.

`get_phi(PSDD)` accepts a Boolean formula of primitive events that represents an outcome from the user in order to compute blame for this outcome.

`meu(decisions,q,PSDD)` calculates the maximum expected utility for a particular

decision variable in `decisions` using a `PSDD` and possibly a separate distribution over contexts `q`.

`get_n(decisions,q,PSDD)` aks the user if they wish to specify a value for N, the measure of cost importance used in generating blameworthiness scores.

B.2 Samples

This section contains the code samples referred to in Sections 4.1.1 and 4.3 respectively.

B.2.1 SDD Manager

Below is an SDD manager file used to construct SDDs by compiling the constraints therein. The example below corresponds to our original example from Section 2.3.2 although has been handwritten instead of automatically generated by our constraint parsing function (see Section 4.2.1). It is based on example code provided as part of the SDD package [38].

```
#include <stdio.h>
#include <stdlib.h>
#include "sddapi.h"

//  returns an SDD node representing ( node1 => node2 )
SddNode* sdd_imply(SddNode* node1, SddNode* node2,
SddManager* manager)
{return sdd_disjoin(sdd_negate(node1,manager),node2,manager);}

//  returns an SDD node representing ( node1 <=> node2 )
SddNode* sdd_equiv(SddNode* node1, SddNode* node2,
SddManager* manager)
{return sdd_conjoin(sdd_imply(node1,node2,manager),
sdd_imply(node2,node1,manager),manager);}

int main(int argc, char** argv) {

////////// SET UP VTREE AND MANAGER //////////

SddLiteral var_count = 4;
int auto_gc_and_minimize = 0;
SddManager* m = sdd_manager_create(var_count,auto_gc_and_minimize);
Vtree* v = sdd_manager_vtree(m);

SddLiteral R = 1, U = 2, L = 3, W = 4;
SddNode* delta = sdd_manager_true(m);
SddNode* alpha;
SddNode* beta;
```

```
////////// CONSTRUCT THEORY //////////

// !U -> !L
alpha = sdd_negate(sdd_manager_literal(U,m),m);
beta = sdd_negate(sdd_manager_literal(L,m),m);
alpha = sdd_imply(alpha,beta,m);
delta = sdd_conjoin(delta,alpha,m);

// (R & !U) <-> W
alpha = sdd_negate(sdd_manager_literal(U,m),m);
alpha = sdd_conjoin(sdd_manager_literal(R,m),alpha,m);
beta = sdd_manager_literal(W,m);
alpha = sdd_equiv(alpha,beta,m);
delta = sdd_conjoin(delta,alpha,m);

////////// SAVE VTREE AND SDD //////////

printf("Saving SDD and vtree ...\n");
sdd_vtree_save("output/example.vtree",v);
sdd_vtree_save_as_dot("output/examplevtree.dot",v);
sdd_save("output/example.sdd",delta);
sdd_save_as_dot("output/examplesdd.dot",delta);

////////// CLEAN UP //////////

sdd_manager_free(m);
return 0;}
```

B.2.2 Pipeline Execution

The trace of the pipeline below is a copy of the text that is inputted and outputted to the command line during the program's run-time. This is completely unedited other than our excluding some of the setup information printed by LearnPSDD and also the printed summary of the learnt model, both of which we omit for reasons of space. Once again we re-use our original example from Section 2.2.2 and even re-compute the same blameworthiness score (albeit with different results due to the learning process, normalisation of the utility function, and different cost measure parameter N).

```
================================
Welcome to the blame attributor!
================================

Would you like to use an existing model? (y/n) n

We begin by constructing a PSDD

Please enter a name for the new model: example
Do you have an existing PSDD to input? (y/n) n
Please enter the path to the training data: example.data
Please enter the path to the validation data (or enter none): none
Please enter the path to the test data (or enter none): none
Data inputted successfully!
Do you have an existing SDD to input? (y/n) n
Do you have an existing vtree to input? (y/n) n
```

```
Learning vtree from training data...
Setting up LearnPSDD...

... [LEARNPSDD INFORMATION PRINTED OUT HERE]

LearnPSDD set up successfully!

... [LEARNPSDD INFORMATION PRINTED OUT HERE]

vtree learnt from training data successfully!
Would you like to specify any logical constraints on the variables?
(y/n) y
Logical contstraints may be entered one at a time using the
following prefix notation:
Variables       ==   A, B, C, ... (or 1, 2, 3, ...)
Not             ==   !(x)
Or              ==   |(x,y)
And             ==   &(x,y)
Implication     ==   >(x,y)
Biconditional   ==   =(x,y)
For example, (A > B) | !C is written |(>(A,B),!(C))
When you have finished entering all constraints, enter done
First, please enter the number of variables in the model: 4
Now please enter the logical constraints using the notation above:
>(!(2),!(3))
=(4,&(1,!(2)))
done
Learning SDD from constraints...
Creating SDD manager...
SDD manager created successfully!
Model count: 6
SDD learnt successfully from constraints!
Learning PSDD from SDD and data file(s)...
Setting up LearnPSDD...

... [LEARNPSDD INFORMATION PRINTED OUT HERE]

LearnPSDD set up successfully!

... [LEARNPSDD INFORMATION PRINTED OUT HERE]

PSDD learnt from SDD and data file(s) successfully!
If the PSDD was learnt without smoothing then it needs to be
re-fitted before learning a utility function
Would you like to re-fit your existing PSDD? (y/n) y
Re-fitting the PSDD using the existing PSDD, vtree, and data
file(s)...
Setting up LearnPSDD...

... [LEARNPSDD INFORMATION PRINTED OUT HERE]

PSDD re-fitted successfully!
Creating PSDD data structure...
PSDD data structure created successfully!

We then add a utility function

First, please specify the decision and output variables (either
as letters or numbers)
Enter the decision variables as a comma-separated list (for
example A,C,G or 1,3,7): 2
```

```
Enter the outcome variables similarly, or enter none if they are
not modelled: 3,4
Do you have an existing utility function to input? (y/n) y
Utility functions should be inputted as data files where each
entry specifies a setting of the variables and a utility
For example, with 4 variables, one row might appear as 0,1,1,0,5.7
Please now enter the path to such a file: example.util
Utility function inputted successfully!
Is the utility function linear in the outcome variables? (y/n) y
Is the utility function context-relative? (y/n) n
Utility function created successfully!

Our final model is thus complete:

... [MODEL PRINTED OUT HERE]

We may now compute blameworthiness scores

Would you like to specify a different probability distribution
over contexts than that in the data? (y/n) n
Please input the decision taken in the form D=d or 3=d, for
example, where d is 0 or 1: 2=1
Please enter an alternative decision in the same form, or enter
none if you want to calculate general blameworthiness: 2=0
To specify phi (a Boolean formula of primitive events) we use
Python syntax
We also refer to outcome variable i as o[i]
For example, B=1 -> E=0 would be entered as
(o[2] != 1) | (o[5] == 0)
Please now input the Boolean formula over outcome variables:
o[3]==1
Would you like to specify a particular value for N (cost
indifference)? (y/n) n
Standard values for N are either:
- the maximum possible utility (1)
- 1.1 times the maximum expected utility of any decision
To use the former enter mpu, to use the latter enter meu: meu
Please indicate all of the (binary) variables corresponding
to the decision in a comma-separated list: 2
The blameworthiness score of decision B=1 for outcome C==1
relative to decision B=0 (using N=0.88) is: 0.435065

Would you like to compute another blameworthiness score? (y/n) n

=====================================
Thanks for using the blame attributor!
=====================================
```

Appendix C

Datasets and Further Analysis

All datasets used in the project are available online [41]. Here we present some extra details and analysis for each of them.

C.1 Example

In the model based on our example from Sections 2.2.2 and 2.3.2 we used a small handcrafted dataset to reflect the probability distribution given in Table 2.1. As the data files themselves are simply rows of comma-separated zeroes and ones we include Table C.1 to allow the reader to more easily follow the variable naming convention used in our pipeline.

Number	Letter	Abbreviation	Description
1	A	R	It is raining
2	B	U	Woman collects umbrella
3	C	L	Woman is late to work
4	D	W	Woman gets wet from the rain

Table C.1: A description of the variables in the example dataset.

C.2 Lung Cancer Staging

Here we provide the remaining probabilities used to generate our synthetic dataset from Section 5.1.1. These are used to fully specify the ID from [42] shown in Figure 5.1 and are based upon statistics from a series of medical studies:

$\mathbf{p}(CT_+|MM) = 0.82$ $\mathbf{p}(M_+|MM) = 0.82$ $\mathbf{p}(S_T|T, S_{DP}) = 0.963$
$\mathbf{p}(CT_-|MM) = 0.18$ $\mathbf{p}(M_-|MM) = 0.18$ $\mathbf{p}(S_T|\neg T, S_{DP}) = 0.998$
$\mathbf{p}(CT_+|\neg MM) = 0.19$ $\mathbf{p}(M_+|\neg MM) = 0.005$ $\mathbf{p}(S_{DP}|M) = 0.995$
$\mathbf{p}(CT_-|\neg MM) = 0.81$ $\mathbf{p}(M_-|\neg MM) = 0.995$ $\mathbf{p}(S_{DP}|\neg M) = 1$

As in the previous section we also provide a description of the variables as they are represented in the datasets, and hence in the model and the outputs it generates. This can be seen in Table C.2.

Number	Letter	Abbreviation	Description
1	A	MM	Patient has mediastinal metastases
2	B	$CT_{N/A}$	No CT scan result
3	C	CT_+	Positive CT scan result
4	D	CT_-	Negative CT scan result
5	E	$M_{N/A}$	No mediastinoscopy result
6	F	M_+	Positive mediastinoscopy result
7	G	M_-	Negative mediastinoscopy result
8	H	CT	CT scan performed
9	I	M	Mediastinoscopy performed
10	J	T	Thoractomy performed
11	K	S_{DP}	Diagnosis procedures survived
12	L	S_T	Treatment survived

Table C.2: A description of the variables in the lung cancer staging datasets.

C.3 Teamwork Management

The dataset used in our second experiment (see Section 5.2) varied quite substantially from the original data in [45] and so here we offer a brief description of how it was modified to better suit our purposes. In order to clean and pre-process the data (which initially consisted of 9855 data points instead of our 7446) we performed the following tasks:

- Removed all entries where the player did not provide any response to the emotional variables

- Removed all unnecessary variables including game and player IDs, AI score (the player of the game measures their success against an AI opponent), reported facial expression (representing the player's emotions after completing the game), other emotion variables that were not happiness, and time-stamps
- Removed all entries where the sum of the final score, score lost due to lateness, and score lost due to low quality was below 95% or above 105% (in theory this sum should have been exactly 100% but in practice this was often not reflected in the data)
- Converted score loss to quality and timeliness ratings (as described in Section 5.2.1) and removed the overall score variable
- Removed the placeholder digit from the player strategy indexes before separating each value into binary digits and one-hot encoding the remaining variables

A description of the variables as used by our model and as can be seen in the relevant datasets is given in Table C.3.

Finally, we showed in Figure 5.4 that the distribution over overall strategies differs little across each of the six game levels and that our model captures this distribution fairly well. In Figure C.1 we also show the slightly weaker statement that the distribution over *individual* strategy variables also differs little across levels. Here, across all contexts, our model is particularly accurate.

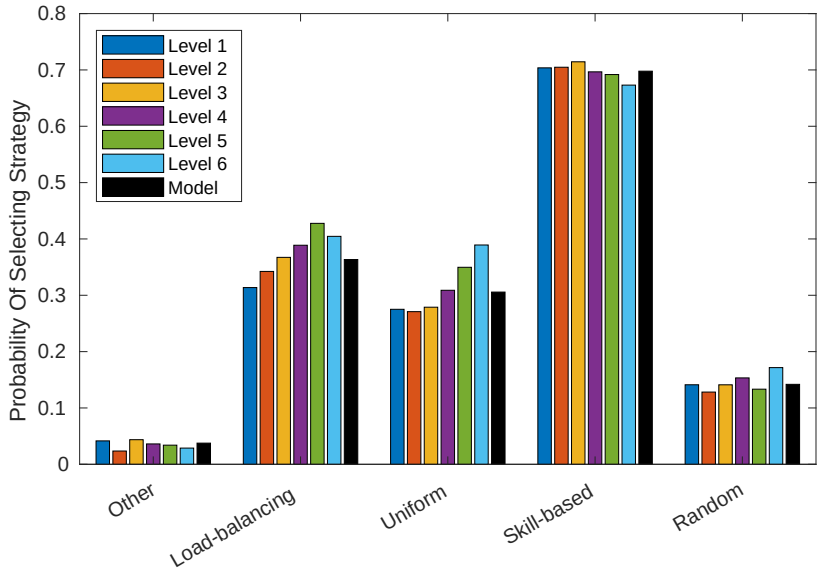

Figure C.1: The probabilities of a player using each individual strategy in each level, compared with our model's prediction over all levels.

Number	Letter	Abbreviation	Description
1	A	L_1	Level 1
2	B	L_2	Level 2
3	C	L_3	Level 3
4	D	L_4	Level 4
5	E	L_5	Level 5
6	F	L_6	Level 6
7	G	O	Other strategy
8	H	L	Load-balancing strategy
9	I	U	Uniform strategy
10	J	S	Skill-based strategy
11	K	R	Random Strategy
12	L	Q_1	Quality rating 1
13	M	Q_2	Quality rating 2
14	N	Q_3	Quality rating 3
15	O	Q_4	Quality rating 4
16	P	Q_5	Quality rating 5
17	Q	T_1	Timeliness rating 1
18	R	T_2	Timeliness rating 2
19	S	T_3	Timeliness rating 3
20	T	T_4	Timeliness rating 4
21	U	T_5	Timeliness rating 5

Table C.3: A description of the variables in the teamwork management datasets.

C.4 Trolley Problems

As in each of the other sections within this appendix we display the naming conventions used for the variables in the model and its outputs as a reference (see Table C.4) so that the reader may more easily understand the existing results and use our system to generate new blameworthiness scores, if they so desire.

Here we also present full versions of the stacked bar charts (akin to those of Figure 5.8) used to represent the decisions made by the participants (Figure C.2) and the decisions predicted by our model (Figure C.3).

To conclude this appendix we attach the full survey document that was provided to each of the participants. It comprises an introductory explanation and consent form, details of the decision-making scenarios and experimental setup, and a table for the participant to record their choices.

Number	Letter	Abbreviation	Description
1	A	A_1	One person on main track
2	B	B_1	One person on side track
3	C	A_5	Five people on main track
4	D	B_5	Five people on side track
5	E	A_{100}	100 people on main track
6	F	B_{100}	100 people on side track
7	G	A_{Pet}	Pet on main track
8	H	B_{Pet}	Pet on side track
9	I	A_{Fr}	Best friend on main track
10	J	B_{Fr}	Best friend on side track
11	K	A_{Fa}	Family on main track
12	L	B_{Fa}	Family on side track
13	M	N	Do nothing
14	N	F	Flip switch
15	O	P	Push character on side track
16	P	S	Sacrifice yourself
17	Q	L_1	One person lives
18	R	L_5	Five people live
19	S	L_{100}	100 people live
20	T	L_{Pet}	Pet lives
21	U	L_{Fr}	Best friend lives
22	V	L_{Fa}	Family lives
23	W	L_Y	You live

Table C.4: A description of the variables in the trolley problem datasets.

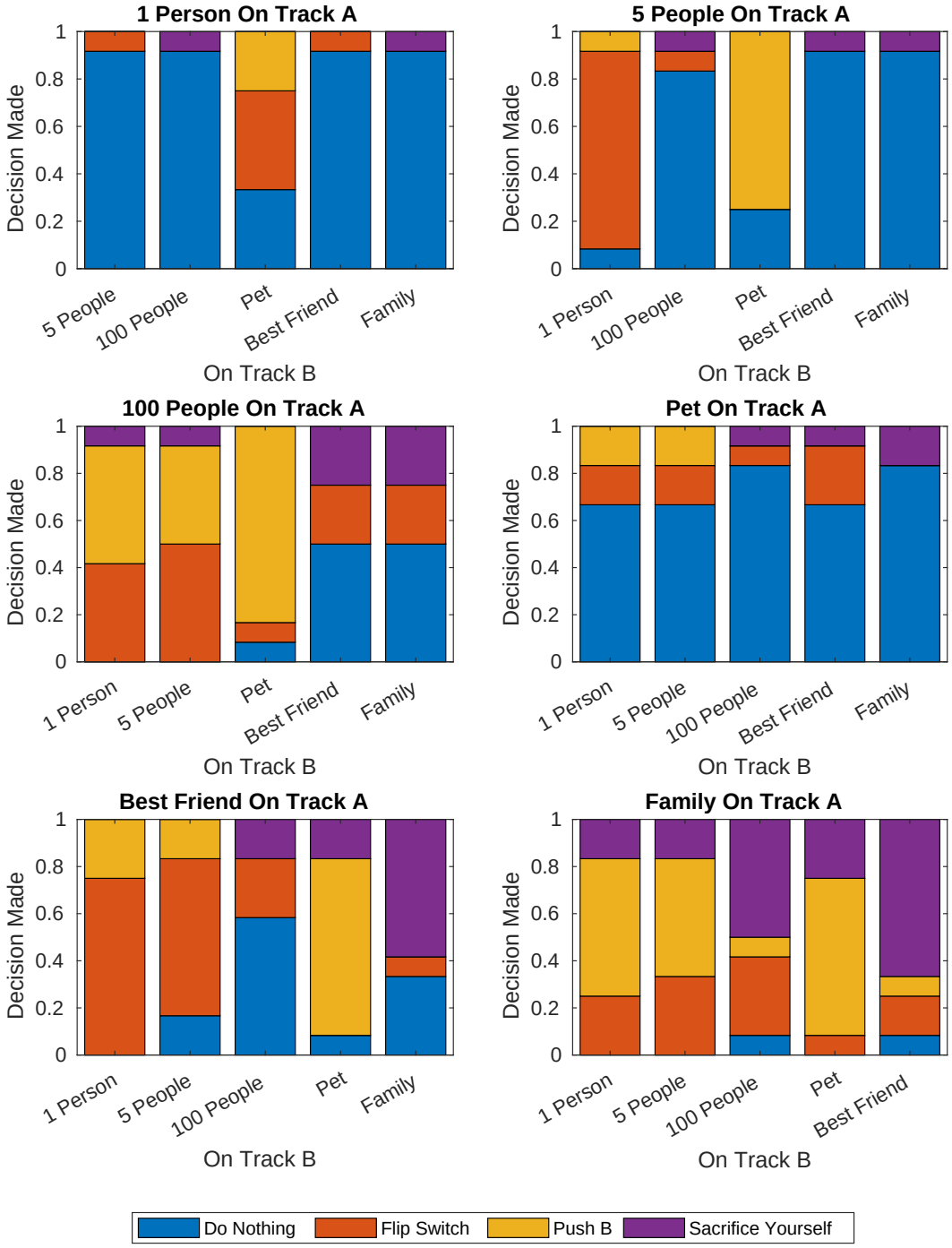

Figure C.2: Survey results from our third experiment.

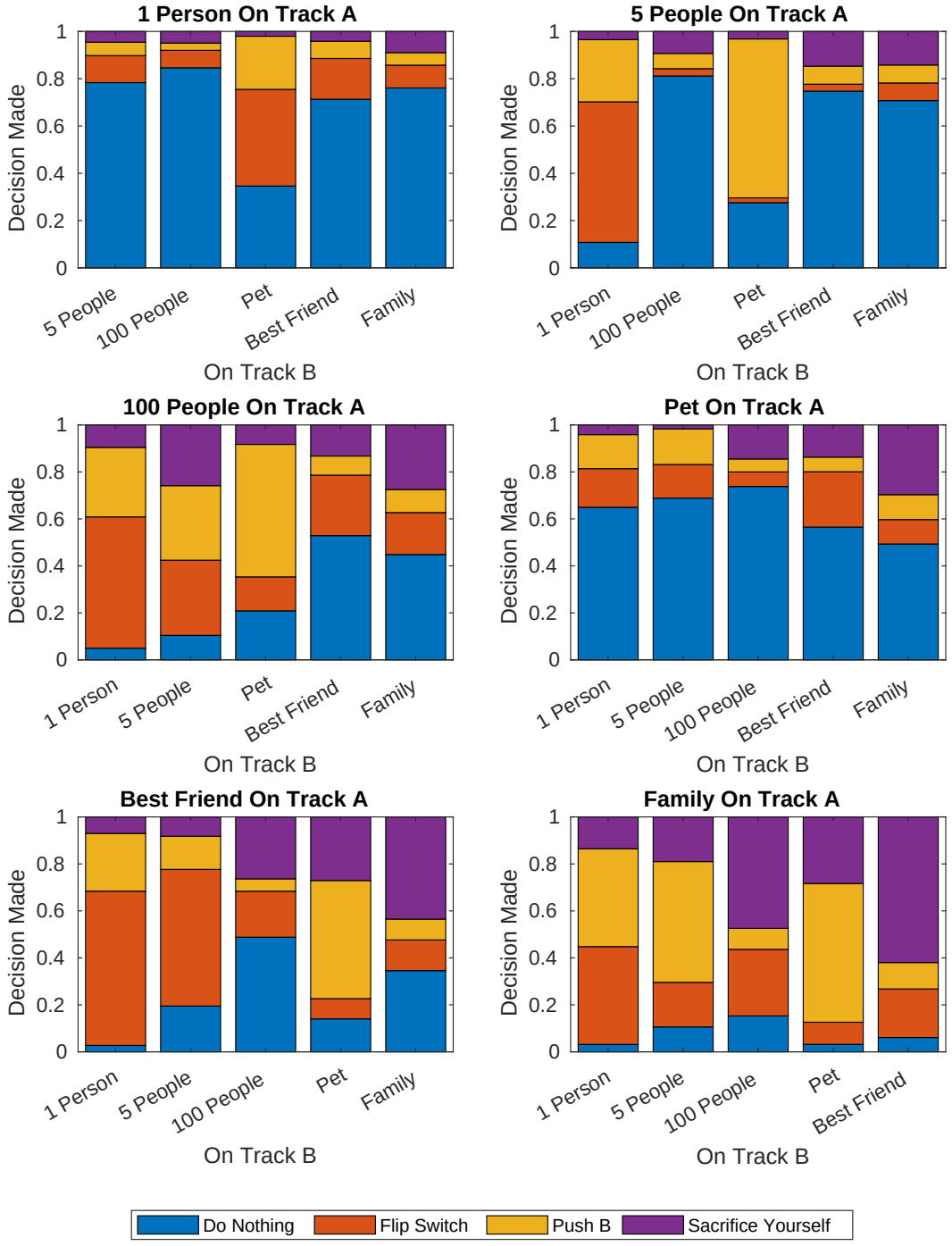

Figure C.3: Model predictions from our third experiment.

Attributing Blame To Decisions Using Tractable Probabilistic Models: An Experiment

Lewis Hammond
University of Edinburgh
s1731219@sms.ed.ac.uk

1 Consent

In this project we are attempting to create models that can correctly and efficiently make attributions of blame to decisions in a variety of complex, uncertain settings. One of the primary features of these models is the ability to be learnt from decision-making data, which we are using this survey to collect.

In the survey we will ask you for a series of answers to multiple-choice questions corresponding to actions in a variety of hypothetical scenarios. The precise details of these scenarios can be found overleaf. Despite the title of the project there will be no blame or judgement attached to your answers here; we are merely interested in your opinions.

Completing the survey should take approximately 10 minutes and you have the right to withdraw your consent or discontinue participation at any time without penalty. You also have the right to refuse to answer particular questions. All data collected is recorded entirely anonymously and your individual privacy will be maintained in all published and written data resulting from the project. You may request to have your data deleted at any time. There are no notable benefits or risks from taking the survey.

This research in the project follows the School of Informatics research ethics code and this experiment has undergone the appropriate screening and ethical approval processes (RT #2943). By signing below and agreeing to take part in the experiment you confirm that you have read the above consent form, you have understood it, and you accept the conditions therein.

Signature: ..... Date: .....

2 Experiment Details

Consider the following scenario, as shown in Figure 1. You (😰) are standing nearby a junction on a railway track at which a side track splits off from the main track. There are two locations, A on the main track (🅰️) and B on the sidetrack (🅱️), where an animal, a person, or a group of people are situated and unable to move. In this general description we refer just to A and B, although in each scenario these variables will be instantiated differently. Tragically, there is a train with a brake failure (🚂) speeding along the track in your direction and you only have time to make one of the following four choices:

- The first option is to *do nothing* (👀), in which case the train will continue along the main the track and A will die with probability 1.0, though B will live with probability 1.0.
- You may instead *flip the switch* (🕹️) to divert the train onto the side track. However the switch is faulty and only works some of the time, hence there is a 0.6 probability that A lives and B dies, but also a 0.4 probability that B lives and A dies.
- You can choose to *push B* (🫴) onto the main track to attempt to stop the train before it reaches A. There is a 0.8 probability that this succeeds (in which case B dies and A will survive), but a probability of 0.2 that you fail (in which case A still gets hit by the train and dies, but B survives).
- The final alternative is to *sacrifice yourself* (☠️) by jumping in front of the train. This selfless act will result in saving the lives of both A and B with probability 1.0, although you will die instead.

Finally we have six possible instantiations of A and B: one person (👤), five people (👥), 100 people (💯), your pet (🐶), your best friend (👨), or your family (👨‍👩‍👧). We assume that each can only appear at either A or B, giving rise to 30 possible combinations. We also ignore any problems concerning realism (not least how you might push 100 people onto a train track in a split second).

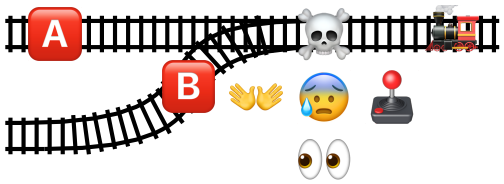

Figure 1: The unfortunate scene

3 Survey

Please fill in the table (mark one blank column per row) below according to which action you would take in the scenario as described above, with A and B instantiated according to the two left-most columns.

A	B	Do Nothing	Flip Switch	Push B	Sacrifice Yourself
👤	👥				
👤	💯				
👤	🐶				
👤	👨				
👤	👯				
👥	👤				
👥	💯				
👥	🐶				
👥	👨				
👥	👯				
💯	👤				
💯	👥				
💯	🐶				
💯	👨				
💯	👯				
🐶	👤				
🐶	👥				
🐶	💯				
🐶	👨				
🐶	👯				
👨	👤				
👨	👥				
👨	💯				
👨	🐶				
👨	👯				
👯	👤				
👯	👥				
👯	💯				
👯	🐶				
👯	👨				

Bibliography

[1] J. Y. Halpern and M. Kleiman-Weiner, "Towards formal definitions of blameworthiness, intention, and moral responsibility," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pp. 1853–1860, 2018.

[2] D. Kisa, G. Van den Broeck, A. Choi, and A. Darwiche, "Probabilistic sentential decision diagrams," in *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 558–567, 2014.

[3] L. Hammond, "Attributing blame to decisions made by learnt tractable deep symbolic models," *Informatics Project Proposal*, 2018.

[4] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission," in *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1721–1730, 2015.

[5] J.-F. Bonnefon, A. Shariff, and I. Rahwan, "The social dilemma of autonomous vehicles," *Science*, vol. 352, no. 6293, pp. 1573–1576, 2016.

[6] L. Hammond, "Explainable medical decision support systems," *Informatics Research Review*, 2018.

[7] W. Sinnott-Armstrong, "Consequentialism," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab (Stanford University), 2015. *https://plato.stanford.edu/archives/win2015/entries/consequentialism/*, Accessed 2018-08-17.

[8] L. Alexander and M. Moore, "Deontological ethics," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab (Stanford University), 2016. *https://plato.stanford.edu/archives/win2016/entries/ethics-deontological/*, Accessed 2018-08-17.

[9] R. Hursthouse and G. Pettigrove, "Virtue ethics," in *The Stanford Encyclopedia of Philosophy* (E. N. Zalta, ed.), Metaphysics Research Lab (Stanford University), 2016. *https://plato.stanford.edu/archives/win2016/entries/ethics-virtue/*, Accessed 2018-08-17.

[10] G. Debreu, "Representation of a preference ordering by a numerical function," in *Decision Processes* (M. Thrall, R. C. Davis, and C. H. Coombs, eds.), pp. 159–165, John Wiley and Sons, New York, 1954.

[11] J. Von Neumann and O. Morgenstern, *Theory of games and economic behavior*. Princeton University Press, Princeton, 2 ed., 1947.

[12] J. Y. Halpern and J. Pearl, "Causes and explanations: A structural-model approach. Part I: Causes," *The British Journal for the Philosophy of Science*, vol. 56, no. 4, pp. 843–887, 2005.

[13] G. F. Cooper, "The computational complexity of probabilistic inference using bayesian belief networks," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 393–405, 1990.

[14] A. Darwiche, "SDD: A new canonical representation of propositional knowledge bases," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, p. 819, 2011.

[15] S. B. Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 509–516, 1978.

[16] H. Poon and P. Domingos, "Sum-product networks: A new deep architecture," in *IEEE International Conference on Computer Vision Workshops*, pp. 689–690, 2011.

[17] A. Darwiche, "A logical approach to factoring belief networks," in *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 409–420, 2002.

[18] A. Choi and A. Darwiche, "On relaxing determinism in arithmetic circuits," *arXiv preprint arXiv:1708.06846*, 2017.

[19] Y. Liang, J. Bekker, and G. Van den Broeck, "Learning the structure of probabilistic sentential decision diagrams," in *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence*, pp. 134–145, 2017.

[20] D. Gunning, *Explainable Artificial Intelligence (XAI) - DARPA-BAA-16-53*. Defense Advanced Research Projects Agency, 2016.

[21] H. Chockler and J. Y. Halpern, "Responsibility and blame: A structural-model approach," *Journal of Artificial Intelligence Research*, vol. 22, pp. 93–115, 2004.

[22] M. Kleiman-Weiner, T. Gerstenberg, S. Levine, and J. B. Tenenbaum, "Inference of intention and permissibility in moral decision making," in *Proceedings of the 37th Annual Conference of the Cognitive Science Society*, pp. 1123–1128, 2015.

[23] M. Kleiman-Weiner, R. Saxe, and J. B. Tenenbaum, "Learning a commonsense moral theory," *Cognition*, vol. 167, pp. 107–123, 2017.

[24] D. Bhattacharjya and R. D. Shachter, "Evaluating influence diagrams with decision circuits," *arXiv preprint arXiv:1206.5257*, 2012.

[25] M. A. Melibari, P. Poupart, and P. Doshi, "Sum-product-max networks for tractable decision making," in *Proceedings of the 15th International Conference on Autonomous Agents & Multiagent Systems*, pp. 1419–1420, 2016.

[26] A. Choi, G. Van den Broeck, and A. Darwiche, "Tractable learning for structured probability spaces: A case study in learning preference distributions," in *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pp. 2861–2868, 2015.

[27] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the 17th International Conference on Machine Learning*, pp. 663–670, 2000.

[28] C. L. Baker, R. Saxe, and J. B. Tenenbaum, "Action understanding as inverse planning," *Cognition*, vol. 113, no. 3, pp. 329–349, 2009.

[29] T. D. Nielsen and F. V. Jensen, "Learning a decision maker's utility function from (possibly) inconsistent behavior," *Artificial Intelligence*, vol. 160, no. 1-2, pp. 53–78, 2004.

[30] O. Evans, A. Stuhlmüller, and N. D. Goodman, "Learning the preferences of ignorant, inconsistent agents," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pp. 323–329, 2016.

[31] V. Conitzer, W. Sinnott-Armstrong, J. S. Borg, Y. Deng, and M. Kramer, "Moral decision making frameworks for artificial intelligence," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pp. 4831–4835, 2017.

[32] J. Greene, F. Rossi, J. Tasioulas, K. B. Venable, and B. C. Williams, "Embedding ethical principles in collective decision support systems," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pp. 4147–4151, 2016.

[33] M. Dehghani, E. Tomai, K. D. Forbus, and M. Klenk, "An integrated reasoning approach to moral decision-making," in *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pp. 1280–1286, 2008.

[34] W. Mao and J. Gratch, "Modeling social causality and responsibility judgement in multi-agent interactions," *Journal of Artificial Intelligence Research*, vol. 44, pp. 223–273, 2012.

[35] Scalable Cooperation (MIT Media Lab), *Moral Machine*, 2016. *http://moralmachine.mit.edu/*, Accessed 2018-08-14.

[36] R. Kim, M. Kleiman-Weiner, A. Abeliuk, E. Awad, S. Dsouza, J. Tenenbaum, and I. Rahwan, "A computational model of commonsense moral decision making," *arXiv preprint arXiv:1801.04346*, 2018.

[37] R. Noothigattu, S. Gaikwad, E. Awad, S. Dsouza, I. Rahwan, P. Ravikumar, and A. D. Procaccia, "A voting-based system for ethical decision making," *arXiv preprint arXiv:1709.06692*, 2017.

[38] Automated Reasoning Group (University Of California, Los Angeles), *The SDD Package 2.0*, 2018. *http://reasoning.cs.ucla.edu/sdd/*, Accessed 2018-08-17.

[39] Y. Liang, *LearnPSDD Manual*. Automated Reasoning Group (University Of California, Los Angeles), 2018. *Unpublished*.

[40] R. Peharz, R. Gens, F. Pernkopf, and P. Domingos, "On the latent variable interpretation in sum-product networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 10, pp. 2030–2044, 2017.

[41] L. Hammond, *Dissertation Package*. University of Edinburgh, 2018. *https://github.com/LRHammond/dissertation*, Accessed 2018-08-23.

[42] R. F. Nease Jr and D. K. Owens, "Use of influence diagrams to structure medical decisions," *Medical Decision Making*, vol. 17, no. 3, pp. 263–275, 1997.

[43] D. J. Malenka, G. L. Colice, C. Jacobs, and J. R. Beck, "Mediastinal staging in non-small-cell lung cancer," *Medical Decision Making*, vol. 9, no. 4, pp. 231–242, 1989.

[44] A. Kalra, *Tachyon*. University of Waterloo, 2017. *https://github.com/KalraA/Tachyon*, Accessed 2018-08-23.

[45] H. Yu, Z. Shen, C. Miao, C. Leung, Y. Chen, S. Fauvel, J. Lin, L. Cui, Z. Pan, and Q. Yang, "A dataset of human decision-making in teamwork management," *Scientific Data*, vol. 4, p. 160127, 2017.

[46] H. Yu, X. Yu, S. F. Lim, J. Lin, Z. Shen, and C. Miao, "A multi-agent game for studying human decision-making," in *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1661–1662, 2014.

[47] Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly, *The Agile Manager Game*, 2014. *http://agilemanager.algorithmic-crowdsourcing.com/*, Accessed 2018-08-14.

[48] J. J. Thomson, "The trolley problem," *The Yale Law Journal*, vol. 94, no. 6, pp. 1395–1415, 1985.

[49] P. Singer, "Ethics and intuitions," *The Journal of Ethics*, vol. 9, no. 3-4, pp. 331–352, 2005.

[50] C. Boutilier, R. Dearden, and M. Goldszmidt, "Exploiting structure in policy construction," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 1104–1111, 1995.

[51] T. Degris, O. Sigaud, and P.-H. Wuillemin, "Learning the structure of factored markov decision processes in reinforcement learning problems," in *Proceedings of the 23rd International Conference on Machine Learning*, pp. 257–264, 2006.

[52] L. Dennis, M. Fisher, M. Slavkovik, and M. Webster, "Formal verification of ethical choices in autonomous systems," *Robotics and Autonomous Systems*, vol. 77, pp. 1–14, 2016.

[53] D. Kahneman and A. Tversky, "Prospect theory: An analysis of decision under risk," *Econometrica*, vol. 47, no. 2, pp. 263–291, 1979.