UNCERTAINTY AND PROBABILISTIC REASONING

UNCERTAINTY AND PROBABILISTIC REASONING

PROBABILISTIC REASONING

Context-specific Sign-propagation in Qualitative Probabilistic Networks

Silja Renooij[1], **Simon Parsons**[2,*], and **Linda C. van der Gaag**[1]
[1]Institute of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{silja,linda}@cs.uu.nl
[2]Department of Computer Science, University of Liverpool
Chadwick Building, Liverpool L69 7ZF, United Kingdom
s.d.parsons@csc.liv.ac.uk

Abstract

Qualitative probabilistic networks represent probabilistic influences between variables. Due to the level of representation detail provided, knowledge about influences that hold only in specific contexts cannot be expressed. The results computed from a qualitative network, as a consequence, can be quite weak and uninformative. We extend the basic formalism of qualitative probabilistic networks by providing for the inclusion of context-specific information about influences and show that exploiting this information upon inference has the ability to forestall unnecessarily weak results.

1 Introduction

Qualitative probabilistic networks are qualitative abstractions of probabilistic networks [Wellman, 1990], introduced for probabilistic reasoning in a qualitative way. A qualitative probabilistic network encodes statistical variables and the probabilistic relationships between them in a directed acyclic graph. Each node A in this digraph represents a variable. An arc $A \rightarrow B$ expresses a probabilistic influence of the variable A on the probability distribution of the variable B; the influence is summarised by a qualitative sign indicating the direction of shift in B's distribution. For probabilistic inference with a qualitative network, an efficient algorithm, based upon the idea of propagating and combining signs, is available [Druzdzel & Henrion, 1993].

Qualitative probabilistic networks can play an important role in the construction of probabilistic networks for real-life application domains. While constructing the digraph of a probabilistic network is doable, the assessment of all probabilities required is a much harder task and is only performed when the network's digraph is considered robust. By eliciting signs from domain experts, the obtained qualitative probabilistic network can be used to study and validate the reasoning behaviour of the network prior to probability assessment; the signs can further be used as constraints on the probabilities to be assessed [Druzdzel & Van der Gaag, 1995]. To be able to thus exploit a qualitative probabilistic network, it should capture as much qualitative information from the application domain as possible. In this paper, we propose an extension to the basic formalism of qualitative networks to enhance its expressive power for this purpose.

Probabilistic networks provide, by means of their digraph, for a qualitative representation of the conditional independences that are embedded in a joint probability distribution. The digraph in essence captures independences between nodes, that is, it models independences that hold for all values of the associated variables. The independences that hold only for specific values are not represented in the digraph but are captured instead by the conditional probabilities associated with the nodes in the network. Knowledge of these latter independences allows further decomposition of conditional probabilities and can be exploited to speed up inference. For this purpose, a notion of *context-specific independence* was introduced for probabilistic networks to explicitly capture independences that hold only for specific values of variables [Boutilier *et al.*, 1996; Zhang & Poole, 1999].

A qualitative probabilistic network equally captures independences between variables by means of its digraph. Since its qualitative influences pertain to variables as well, independences that hold only for specific values of the variables involved cannot be represented. In fact, qualitative influences implicitly *hide* such context-specific independences: if the influence of a variable A on a variable B is positive in one context, that is, for one combination of values for some other variables, and zero in all other contexts – indicating independence – then the influence is captured by a positive sign. Also, positive and negative influences may be hidden: if a variable A has a positive influence on a variable B in some context and a negative influence in another context, then the influence of A on B is modelled as being ambiguous.

As context-specific independences basically are qualitative by nature, we feel that they can and should be captured explicitly in a qualitative probabilistic network. For this purpose, we introduce a notion of *context-specific sign*. We extend the basic formalism of qualitative networks by providing for the inclusion of context-specific information about influences and show that exploiting this information upon inference can prevent unnecessarily weak results. The paper is organised as follows. In Section 2, we provide some preliminaries concerning qualitative probabilistic networks. We present two examples of the type of information that can be hidden in

*This work was partly funded by the EPSRC under grant GR/L84117

qualitative influences, in Section 3. We present our extended formalism and associated algorithm for exploiting context-specific information in Section 4. In Section 5, we discuss the context-specific information that is hidden in the qualitative abstractions of two real-life probabilistic networks. In Section 6, we briefly show that context-specific information can also be incorporated in qualitative probabilistic networks that include a qualitative notion of strength of influences. The paper ends with some concluding observations in Section 7.

2 Qualitative probabilistic networks

A *qualitative probabilistic network* models statistical variables as nodes in its digraph; from now on, we use the terms variable and node interchangeably. We assume, without loss of generality, that all variables are binary, using a and $\bar{a}$ to indicate the values *true* and *false* for variable A, respectively. A qualitative network further associates with its digraph a set of *qualitative influences*, describing probabilistic relationships between the variables [Wellman, 1990]. A qualitative influence associated with an arc $A \rightarrow B$ expresses how the values of node A influence the probabilities of the values of node B. A *positive qualitative influence*, for example, of A on B, denoted $S^+(A, B)$, expresses that observing higher values for node A makes higher values for node B more likely, regardless of any other influences on B, that is,

$$\Pr(b \mid ax) \geq \Pr(b \mid \bar{a}x),$$

for any combination of values x for the set X of parents of B other than A. The '$+$' in $S^+(A, B)$ is termed the influence's *sign*. A negative qualitative influence S^-, and a zero qualitative influence S^0, are defined analogously. If the influence of node A on node B is non-monotonic or unknown, we say that it is *ambiguous*, denoted $S^?(A, B)$.

The set of influences of a qualitative probabilistic network exhibits various properties [Wellman, 1990]. The *symmetry* property states that, if $S^\delta(A, B)$, then also $S^\delta(B, A)$, $\delta \in \{+, -, 0, ?\}$. The *transitivity* property asserts that a sequence of qualitative influences along a chain that specifies at most one incoming arc per node, combine into a single influence with the $\otimes$-operator from Table 1. The *composition* property asserts that multiple influences between two nodes along parallel chains combine into a single influence with the $\oplus$-operator.

$\otimes$	$+$	$-$	0	$?$
$+$	$+$	$-$	0	$?$
$-$	$-$	$+$	0	$?$
0	0	0	0	0
$?$	$?$	$?$	0	$?$

$\oplus$	$+$	$-$	0	$?$
$+$	$+$	$?$	$+$	$?$
$-$	$?$	$-$	$-$	$?$
0	$+$	$-$	0	$?$
$?$	$?$	$?$	$?$	$?$

Table 1: The $\otimes$- and $\oplus$-operators.

A qualitative network further captures *qualitative synergies* between three or more nodes; for details we refer to [Druzdzel & Henrion, 1993; Wellman, 1990].

For inference with a qualitative network, an efficient algorithm is available [Druzdzel & Henrion, 1993]. The basic idea of the algorithm is to trace the effect of observing a node's value on the other nodes in the network by message passing between neighbouring nodes. For each node, a node sign is determined, indicating the direction of change in the node's probability distribution occasioned by the new observation given all previously observed node values. Initially, all node signs equal '0'. For the newly observed node, an appropriate sign is entered, that is, either a '$+$' for the observed value *true* or a '$-$' for the value *false*. Each node receiving a message updates its node sign and subsequently sends a message to each neighbour whose sign needs updating. The sign of this message is the $\otimes$-product of the node's (new) sign and the sign of the influence it traverses. This process is repeated throughout the network, building on the properties of symmetry, transitivity, and composition of influences. Since each node can change its sign at most twice, once from '0' to '$+$' or '$-$', and then only to '?', the process visits each node at most twice and is therefore guaranteed to halt.

3 Context-independent signs

Context-specific information cannot be represented explicitly in a qualitative probabilistic network, but is hidden in the network's qualitative influences. If, for example, the influence of a node A on a node B is positive for one combination of values for the set X of B's parents other than A, and zero for all other combinations of values for X, then the influence of A on B is positive by definition. The zero influences are hidden due to the fact that the inequality in the definition of qualitative influence is not strict. We present an example illustrating such hidden zeroes.

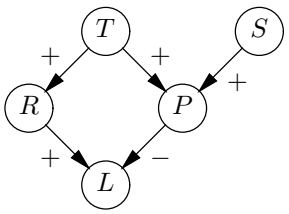


Figure 1: The qualitative *surgery* network.

Example 1 The qualitative network from Figure 1 represents a highly simplified fragment of knowledge in oncology; it pertains to the effects and complications to be expected from treatment of oesophageal cancer. Node L models the life expectancy of a patient after therapy; the value l indicates that the patient will survive for at least one year. Node T models the therapy instilled; we consider surgery, modelled by t, and no treatment, modelled by $\bar{t}$, as the only alternatives. The effect to be attained from surgery is a radical resection of the oesophageal tumour, modelled by node R. After surgery a life-threatening pulmonary complication, modelled by node P, may result; the occurrence of this complication is heavily influenced by whether or not the patient is a smoker, modelled by node S.

We consider the conditional probabilities from a quantified network representing the same knowledge. We would like to note that these probabilities serve illustrative purposes only; although not entirely unrealistic, they have not been specified by domain experts. The probability of attaining a radical resection upon surgery is $\Pr(r \mid t) = 0.45$; as without surgery there can be no radical resection, we have $\Pr(r \mid \bar{t}) = 0$.

From these probabilities we have that node T indeed exerts a positive qualitative influence on node R. The probabilities of a pulmonary complication occurring and of a patient's life expectancy after therapy are, respectively,

$\Pr(p)$	s	$\bar{s}$
t	0.75	0.00
$\bar{t}$	0.00	0.00

$\Pr(l)$	p	$\bar{p}$
r	0.15	0.95
$\bar{r}$	0.03	0.50

From the left table, we verify that both T and S exert a positive qualitative influence on node P. The fact that the influence of T on P is actually zero in the context of the value $\bar{s}$ for node S, is not apparent from the influence's sign. Note that this zero influence does not arise from the probabilities being zero, but rather from their having the same value. From the right table we verify that node R exerts a positive influence on node L; the qualitative influence of P on L is negative. □

The previous example shows that the level of representation detail of a qualitative network can result in information hiding. As a consequence, unnecessarily weak answers may result upon inference. For example, from the probabilities involved we know that performing surgery on a non-smoker has a positive influence on life expectancy. Due to the conflicting reasoning chains from T to L in the qualitative network, however, entering the observation t for node T will result in a '?' for node L, indicating that the influence is unknown.

We recall from the definition of qualitative influence that the sign of an influence of a node A on a node B is independent of the values for the set X of parents of B other than A. A '?' for the influence of A on B may therefore hide the information that node A has a positive influence on node B for some combination of values of X and a negative influence for another combination. If so, the ambiguous influence is *non-monotonic* in nature and can in fact be looked upon as specifying different signs for different contexts. We present an example to illustrate this observation.

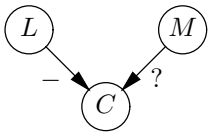


Figure 2: The qualitative *cervical metastases* network.

Example 2 The qualitative network from Figure 2 represents another fragment of knowledge in oncology; it pertains to the metastasis of oesophageal cancer. Node L represents the location of the primary tumour that is known to be present in a patient's oesophagus; the value l models that the tumour resides in the lower two-third of the oesophagus and the value $\bar{l}$ expresses that the tumour is in the oesophagus' upper one-third. An oesophageal tumour upon growth typically gives rise to lymphatic metastases, the extent of which are captured by node M. The value $\bar{m}$ of M indicates that just the local and regional lymph nodes are affected; m denotes that distant lymph nodes are affected. Which lymph nodes are local or regional and which are distant depends on the location of the tumour in the oesophagus. The lymph nodes in the neck, or cervix, for example, are regional for a tumour in the upper one-third of the oesophagus and distant otherwise. Node C represents the presence or absence of metastases in the cervical lymph nodes.

We consider the conditional probabilities from a quantified network representing the same knowledge; once again, these probabilities serve illustrative purposes only. The probabilities of the presence of cervical metastases in a patient are

$\Pr(c)$	l	$\bar{l}$
m	0.35	0.95
$\bar{m}$	0.00	1.00

From these probabilities we have that node L indeed has a negative influence on node C. The influence of node M on C, however, is non-monotonic:

$$\Pr(c \mid ml) > \Pr(c \mid \bar{m}l), \text{ yet } \Pr(c \mid m\bar{l}) < \Pr(c \mid \bar{m}\bar{l})$$

The non-monotonic influence hides a '$+$' for the value l of node L and a '$-$' for the context $\bar{l}$. □

From the two examples above, we observe that context-specific information about influences that is present in the conditional probabilities of a quantified network cannot be represented explicitly in a qualitative probabilistic network: upon abstracting the quantified network to the qualitative network, the information is effectively hidden.

4 Context-specificity and its exploitation

The level of representation detail of a qualitative probabilistic network enforces influences to be independent of specific contexts. In this section we present an extension to the basic formalism of qualitative networks that allows for associating context-specific signs with qualitative influences. In Section 4.1, the extended formalism is introduced; in Section 4.2, we show, by means of the example networks from the previous section, that exploiting context-specific information can prevent unnecessarily weak results upon inference.

4.1 Context-specific signs

Before introducing context-specific signs, we define a notion of context for qualitative networks. Let X be a set of nodes, called the *context nodes*. A *context* c_X for X is a combination of values for a subset $Y \subseteq X$ of the set of context nodes. When $Y = \varnothing$, we say that the context is *empty*, denoted ϵ; when $Y = X$, we say that the context is *maximal*. The set of all possible contexts for X is called the *context set* for X and is denoted $\mathcal{C}_X$. To compare different contexts for the same set of context nodes X, we use an ordering on contexts: for any two combinations of values c_X and c'_X for $Y \subseteq X$ and $Y' \subseteq X$, respectively, we say that $\mathrm{c}_X > \mathrm{c}'_X$ iff $Y \supset Y'$ and c_X and c'_X specify the same combination of values for Y'.

A *context-specific sign* now basically is a sign that may vary from context to context. It is defined as a function $\delta : \mathcal{C}_X \rightarrow \{+, -, 0, ?\}$ from a context set $\mathcal{C}_X$ to the set of basic signs, such that for any two contexts c_X and c'_X with $\mathrm{c}_X > \mathrm{c}'_X$ we have that, if $\delta(\mathrm{c}'_X) = \delta_i$ for $\delta_i \in \{+, -, 0\}$, then $\delta(\mathrm{c}_X) \in \{\delta_i, 0\}$. For abbreviation, we will write $\delta(X)$ to denote the context-specific sign δ that is defined on the context set $\mathcal{C}_X$. Note that the basic signs from regular qualitative networks can be looked upon as context-specific signs that are defined by a constant function.

In our extended formalism of qualitative networks, we assign context-specific signs to influences. We say that a node A exerts a *qualitative influence of sign* $\delta(X)$ on a node B, denoted $S^{\delta(X)}(A, B)$, where X is the set of parents of B other than A, iff for each context c_X for X we have that

- $\delta(\mathrm{c}_X) = +$ iff $\Pr(b \mid a\mathrm{c}_X y) \geq \Pr(b \mid \bar{a}\mathrm{c}_X y)$ for any combination of values $\mathrm{c}_X y$ for X;
- $\delta(\mathrm{c}_X) = -$ iff $\Pr(b \mid a\mathrm{c}_X y) \leq \Pr(b \mid \bar{a}\mathrm{c}_X y)$ for any such combination of values $\mathrm{c}_X y$;
- $\delta(\mathrm{c}_X) = 0$ iff $\Pr(b \mid a\mathrm{c}_X y) = \Pr(b \mid \bar{a}\mathrm{c}_X y)$ for any such combination of values $\mathrm{c}_X y$;
- $\delta(\mathrm{c}_X) = ?$ otherwise.

Note that we take the set of parents of node B other than A for the set of context nodes; the definition is readily extended to apply to arbitrary sets of context nodes, however. Context-specific qualitative synergies can be defined analogously.

A context-specific sign $\delta(X)$ in essence has to specify a basic sign from $\{+, -, 0, ?\}$ for each possible combination of values in the context set $\mathcal{C}_X$. From the definition of $\delta(X)$, however, we have that it is not necessary to explicitly indicate a basic sign for every such context. For example, consider an influence of a node A on a node B with the set of context nodes $X = \{D, E\}$. Suppose that the sign $\delta(X)$ of the influence is defined as

$$\begin{array}{c} \delta(\epsilon) = ?, \\ \delta(d) = +, \quad \delta(\bar{d}) = -, \quad \delta(e) = ?, \quad \delta(\bar{e}) = +, \\ \delta(de) = +, \quad \delta(d\bar{e}) = +, \quad \delta(\bar{d}e) = -, \quad \delta(\bar{d}\bar{e}) = 0 \end{array}$$

The function $\delta(X)$ is uniquely described by the signs of the smaller contexts whenever the larger contexts are assigned the same sign. The function is therefore fully specified by

$$\delta(\epsilon) = ?,\ \delta(d) = +,\ \delta(\bar{d}) = -,\ \delta(\bar{e}) = +,\ \delta(\bar{d}\bar{e}) = 0$$

The sign-propagation algorithm for probabilistic inference with a qualitative network, as discussed in Section 2, is easily extended to handle context-specific signs. The extended algorithm propagates and combines *basic signs* only. Before a sign is propagated over an influence, it is investigated whether or not the influence's sign is context-specific. If so, the currently valid context is determined from the available observations and the basic sign specified for this context is propagated; if none of the context nodes have been observed, then the sign specified for the empty context is propagated.

4.2 Exploiting context-specific signs

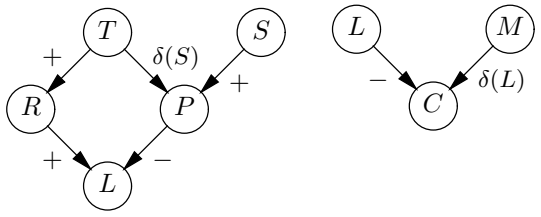
In Section 3 we presented two examples showing that the influences of a qualitative probabilistic network can hide context-specific information. Revealing this hidden information and exploiting it upon inference can be worthwhile. The information that an influence is zero for a certain context can be used, for example, to improve the runtime of the sign-propagation algorithm because propagation of a sign can be stopped as soon as a zero influence is encountered. More importantly, however, exploiting the information can prevent conflicting influences arising during inference. We illustrate this observation by means of an example.

Example 3 We reconsider the qualitative *surgery* network from Figure 1. Suppose that a non-smoker is undergoing surgery. In the context of the observation $\bar{s}$ for node S, propagating the observation t for node T with the basic sign-propagation algorithm results in the sign '?' for node L: there is not enough information present in the network to compute a non-ambiguous sign from the two conflicting reasoning chains from T to L.

We now extend the qualitative *surgery* network by assigning the context-specific sign $\delta(S)$, defined by

$$\delta(s) = +,\ \delta(\bar{s}) = 0,\ \ \delta(\epsilon) = +$$

to the influence of node T on node P, that is, we explicitly include the information that non-smoking patients are not at risk for pulmonary complications after surgery. The thus extended network is shown in Figure 3(a). We now reconsider our non-smoking patient undergoing surgery. Propagating the observation t for node T with the extended sign-propagation algorithm in the context of $\bar{s}$ results in the sign '$(+ \otimes +) \oplus (0 \otimes -)$' = '+' for node L: we find that surgery is likely to increase life expectancy for the patient. □



(a) (b)

Figure 3: A hidden zero revealed, (a), and a non-monotonicity captured, (b), by a context-specific sign.

In Section 3 we not only discussed hidden zero influences, but also argued that positive and negative influences can be hidden in non-monotonic influences. As the initial '?'s of these influences tend to spread to major parts of a network upon inference, it is worthwhile to resolve the non-monotonicities involved whenever possible. Our extended formalism of qualitative networks provides for effectively capturing information about non-monotonicities, as is demonstrated by the following example.

Example 4 We reconsider the qualitative *cervical metastases* network from Figure 2. We recall that the influence of node M on node C is non-monotonic since

$$\Pr(c \mid ml) > \Pr(c \mid \bar{m}l) \text{ and } \Pr(c \mid m\bar{l}) < \Pr(c \mid \bar{m}\bar{l})$$

In the context l, therefore, the influence is positive, while it is negative in the context $\bar{l}$. In the extended network, shown in Figure 3(b), this information is captured explicitly by assigning the sign $\delta(L)$, defined by

$$\delta(l) = +,\ \delta(\bar{l}) = -,\ \delta(\epsilon) = ?$$

to the influence of node M on node C. □

5 Context-specificity in real-life networks

To get an impression of the context-specific information that is hidden in real-life qualitative probabilistic networks, we

	# *influences with sign* δ:				
	$+$	$-$	0	?	*total*
ALARM	17	9	0	20	46
oesophagus	32	12	0	15	59

Table 2: The numbers of influences with '$+$', '$-$', '0' and '?' signs for the qualitative ALARM and oesophagus networks.

computed qualitative abstractions of the well-known ALARM-network and of the network for oesophageal cancer. The ALARM-network consists of 37, mostly non-binary, nodes and 46 arcs; the number of direct qualitative influences in the abstracted network – using the basic definition of qualitative influence – therefore equals 46. The oesophagus network consists of 42, also mostly non-binary, nodes and 59 arcs. Table 2 summarises for the two abstracted networks the numbers of direct influences with the four different basic signs.

The numbers reported in Table 2 pertain to the basic signs of the qualitative influences associated with the arcs in the networks' digraphs. Each such influence, and hence each associated basic sign, covers a number of maximal contexts. For a qualitative influence associated with the arc $A \rightarrow B$, the number of maximal contexts equals 1 (the empty context) if node B has no other parents than A; otherwise, the number of maximal contexts equals the number of possible combinations of values for the set of parents of B other than A. For every maximal context, we computed the proper (context-specific) sign from the original quantified network. Table 3 summarises the number of context-specific signs covered by the different basic signs in the two abstracted networks. From the table we have, for example, that the 17 qualitative influences with sign '$+$' from the ALARM network together cover 59 different maximal contexts. For 38 of these contexts, the influences are indeed positive, but for 21 of them the influences are actually zero.

		# c_X *with sign* δ':				
ALARM		$+$	$-$	0	?	*total*
	$+$	38	–	21	–	59
δ:	$-$	–	40	11	–	51
	0	–	–	–	–	0
	?	34	24	12	28	108
	total	72	64	44	28	218
		# c_X *with sign* δ':				
oesophagus		$+$	$-$	0	?	*total*
	$+$	74	–	8	–	82
δ:	$-$	–	36	8	–	44
	0	–	–	–	–	0
	?	6	3	2	38	49
	total	80	39	18	38	175

Table 3: The numbers of contexts c_X covered by the '$+$', '$-$', '0' and '?' signs and their associated context-specific signs, for the qualitative ALARM and oesophagus networks.

For the qualitative ALARM-network, we find that 35% of the influences are positive, 17% are negative, and 48% are ambiguous; the network does not include any explicitly specified zero influences. For the extended network, using context-specific signs, we find that 32% of the qualitative influences are positive, 31% are negative, 20% are zero, and 17% remain ambiguous. For the qualitative oesophagus network, we find that 54% of the influences are positive, 21% are negative, and 25% are ambiguous; the network does not include any explicit zero influences. For the extended network, using context-specific signs, we find that 46% of the qualitative influences are positive, 22% are negative, 10% are zero, and 22% remain ambiguous.

We observe that for both the ALARM and the oesophagus network, the use of context-specific signs serves to reveal a considerable number of zero influences and to substantially decrease the number of ambiguous influences. Similar observations were made for qualitative abstractions of two other real-life probabilistic networks, pertaining to Wilson's disease and to ventricular septal defect, respectively. We conclude that by providing for the inclusion of context-specific information about influences, we have effectively extended the expressive power of qualitative probabilistic networks.

6 Extension to enhanced networks

The formalism of *enhanced qualitative probabilistic networks* [Renooij & Van der Gaag, 1999], introduces a qualitative notion of strength of influences into qualitative networks. We briefly argue that the notions from the previous sections can also be used to provide for the inclusion and exploitation of context-specific information about such strengths.

In an enhanced qualitative network, a distinction is made between strong and weak influences by partitioning the set of all influences into two disjoint subsets in such a way that any influence from the one subset is stronger than any influence from the other subset; to this end a *cut-off value* α is used. For example, a *strongly positive qualitative influence* of a node A on a node B, denoted $S^{++}(A, B)$, expresses that

$$\Pr(b \mid ax) - \Pr(b \mid \bar{a}x) \geq \alpha$$

for any combination of values x for the set X of parents of B other than A; a *weakly positive qualitative influence* of A on B, denoted $S^{+}(A, B)$, expresses that

$$0 \leq \Pr(b \mid ax) - \Pr(b \mid \bar{a}x) \leq \alpha$$

for any such combination of values x. The sign '$+_?$' is used to indicate a positive influence whose relative strength is ambiguous. Strongly negative qualitative influences S^{--}, and weakly negative qualitative influences S^{-}, are defined analogously; a negative influence whose relative strength is ambiguous is denoted $S^{-?}$. Zero qualitative influences and ambiguous qualitative influences are defined as in regular qualitative probabilistic networks. Renooij & Van der Gaag (1999) also provide extended definitions for the $\oplus$- and $\otimes$-operators to apply to the double signs. These definitions cannot be reviewed without detailing the enhanced formalism, which is beyond the scope of the present paper; it suffices to say that the result of combining signs is basically as one would intuitively expect.

Our notion of context-specific sign can be easily incorporated into enhanced qualitative probabilistic networks. A context-specific sign now is defined as a function $\delta : \mathcal{C}_X \rightarrow \{++, +_?, +, -, -_?, --, 0, ?\}$ from a context set $\mathcal{C}_X$ to the

extended set of basic signs, such that for any two contexts c_X and c'_X with $c_X > c'_X$ we have that, if the sign is strongly positive for c'_X, then it must be strongly positive for c_X, if the sign is weakly positive for c'_X, then it must be either weakly positive or zero for c_X, and if it is ambiguously positive for c'_X, then it may be (strongly, weakly or ambiguously) positive, or zero for c_X. Similar restrictions hold for negative signs. Context-specific signs are once again assigned to influences, as before.

For distinguishing between strong and weak qualitative influences in an enhanced network, a cut-off value α has to be chosen in such a way that, basically, for *all* strong influences of a node A on a node B we have that $|\Pr(b \mid ax) - \Pr(b \mid \bar{a}x)| \geq \alpha$ for *all* contexts x, and for *all* weak influences we have that $|\Pr(b \mid ax) - \Pr(b \mid \bar{a}x)| \leq \alpha$ for *all* such contexts. If, for a specific cut-off value α, there exists an influence of node A on node B for which there are contexts x and x' with $|\Pr(b \mid ax) - \Pr(b \mid \bar{a}x)| > \alpha$ and $|\Pr(b \mid ax') - \Pr(b \mid \bar{a}x')| < \alpha$, then signs of ambiguous strength would be introduced into the enhanced network, which would seriously hamper the usefulness of exploiting a notion of strength. A different cut-off value had better be chosen, by shifting α towards 0 or 1. Unfortunately, α may then very well end up being 0 or 1. The use of context-specific information about qualitative strengths can now forestall the necessity of shifting the cut-off value, as is illustrated in the following example.

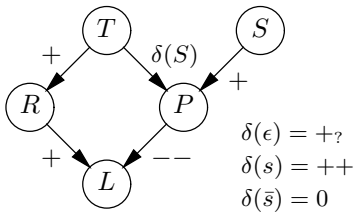


Figure 4: Context-specific sign in an enhanced network.

Example 5 We reconsider the *surgery* network and its associated probabilities from Example 1. Upon abstracting the network to an enhanced qualitative network, we distinguish between strong and weak influences by choosing a cut-off value of, for example, $\alpha = 0.46$. We then have that a pulmonary complication after surgery strongly influences life expectancy, that is, $S^{--}(P, L)$. For this cut-off value, however, the influence of node T on node P is neither strongly positive nor weakly positive; the value $\alpha = 0.46$ therefore does not serve to partition the set of influences in two distinct subsets. To ensure that all influences in the network are either strong or weak, the cut-off value should be either 0 or 1.

For the influence of node T on node P, we observe that, for $\alpha = 0.46$, the influence is strongly positive for the value s of node S and zero for the context $\bar{s}$. By assigning the context-specific sign $\delta(S)$ defined by

$$\delta(s) = ++,\ \delta(\bar{s}) = 0,\ \delta(\epsilon) = +_?$$

to the influence of node T on node P, we explicitly specify the otherwise hidden strong and zero influences. The thus extended network is shown in Figure 4. We recall from Example 3 that for non-smokers the effect of surgery on life expectancy is positive. For smokers, however, the effect could not be unambiguously determined. From the extended network in Figure 4, we now find the effect of surgery on life expectancy for smokers to be negative: upon propagating the observation t for node T in the context of the information s for node S, the sign '$(+ \otimes +) \oplus (+ + \otimes - -)$' = '$-$' results for node L. □

7 Conclusions

We extended the formalism of qualitative probabilistic networks with a notion of context-specificity. By doing so, we enhanced the expressive power of qualitative networks. While in a regular qualitative network, zero influences as well as positive and negative influences can be hidden, in a network extended with context-specific signs this information is made explicit. Qualitative abstractions of some real-life probabilistic networks have shown that networks indeed can incorporate considerable context-specific information. We further showed that incorporating the context-specific signs into enhanced qualitative probabilistic networks that include a qualitative notion of strength renders even more expressive power. The fact that zeroes and double signs can be specified context-specifically allows them to be specified more often, in general. We showed that exploiting context-specific information about influences and about qualitative strengths can prevent unnecessary ambiguous node signs arising during inference, thereby effectively forestalling unnecessarily weak results.

References

[Boutilier *et al.*, 1996] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, 1996, pp. 115 – 123.

[Druzdzel & Henrion, 1993] M.J. Druzdzel and M. Henrion. Efficient reasoning in qualitative probabilistic networks. *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 1993, pp. 548 – 553.

[Druzdzel & Van der Gaag, 1995] M.J. Druzdzel and L.C. van der Gaag. Elicitation of probabilities for belief networks: combining qualitative and quantitative information. *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 1995, pp. 141 – 148.

[Renooij & Van der Gaag, 1999] S. Renooij and L.C. van der Gaag. Enhancing QPNs for trade-off resolution. *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 1999, pp. 559 – 566.

[Wellman, 1990] M.P. Wellman. Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, vol. 44, 1990, pp. 257 – 303.

[Zhang & Poole, 1999] N.L. Zhang and D. Poole. On the role of context-specific independence in probabilistic inference. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999, pp. 1288 – 1293.

Max-norm Projections for Factored MDPs

Carlos Guestrin
Computer Science Dept.
Stanford University
guestrin@cs.stanford.edu

Daphne Koller
Computer Science Dept.
Stanford University
koller@cs.stanford.edu

Ronald Parr
Computer Science Dept.
Duke University
parr@cs.duke.edu

Abstract

Markov Decision Processes (MDPs) provide a coherent mathematical framework for planning under uncertainty. However, exact MDP solution algorithms require the manipulation of a *value function*, which specifies a value for each state in the system. Most real-world MDPs are too large for such a representation to be feasible, preventing the use of exact MDP algorithms. Various approximate solution algorithms have been proposed, many of which use a linear combination of basis functions as a compact approximation to the value function. Almost all of these algorithms use an approximation based on the (weighted) $\mathcal{L}_2$-norm (Euclidean distance); this approach prevents the application of standard convergence results for MDP algorithms, all of which are based on max-norm. This paper makes two contributions. First, it presents the first approximate MDP solution algorithms — both value and policy iteration — that use max-norm projection, thereby directly optimizing the quantity required to obtain the best error bounds. Second, it shows how these algorithms can be applied efficiently in the context of *factored MDPs*, where the transition model is specified using a dynamic Bayesian network.

1 Introduction

Over the last few years, *Markov Decision Processes (MDPs)* have been used as the basic semantics for optimal planning for decision theoretic agents in stochastic environments. In the MDP framework, the system is modeled via a set of states which evolve stochastically. The key problem with this representation is that, in virtually any real-life domain, the state space is quite large. However, many large MDPs have significant internal structure, and can be modeled compactly if the structure is exploited in the representation.

Factored MDPs [Boutilier *et al.*, 1999] are one approach to representing large, structured MDPs compactly. In this framework, a state is implicitly described by an assignment to some set of *state variables*. A *dynamic Bayesian network (DBN)* [Dean and Kanazawa, 1989] can then allow a compact representation of the transition model, by exploiting the fact that the transition of a variable often depends only on a small number of other variables. Furthermore, the momentary rewards can often also be decomposed as a sum of rewards related to individual variables or small clusters of variables.

Even when a large MDP can be represented compactly, e.g., in a factored way, solving it exactly is still intractable: Exact MDP solution algorithms require the manipulation of a value function, whose representation is linear in the number of states, which is exponential in the number of state variables. One approach is to approximate the solution using an approximate value function with a compact representation. A common choice is the use of *linear* value functions as an approximation — value functions that are a linear combination of basis functions.

This paper makes a twofold contribution. First, we provide a new approach for approximately solving MDPs using a linear value function. Previous approaches to linear function approximation typically have utilized a least squares ($\mathcal{L}_2$-norm) approximation to the value function. Least squares approximations are incompatible with most convergence analyses for MDPs, which are based on max-norm. We provide the first MDP solution algorithms — both value iteration and policy iteration — that use a linear max-norm projection to approximate the value function, thereby directly optimizing the quantity required to obtain the best error bounds.

Second, we show how to exploit the structure of the problem in order to apply this technique to factored MDPs. Our work builds on the ideas of Koller and Parr [1999; 2000], by using *factored (linear) value functions*, where each basis function is restricted to some small subset of the domain variables. We show that, for a factored MDP and factored value functions, various key operations can be implemented in closed form without enumerating the entire state space. Thus, our max-norm algorithms can be implemented efficiently, even though the size of the state space grows exponentially in the number of variables.

2 Markov decision processes

A *Markov Decision Process (MDP)* is defined as a 4-tuple (S, A, R, P) where: S is a finite set of $|S|$ states; A is a set of actions; R is a *reward function* $R : S \times A \mapsto \mathbb{R}$, such that $R(s, a)$ represents the reward obtained by the agent in state s after taking action a; and P is a *Markovian transition model* where $P(s' \mid s, a)$ represents the probability of going from state s to state s' with action a.

We will be assuming that the MDP has an infinite horizon and that future rewards are discounted exponentially with a discount factor $\gamma \in [0, 1)$. A stationary policy π for an MDP is a mapping $\pi : S \mapsto A$, where $\pi(s)$ is the action the agent takes at state s. The policy is associated with a *value function* $\mathcal{V}_\pi \in \mathbb{R}^{|S|}$, where $\mathcal{V}_\pi(s)$ is the discounted cumulative value that the agent gets if it starts at state s. The value function for a fixed policy is the fixed point of a set of equations that define the value of a state in terms of the value of its possible successor states. More formally, we define:

Definition 2.1 *The* DP operator, $\mathcal{T}_\pi$, *for a fixed stationary policy* π *is:*

$$\mathcal{T}_\pi \mathcal{V}(s) = R(s, \pi(s)) + \gamma \Sigma_{s'} P(s'|s, \pi(s)) \mathcal{V}(s').$$

$\mathcal{V}_\pi$ *is the fixed point of* $\mathcal{T}_\pi$*:* $\mathcal{V}_\pi = \mathcal{T}_\pi \mathcal{V}_\pi$. ∎

The optimal value function $\mathcal{V}^*$ is also defined by a set of equations. In this case, the value of a state must be the maximal value achievable by any action at that state. More precisely, we define:

Definition 2.2 *The* Bellman operator, $\mathcal{T}^*$, *is:*

$$\mathcal{T}^*\mathcal{V}(s) = \max_a [R(s,a) + \gamma \Sigma_{s'} P(s' \mid s,a)\mathcal{V}(s')].$$

$\mathcal{V}^*$ *is the fixed point of* $\mathcal{T}^*$*:* $\mathcal{V}^* = \mathcal{T}^*\mathcal{V}^*$. ∎

For any value function $\mathcal{V}$, we can define the policy obtained by acting greedily relative to $\mathcal{V}$. In other words, at each state, we take the action that maximizes the one-step utility, assuming that $\mathcal{V}$ represents our long-term utility achieved at the next state. More precisely, we define

$$\textit{Greedy}(\mathcal{V})(s) = \arg\max_a [R(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\mathcal{V}(s')].$$

The greedy policy relative to the optimal value function $\mathcal{V}^*$ is the optimal policy $\pi^* = \textit{Greedy}(\mathcal{V}^*)$. There are several algorithms to compute the optimal policy, we will focus on the two most used: value iteration and policy iteration.

Value iteration relies on the fact that the Bellman operator is a *contraction* — it is guaranteed to reduce the *max-norm* ($\mathcal{L}_\infty$) distance between any pair of value functions by a factor of at least γ. This property guarantees that the Bellman operator has a unique fixed point $\mathcal{V}^*$ [Puterman, 1994]. Value iteration exploits this property, approaching the fixed point through successive applications of the Bellman operator: $\mathcal{V}^{(t+1)} = \mathcal{T}^*\mathcal{V}^{(t)}$. After a finite number of iterations, the greedy policy $\textit{Greedy}(\mathcal{V}^{(t)})$ will be the optimal policy.

Policy iteration iterates over policies. Each iteration consists of two phases. *Value determination* computes, for a policy $\pi^{(t)}$, the value function $\mathcal{V}_{\pi^{(t)}}$, by finding the fixed point of: $\mathcal{T}_{\pi^{(t)}}\mathcal{V}_{\pi^{(t)}} = \mathcal{V}_{\pi^{(t)}}$. *Policy improvement* defines the next policy as $\pi^{(t+1)}(s) = \textit{Greedy}(\mathcal{V}_{\pi^{(t)}})$. It can be shown that this process converges to the optimal policy.

3 Solving MDPs with max-norm projections

In many domains, the state space is very large, and we need to perform our computations using approximate value functions. A very popular choice is to approximate a value function using *linear regression*. Here, we define our space of allowable value functions $\mathcal{V} \in \mathcal{H} \subseteq \mathbb{R}^{|S|}$ via a set of *basis functions* $H = \{h_1, \ldots, h_k\}$. A *linear value function* over H is a function $\mathcal{V}$ that can be written as $\mathcal{V}(s) = \sum_{j=1}^k w_j h_j(s)$ for some coefficients $\mathbf{w} = (w_1, \ldots, w_k)'$. We define $\mathcal{H}$ to be the linear subspace of $\mathbb{R}^{|S|}$ spanned by the basis functions H. It is useful to define an $|S| \times k$ matrix A whose columns are the k basis functions, viewed as vectors. Our approximate value function is then represented by $A\mathbf{w}$.

Linear value functions: The idea of using linear value functions for dynamic programming was proposed, initially, by Bellman et al. [1963] and has been further explored recently [Tsitsiklis and Van Roy, 1996; Koller and Parr, 1999; 2000]. The basic idea is as follows: in the solution algorithms, whether value or policy iteration, we use only value functions within $\mathcal{H}$. Whenever the algorithm takes a step that results in a value function $\mathcal{V}$ that is outside this space, we *project* the result back into the space by finding the value function within the space which is close to $\mathcal{V}$. More precisely:

Definition 3.1 *A* projection operator Π *is a mapping* $\Pi : \mathbb{R}^{|S|} \rightarrow \mathcal{H}$. Π *is said to be a* projection w.r.t. a norm $\|\cdot\|$ *if:* $\Pi\mathcal{V} = A\mathbf{w}^*$ *such that* $\mathbf{w}^* \in \arg\min_{\mathbf{w}} \|A\mathbf{w} - \mathcal{V}\|$. ∎

Unfortunately, these existing algorithms all suffer from a problem that we might call "norm incompatibility." When computing the projection, they all utilize the standard projection operator with respect to $\mathcal{L}_2$ norm or a *weighted* $\mathcal{L}_2$ norm. On the other hand, most of the convergence and error analyses for MDP algorithms utilize max-norm. This incompatibility has made it difficult to provide error guarantees.

In this section, we propose a new approach that addresses the issue of norm compatibility. Our key idea is the use of a projection operator in $\mathcal{L}_\infty$ norm. This problem has been studied in the optimization literature as the problem of finding the Chebyshev solution to an overdetermined linear system of equations [Cheney, 1982]. The problem is defined as finding $\mathbf{w}^*$ such that:

$$\mathbf{w}^* \in \arg\min_{\mathbf{w}} \|C\mathbf{w} - \mathbf{b}\|_\infty . \tag{1}$$

We will use an algorithm due to Stiefel [1960], that solves this problem by linear programming:

$$\begin{array}{lll} \text{Variables:} & w_1, \ldots, w_k, \phi\ ; & \\ \text{Minimize:} & \phi\ ; & \\ \text{Subject to:} & \phi \geq \sum_{j=1}^k c_{ij} w_j - b_i & \text{and} \\ & \phi \geq b_i - \sum_{j=1}^k c_{ij} w_j\,, & i = 1 \ldots |S|\,. \end{array} \tag{2}$$

For the solution $(\mathbf{w}^*, \phi^*)$ of this linear program, $\mathbf{w}^*$ is the solution of Eq. (1) and ϕ is the $\mathcal{L}_\infty$ projection error. Note that this LP only has $k+1$ variables. However, there are $2 \cdot |S|$ constraints, which makes it impractical for large state spaces. In the remainder of this section, we will discuss how this projection addresses the norm incompatibility problem. In Section 4, we will show that, in factored MDPs, all the $2 \cdot |S|$ constraints can be represented efficiently, leading to a tractable algorithm.

Approximate Value iteration: The basic idea of approximate value iteration is quite simple. We define an $\mathcal{L}_\infty$ projection operator Π_∞ that takes a value function $\mathcal{V}$ and finds $\mathbf{w}$ that minimizes $\|A\mathbf{w} - \mathcal{V}\|_\infty$. This is an instance of Eq. (1) and can be solved by Eq. (2). The algorithm alternates applications of the Bellman operator $\mathcal{T}^*$ and projection steps Π_∞:

$$\overline{\mathcal{V}}^{(t+1)} = \mathcal{T}^* A\mathbf{w}^{(t)}; \quad \text{and} \quad A\mathbf{w}^{(t+1)} = \Pi_\infty \overline{\mathcal{V}}^{(t+1)}.$$

In standard value iteration, we only need to perform the first step. However, $\overline{\mathcal{V}}^{(t+1)}$ may not be in $\mathcal{H}$, so we need to add the second step, the projection step, to the process. We can analyze this process, bounding the overall error between our approximate value function $A\mathbf{w}^{(t)}$ and the optimal value function $\mathcal{V}^*$. This analysis shows that the overall error depends on the single-step max-norm projection errors $\phi^{(t+1)} = \left\| A\mathbf{w}^{(t+1)} - \overline{\mathcal{V}}^{(t+1)} \right\|_\infty$. Thus, using the max-norm projection, we can minimize these projection errors directly, we omit the analysis for lack of space. Note that, in approximate value iteration, the error introduced by successive approximations may grow unboundedly. As we will show, this cannot happen in approximate policy iteration.

Approximate Policy iteration: As we discussed, policy iteration is composed of two steps: value determination and

policy improvement. Our algorithm performs the policy improvement step exactly. In the value determination step, the value function is approximated through a linear combination of basis functions. Consider the value determination for a policy $\pi^{(t)}$. Define $R_{\pi^{(t)}}(s) = R(s, \pi^{(t)}(s))$, and $P_{\pi^{(t)}}(s' \mid s) = P(s' \mid s, a = \pi^{(t)}(s))$. We can now rewrite the value determination step in terms of matrices and vectors. If we view $\mathcal{V}_{\pi^{(t)}}$ and $R_{\pi^{(t)}}$ as $|S|$-vectors, and $P_{\pi^{(t)}}$ as an $|S| \times |S|$ matrix, we have the equations: $\mathcal{V}_{\pi^{(t)}} = R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} \mathcal{V}_{\pi^{(t)}}$. This is a system of linear equations with one equation for each state, which can only be solved exactly for small $|S|$. Our goal is to provide an approximate solution, within $\mathcal{H}$. More precisely, we want to find:

$$\begin{aligned} \mathbf{w}^{(t)} &= \arg\min_{\mathbf{w}} \left\| A\mathbf{w} - (R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} A\mathbf{w}) \right\|_\infty ; \\ &= \arg\min_{\mathbf{w}} \left\| (A - \gamma P_{\pi^{(t)}} A)\, \mathbf{w}^{(t)} - R_{\pi^{(t)}} \right\|_\infty . \end{aligned}$$

This minimization is another instance of an $\mathcal{L}_\infty$ projection (Eq. (1)), where $C = (A - \gamma P_{\pi^{(t)}} A)$ and $\mathbf{b} = R_{\pi^{(t)}}$, and can be solved using the linear program of Eq. (2). Thus, our approximate policy iteration alternates between two steps:

$$\begin{aligned} \mathbf{w}^{(t)} &= \arg\min_{\mathbf{w}} \left\| A\mathbf{w} - (R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} A\mathbf{w}) \right\|_\infty ; \\ \pi^{(t+1)} &= \textit{Greedy}(A\mathbf{w}^{(t)}). \end{aligned}$$

For the analysis, we define the projection error for the policy iteration case, i.e., the error resulting from the approximate value determination step: $\beta^{(t)} = \left\| A\mathbf{w}^{(t)} - \left(R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} A\mathbf{w}^{(t)}\right) \right\|_\infty$. Unlike in approximate value iteration, the error is bounded in this case:

Lemma 3.2 *There exists a constant $\beta_P < \infty$ such that $\beta_P \geq \beta^{(t)}$ for all iterations t of the algorithm.* ∎

Finally, we define *discounted accumulated projection error* as $\overline{\beta}^{(t)} = \beta^{(t)} + \gamma \overline{\beta}^{(t-1)}$; $\overline{\beta}^{(0)} = 0$. Lemma 3.2 implies that the accumulated error remains bounded: $\overline{\beta}^{(t)} \leq \frac{\beta_P (1-\gamma^t)}{1-\gamma}$.

We can now bound the error in the value function resulting from approximate policy iteration:

Theorem 3.3 *In the approximate policy iteration algorithm, the distance between our approximate value function at iteration t and the optimal value function is bounded by:*

$$\left\| A\mathbf{w}^{(t)} - \mathcal{V}^* \right\|_\infty \leq \gamma^t \left\| A\mathbf{w}^{(0)} - \mathcal{V}^* \right\|_\infty + \frac{2\gamma \overline{\beta}^{(t)}}{(1-\gamma)^2}.$$ ∎

In words, the difference between our approximation at iteration t and the optimal value function is bounded by the sum of two terms. The first term is present in standard policy iteration and goes to zero exponentially fast. The second is the discounted accumulated projection error and, as the theorem shows, is bounded. This second term can be minimized by choosing $\mathbf{w}^{(t)}$ as the one that minimizes $\left\| A\mathbf{w}^{(t)} - \left(R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} A\mathbf{w}^{(t)}\right) \right\|_\infty$. Therefore, by performing max-norm projections, we can make the bound on the theorem as tight as possible.

4 Solving factored MDPs

4.1 Factored MDPs

Our presentation of factored MDPs follows that of [Koller and Parr, 2000]. In a factored MDP, the set of states is described via a set of random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, where each X_i takes on values in some finite domain $\mathrm{Dom}(X_i)$. A state $\mathbf{x}$ defines a value $x_i \in \mathrm{Dom}(X_i)$ for each variable X_i. We define a state transition model τ using a *dynamic Bayesian network (DBN)* [Dean and Kanazawa, 1989]. Let X_i denote the variable X_i at the current time and X_i' the variable at the next step. The *transition graph* of a DBN is a two-layer directed acyclic graph G_τ whose nodes are $\{X_1, \ldots, X_n, X_1', \ldots, X_n'\}$. We denote the parents of X_i' in the graph by *Parents*$_\tau(X_i')$. For simplicity of exposition, we assume that *Parents*$_\tau(X_i') \subseteq \mathbf{X}$; i.e., all arcs in the DBN are between variables in consecutive time slices. (This assumption can be relaxed, but our algorithm becomes somewhat more complex.) Each node X_i' is associated with a *conditional probability distribution (CPD)* $P_\tau(X_i' \mid$ *Parents*$_\tau(X_i'))$. The transition probability $P_\tau(\mathbf{x}' \mid \mathbf{x})$ is then defined to be $\prod_i P_\tau(x_i' \mid \mathbf{u}_i)$, where $\mathbf{u}_i$ is the value in $\mathbf{x}$ of the variables in *Parents*$_\tau(X_i')$.

Consider, for example, the problem of optimizing the behavior of a system administrator maintaining a network of n computers. Each machine is connected to some subset of other machines. In one simple network, we might connect the machines in a ring, with machine i connected to machines $i+1$ and $i-1$. (In this example, we assume addition and subtraction are performed modulo n). Each machine is associated with a binary random variable F_i, representing whether it has failed. The parents of F_i' are F_i, F_{i-1}, F_{i+1}. The CPD of F_i' is such that if $F_i =$ *true*, then $F_i' =$ *true* with high probability; i.e., failures tend to persist. If $F_i =$ *false*, then F_i' is a *noisy or* of its two other parents; i.e., a failure in either of its neighbors can independently cause machine i to fail.

We can define the transition dynamics of an MDP by defining a separate DBN model $\tau_a = \langle G_a, P_a \rangle$ for each action a. However, in many cases, different actions have very similar transition dynamics, only differing in their effect on some small set of variables. In particular, in many cases a variable has a default evolution model, which only changes if an action affects it directly [Boutilier *et al.*, 1999]. Therefore, as in [Koller and Parr, 2000], we use the notion of a *default transition model* $\tau_d = \langle G_d, P_d \rangle$. For each action a, we define $\mathit{Effects}[a] \subseteq \mathbf{X}'$ to be the variables in the next state whose local probability model is different from τ_d, i.e., those variables X_i' such that $P_a(X_i' \mid$ *Parents*$_a(X_i')) \neq P_d(X_i' \mid$ *Parents*$_d(X_i'))$.

In our system administrator example, we have an action a_i for rebooting each one of the machines, and a default action d for doing nothing. The transition model described above corresponds to the "do nothing" action, which is also the default transition model. The transition model for a_i is different from d only in the transition model for the variable F_i', which is now $F_i' =$ *true* with some small fixed probability, regardless of the status of the neighboring machines.

Finally, we need to provide a compact representation of the reward function. We assume that the reward function is factored additively into a set of localized reward functions, each of which only depends on a small set of variables.

Definition 4.1 *A function f is* restricted *to a domain $\mathbf{C} \subseteq \mathbf{X}$ if $f : \mathrm{Dom}(\mathbf{C}) \mapsto \mathbb{R}$. If f is restricted to $\mathbf{Y}$ and $\mathbf{Y} \subset \mathbf{Z}$, we will use $f(\mathbf{z})$ as shorthand for $f(\mathbf{y})$ where $\mathbf{y}$ is the part of the instantiation $\mathbf{z}$ that corresponds to variables in $\mathbf{Y}$.*

Let $R_1, \ldots, R_r$ be a set of functions, where each R_i is restricted to variable cluster $\mathbf{W}_i \subset \{X_1, \ldots, X_n\}$. The reward function for state $\mathbf{x}$ is defined to be $\sum_{i=1}^{r} R_i(\mathbf{x}) \in \mathbb{R}$. In our example, we might have a reward function associated with each machine i, which depends on F_i.

Factorization may allow us to represent large MDPs very compactly. However, we must still address the problem of solving these MDPs. Solution algorithms rely on the ability to represent value functions and policies, the representations which requires the same number of parameters as the size of the space. One might be tempted to believe that factored transition dynamics and rewards would result in a factored value function, which can thereby be represented compactly. Unfortunately, even in factored MDPs, the value function rarely has any internal structure [Koller and Parr, 1999].

Koller and Parr [1999] suggest that there are many domains where our value function might be "close" to structured, i.e., well-approximated using a linear combination of functions each of which refers only to a small number of variables. More precisely, they define a value function to be a *factored (linear) value function* if it is a linear function over the basis $h_1, \ldots, h_k$, where each h_i is restricted to some subset of variables $\mathbf{C}_i$. In our example, we might have basis functions whose domains are pairs of neighboring machines, e.g., one basis function which is an indicator function for each of the four combinations of values for the pair of failure variables.

As shown by Koller and Parr [1999; 2000], factored value functions provide the key to performing efficient computations over the exponential-sized state sets that we have in factored MDPs. The key insight is that restricted domain functions (including our basis functions) allow certain basic operations to be implemented very efficiently. In the remainder of this section, we will show that this key insight also applies in the context of our algorithm.

4.2 Factored Max-norm Projection

The key computational step in both of our algorithms is the solution of Eq. (1) using the linear program in Eq. (2). In our setting, the vectors $\mathbf{b}$ and $C\mathbf{w}$ are vectors in $\mathbb{R}^{|S|}$, where S is our state space. In the case of factored MDPs, our state space is a set of vectors $\mathbf{x}$ which are assignments to the state variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. We can view both $C\mathbf{w}$ and $\mathbf{b}$ as functions of these state variables, and hence also their difference. Thus, we can define a function $F^{\mathbf{w}}(X_1, \ldots, X_n)$ such that $F^{\mathbf{w}}(\mathbf{x}_i) = ((C\mathbf{w})_i - b_i)$. Note that we have executed a representation shift; we are viewing $F^{\mathbf{w}}$ as a function of $\mathbf{X}$, which is parameterized by $\mathbf{w}$.

The size of the state space is exponential in the number of variables. Hence, our goal in this section is to optimize Eq. (1) without explicitly considering each of the exponentially many states. The key is to use the fact that $F^{\mathbf{w}}$ has a factored representation. More precisely, $C\mathbf{w}$ has the form $\sum_j w_j f'_j(\mathbf{Z}_j)$, where $\mathbf{Z}_j$ is a subset of $\mathbf{X}$. For example, we might have $f'_1(X_1, X_2)$ which takes value 1 in states where $X_1 = \textit{true}$ and $X_2 = \textit{false}$ and 0 otherwise. Similarly, the vector $\mathbf{b}$ in our case is also a sum of restricted domain functions. Thus, we can express $F^{\mathbf{w}}$ as a sum $\sum_j f_j^{\mathbf{w}}(\mathbf{Z}_j)$, where $f_j^{\mathbf{w}}$ may or may not depend on $\mathbf{w}$. In the future, we sometimes drop the superscript $\mathbf{w}$ when it is clear from context.

We tackle the problem of a factored solution to the LP in Eq. (2) in two steps.

Maximizing over the state space: First, assume that $\mathbf{b}$ and $C\mathbf{w}$ are given, and that our goal is simply to compute $\max_i((C\mathbf{w})_i - b_i) = \max_{\mathbf{x}} F(\mathbf{x})$, i.e., to find the state $\mathbf{x}$ over which F is maximized. Recall that $F = \sum_{j=1}^{m} f_j(\mathbf{Z}_j)$. We can maximize such a function F using *non-serial dynamic programming* [Bertele and Brioschi, 1972] or *cost networks* [Dechter, 1999]. The idea is virtually identical to variable elimination in a Bayesian network. We review this construction here, as it is a key component in our solution LP.

Our goal is to compute

$$\max_{x_1,\ldots,x_n} \sum_j f_j(\mathbf{Z}_j[\mathbf{x}]),$$

where $\mathbf{Z}_j[\mathbf{x}]$ is the instantiation of the variables in $\mathbf{Z}_j$ in the assignment $\mathbf{x}$. The key idea is, rather than summing all functions and then doing the maximization, we maximize over variables one at a time. When maximizing over x_l, only summands involving x_l participate in the maximization. For example, assume

$$F = f_1(x_1, x_2) + f_2(x_1, x_3) + f_3(x_2, x_4) + f_4(x_3, x_4).$$

We therefore wish to compute:

$$\max_{x_1,x_2,x_3,x_4} f_1(x_1, x_2) + f_2(x_1, x_3) + f_3(x_2, x_4) + f_4(x_3, x_4).$$

We can first compute the maximum over x_4; the functions f_1 and f_2 are irrelevant, so we can push them out. We get

$$\max_{x_1,x_2,x_3} f_1(x_1, x_2) + f_2(x_1, x_3) + \max_{x_4}[f_3(x_2, x_4) + f_4(x_3, x_4)].$$

The result of the internal maximization depends on the values of x_2, x_3; i.e., we can introduce a new function $e_1(X_2, X_3)$ whose value at the point x_2, x_3 is the value of the internal $\max$ expression. Our problem now reduces to computing

$$\max_{x_1,x_2,x_3} f_1(x_1, x_2) + f_2(x_1, x_3) + e_1(x_2, x_3),$$

having one fewer variable. Next, we eliminate another variable, say X_3, with the resulting expression reducing to:

$$\max_{x_1,x_2} f_1(x_1, x_2) + e_2(x_1, x_2),$$

$$\text{where} \quad e_2(x_1, x_2) = \max_{x_3}[f_2(x_1, x_3) + e_1(x_2, x_3)].$$

Finally, we define

$$e_3 = \max_{x_1,x_2} f_1(x_1, x_2) + e_2(x_1, x_2)$$

The result at this point is a number, which is the desired maximum over $x_1, \ldots, x_4$.

In general, the variable elimination algorithm maintains a set $\mathcal{F}$ of functions, which initially $\{f_1, \ldots, f_m\}$. The algorithm then repeats the following steps:

1. Select an uneliminated variable X_l;
2. Take all $e_1, \ldots, e_L \in \mathcal{F}$ whose domain contains X_l.
3. Define a new function $e = \max_{x_l} \sum_j e_j$ and introduce it into $\mathcal{F}$. The domain of e is $\cup_{j=1}^{L} \mathrm{Dom}[e_j] - \{X_l\}$.

The computational cost of this algorithm is linear in the number of new "function values" introduced in the elimination process. More precisely, consider the computation of a new function e whose domain is $\mathbf{Z}$. To compute this function, we need to compute $|\mathrm{Dom}[\mathbf{Z}]|$ different values. The cost of the algorithm is linear in the overall number of these values, introduced throughout the algorithm. As shown in [Dechter, 1999], this cost is exponential in the induced width of the undirected graph defined over the variables $X_1, \ldots, X_n$, with an edge between X_l and X_m if they appear together in one of the original functions f_j.

Factored LP: Now, consider our original problem of minimizing $\|C\mathbf{w} - \mathbf{b}\|_\infty = \max_{\mathbf{x}} |F^{\mathbf{w}}(\mathbf{x})|$ over $\mathbf{w}$. As in Eq. (2), we want to construct a linear program which performs this optimization. However, we want a compact LP, that avoids an explicit enumeration of the constraints for the exponentially many states. The first key insight is that we can replace the entire set of constraints — $\phi \geq (C\mathbf{w})_i - b_i$ for all states i — by the equivalent constraint $\phi \geq \max_i((C\mathbf{w})_i - b_i)$, or equivalently, $\phi \geq \max_{\mathbf{x}} F^{\mathbf{w}}(\mathbf{x})$. The second key insight is that this new constraint can be implemented using a construction that follows the structure of variable elimination in cost networks. (An identical construction applies to the complementary constraints: $\phi \geq b_i - (C\mathbf{w})_i$.)

Consider any function e used within $\mathcal{F}$ (including the original f_i's), and let $\mathbf{Z}$ be its domain. For any assignment $\mathbf{z}$ to $\mathbf{Z}$, we introduce a variable into the linear program whose value represents $u^e_{\mathbf{z}}$. For the initial functions $f_i^{\mathbf{w}}$, we include the constraint that $u^{f_i}_{\mathbf{z}} = f_i^{\mathbf{w}}(\mathbf{z})$. As $f_i^{\mathbf{w}}$ is linear in $\mathbf{w}$, this constraint is linear in the LP variables. Now, consider a new function e introduced into $\mathcal{F}$ by eliminating a variable X_l. Let $e_1, \ldots, e_L$ be the functions extracted from $\mathcal{F}$, and let $\mathbf{Z}$ be the domain of the resulting e. We introduce a set of constraints:

$$u^e_{\mathbf{z}} \geq \sum_{j=1}^{L} u^{e_j}_{(\mathbf{z},x_l)[\mathbf{Z}_j]} \qquad \forall x_l,$$

where $\mathbf{Z}_j$ is the domain of e_j and $(\mathbf{z}, x_l)[\mathbf{Z}_j]$ denotes the value of the instantiation $(\mathbf{z}, x_l)$ restricted to $\mathbf{Z}_j$. Let e_n be the last function generated in the elimination, and recall that its domain is empty. Hence, we have only a single variable u^{e_n}. We introduce the additional constraint $\phi \geq u^{e_n}$.

To understand this construction, consider our simple example above, and assume we want to express the fact that $\phi \geq \max_{\mathbf{x}} F^{\mathbf{w}}(\mathbf{x})$. We first introduce a set of variables $u^{f_1}_{x_1,x_2}$ for every instantiation of values x_1, x_2 to the variables X_1, X_2. Thus, if X_1 and X_2 are both binary, we would have four such variables. We would then introduce a constraint defining the value of $u^{f_1}_{x_1,x_2}$ appropriately. For example, for our f_1 above, we would have $u^{f_1}_{t,t} = 0$ and $u^{f_1}_{t,f} = w_1$. We would have similar variables and constraints for each f_j and each value $\mathbf{z}$ in $\mathbf{Z}_j$. Note that each of the constraints is a simple equality constraint involving numerical constants and perhaps the weight variables $\mathbf{w}$.

Next, we introduce variables for each of the intermediate expressions. For example, we would have a set of variables $u^{e_1}_{x_2,x_3}$; for each of them, we would have a set of constraints

$$u^{e_1}_{x_2,x_3} \geq u^{f_3}_{x_2,x_4} + u^{f_4}_{x_3,x_4}$$

one for each value x_4 of X_4. We would have a similar set of constraint for $u^{e_2}_{x_1,x_2}$ in terms of $u^{f_2}_{x_1,x_3}$ and $u^{e_1}_{x_2,x_3}$. Note that each constraint is a simple linear inequality.

It is easy to show that minimizing ϕ "drives down" the value of each variable $u^e_{\mathbf{z}}$, so that

$$u^e_{\mathbf{z}} = \max_{x_i} \sum_{j=1}^{\ell} u^{e_j}_{(\mathbf{z},x_i)[\mathbf{Z}_j]}.$$

We can then prove, by induction, that u^{e_n} must be equal to $\max_{\mathbf{x}} \sum_j f_j^{\mathbf{w}}(\mathbf{x})$. Our constraints on ϕ ensure that it is greater than this value, which is the maximum of $\sum_j f_j^{\mathbf{w}}(\mathbf{x})$ over the entire state space. The LP, subject to those constraints, will minimize ϕ, guaranteeing that we find the vector $\mathbf{w}$ that achieves the lowest value for this expression.

Returning to our original formulation, we have that $\sum_j f_j^{\mathbf{w}}$ is $C\mathbf{w} - \mathbf{b}$ in one set of constraints and $\mathbf{b} - C\mathbf{w}$ in the other. Hence our new set of constraints is equivalent to the original set: $\phi \geq C\mathbf{w} - \mathbf{b}$ and $\phi \geq \mathbf{b} - C\mathbf{w}$. Minimizing ϕ finds the $\mathbf{w}$ that minimizes the $\mathcal{L}_\infty$ norm, as required.

4.3 Factored solution algorithms

The factored max-norm projection algorithm described previously is the key to applying our max-norm solution algorithms in the context of factored MDPs.

Approximate value iteration: Let us begin by considering the value iteration algorithm. As described above, the algorithm repeatedly applies two steps. It first applies the Bellman operator to obtain $\overline{\mathcal{V}}^{(t+1)} = \mathcal{T}^* A\mathbf{w}^{(t)}$. Let $\pi^{(t)}$ be the stationary policy $\pi^{(t)} = \textit{Greedy}(A\mathbf{w}^{(t)})$. Note that $\pi^{(t)}$ corresponds to the maximizing policy used in the Bellman operator at iteration t, i.e., $\mathcal{T}_{\pi^{(t)}} = \mathcal{T}^*$ for iteration t. Thus, we can compute $\overline{\mathcal{V}}^{(t+1)} = \mathcal{T}^* A\mathbf{w}^{(t)}$ by first computing $\pi^{(t)}$ and then performing a *backprojection* operation for this fixed policy, i.e., $\overline{\mathcal{V}}^{(t+1)} = R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} A\mathbf{w}^{(t)}$. Assume, for the moment, that $P_{\pi^{(t)}}$ is a factored transition model; we discuss the computation of the greedy policy and the resulting transition model below. As discussed by Koller and Parr [1999], the backprojection operation can be performed efficiently if the transition model and the value function are both factored appropriately. Furthermore, the resulting function $\overline{\mathcal{V}}^{(t+1)}$ is also factored, although the factors involve larger domains.

To recap their construction briefly, let f be a restricted domain function with domain $\mathbf{Y}$; our goal is to compute $P_\tau f$. We define the *back-projection of* $\mathbf{Y}$ *through* τ as the set of parents of $\mathbf{Y}'$ in the transition graph G_τ; $\Gamma_\tau(\mathbf{Y}') = \cup_{Y' \in \mathbf{Y}'} \textit{Parents}_\tau(Y')$. It is easy to show that: $(P_\tau f)(\mathbf{x}) = \sum_{\mathbf{y}'} P_\tau(\mathbf{y}' \mid \mathbf{z}) f(\mathbf{y}')$, where $\mathbf{z}$ is the value of $\Gamma_\tau(\mathbf{Y})$ in $\mathbf{x}$. Thus, we see that $(P_\tau f)$ is a function whose domain is restricted to $\Gamma_\tau(\mathbf{Y})$. Note that the cost of the computation depends linearly on $|\mathrm{Dom}(\Gamma_\tau(\mathbf{Y}))|$, which depends on $\mathbf{Y}$ (the domain of f) and on the complexity of the process dynamics.

Therefore, $\overline{\mathcal{V}}^{(t+1)}$ is composed of the sum of two factored functions: the reward function $R_{\pi^{(t)}}$ which is assumed to be factored, and the backprojected basis functions $P_{\pi^{(t)}} A$ which are also factored, as we have just shown. Finally, the second step in approximate value iteration is to compute the projection $A\mathbf{w}^{(t+1)} = \Pi_\infty \overline{\mathcal{V}}^{(t+1)}$, i.e., find $\mathbf{w}^{(t+1)}$ that minimizes $\left\| A\mathbf{w}^{(t+1)} - \overline{\mathcal{V}}^{(t+1)} \right\|_\infty$. As both $\overline{\mathcal{V}}^{(t+1)}$ and $A\mathbf{w}^{(t+1)}$ are factored, we can perform this computation using the factored LP discussed in the previous section.

Approximate policy iteration: Policy iteration also iterates through two steps. The policy improvement step simply computes the greedy policy relative to $\pi^{(t)}$. We discuss this step below. The approximate value determination step computes:

$$\arg\min_{\mathbf{w}} \left\| \left(A - \gamma P_{\pi^{(t)}} A\right) \mathbf{w}^{(t)} - R_{\pi^{(t)}} \right\|_\infty .$$

Again, assuming that $P_{\pi^{(t)}}$ is factored, we can conclude that $C = (A - \gamma P_{\pi^{(t)}} A)$ is also a matrix whose columns correspond to restricted-domain functions. The target $\mathbf{b} = R_{\pi^{(t)}}$ corresponds to the reward function, which is assumed to be factored. Thus, we can again apply our factored LP.

In our discussion so far, we assumed that we have some mechanism for computing the greedy policy *Greedy*$(A\mathbf{w}^{(t)})$, and that this policy has a compact representation and a factored transition model. As shown in [Koller and Parr, 2000], the greedy policy relative to a factored value function has the form of a *decision list*. More precisely, the policy can be written in the form $\langle \mathbf{t}_1, a_1 \rangle, \langle \mathbf{t}_2, a_2 \rangle, \ldots, \langle \mathbf{t}_L, a_L \rangle$, where each $\mathbf{t}_i$ is an assignment of values to some small subset $\mathbf{T}_i$ of variables, and each a_i is an action. The optimal action to take in state $\mathbf{x}$ is the action a_j corresponding to the first event $\mathbf{t}_j$ in the list with which $\mathbf{x}$ is consistent.

Koller and Parr show the greedy policy can be represented compactly using such a decision list, and provide an efficient algorithm for computing it. Unfortunately, as they discuss, the resulting transition model is usually not factored. Thus, we cannot simply apply our factored LP construction, as suggested above. However, we can adapt it to deal with this issue.

The basic idea is to introduce cost networks corresponding to each branch in the decision list. Let S_i be the set of states $\mathbf{x}$ for which $\mathbf{t}_i$ is the first event in the decision list for which $\mathbf{x}$ is consistent. Recall that our LP construction defines a set of constraints that imply that $\phi \geq ((C\mathbf{w})_i - b_i)$ for each state i. Instead, we will have a separate set of constraints for the states in each subset S_i. For each state in S_i, we know that action a_i is taken. Hence, we can apply our construction above using P_{a_i} — a transition model which is factored by assumption — in place of the non-factored $P_{\pi^{(t)}}$.

The only issue is to guarantee that the cost network constraints derived from this transition model are applied only to states in S_i. Specifically, we must guarantee that they are applied only to states consistent with $\mathbf{t}_i$, but not to states that are consistent with some $\mathbf{t}_j$ for $j < i$. To guarantee the first condition, we simply instantiate the variables in $\mathbf{T}_i$ to take the values specified in $\mathbf{t}_i$. That is, our cost network now considers only the variables in $\{X_1, \ldots, X_n\} - \mathbf{T}_i$, and computes the maximum only over the states consistent with $\mathbf{T}_i = \mathbf{t}_i$. To guarantee the second condition, we ensure that we do not impose any constraints on states associated with previous decisions. This is achieved by adding indicators $\mathcal{I}_j$ for each previous decision $\mathbf{t}_j$, with weight $-\infty$. These will cause the constraints associated with $\mathbf{t}_i$ to be trivially satisfied by states in S_j for $j < i$. Note that each of these indicators is a restricted domain function of $\mathbf{T}_j$ and can be handled in the same fashion as all other terms in the factored LP. Thus, for a decision list of size L, our factored LP contains constraints from $2L$ cost networks.

It is instructive to compare our max-norm policy iteration algorithm with $\mathcal{L}_2$-projection policy iteration algorithm of Koller and Parr [2000] in terms of computational costs per iteration and implementation complexity. Computing the $\mathcal{L}_2$ projection requires (among other things) a series of dot product operations between basis functions and backprojected basis functions $\langle h_i \bullet P_\pi h_j \rangle$. These expressions are easy to compute if P_π refers to the transition model of a particular action a. However, if the policy π is represented as a decision list, as is the result of the policy improvement step, then this step becomes much more complicated. In particular, for every point in the decision list, for every pair of basis functions i and j, and for each assignment to the variables in $\mathrm{Dom}(h_i) \cup \mathrm{Dom}(P_a h_j)$, it requires the solution of a counting problem which is $\sharp P$-complete in general. Although, Koller and Parr show that this computation can be performed using a Bayesian Network (BN) inference, the algorithm still requires a BN inference for each one of those assignments at each point of the decision list. This makes the algorithm very difficult to implement efficiently in practice.

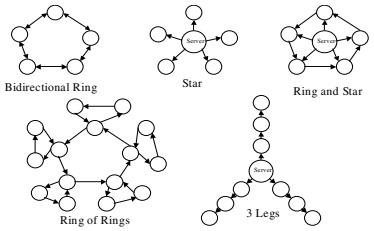


Figure 1: Network topologies tested.

The max-norm projection, on the other hand, relies on solving a linear program at every iteration. The size of the linear program depends on the cost networks generated. As we discuss, two cost networks are needed for each point in the decision list. The complexity of each of these cost networks is approximately the same as one of the BN inferences in the counting problem for the $\mathcal{L}_2$ projection. Overall, for each point in the decision list, we have a total of two of these "inferences", as opposed to one for each assignment of $\mathrm{Dom}(h_i) \cup \mathrm{Dom}(P_a h_j)$ for every pair of basis functions i and j. Thus, the max-norm policy iteration algorithm is substantially less complex computationally than the approach based on $\mathcal{L}_2$-projection. Furthermore, the use of linear programming allows us to rely on existing LP packages (such as CPLEX), which are very highly optimized.

5 Experimental Results

The factored representation of a value function is most appropriate in certain types of systems: Systems that involve many variables, but where the strong interactions between the variables are fairly sparse, so that the decoupling of the influence between variables does not induce an unacceptable loss in accuracy. As argued by Herbert Simon [1981] in "Architecture of Complexity", many complex systems have a "nearly decomposable, hierarchic structure", with the subsystems of such systems interacting only weakly between them. We selected to try our algorithm on a problem that we believe characterizes this type of structure.

The problem relates to a system administrator who has to maintain a network of computers; we experimented with various network architectures, shown in Fig. 1. Machines fail randomly, and a faulty machine increases the probability that its neighboring machines will fail. At every time step, the SysAdmin can go to one machine and reboot it, causing it to be working in the next time step with high probability. Each machine receives a reward of 1 when working (except in the ring, where one machine receives a reward of 2, to introduce

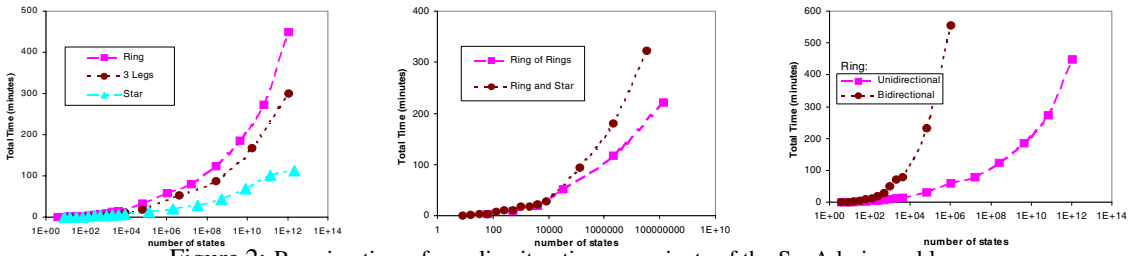
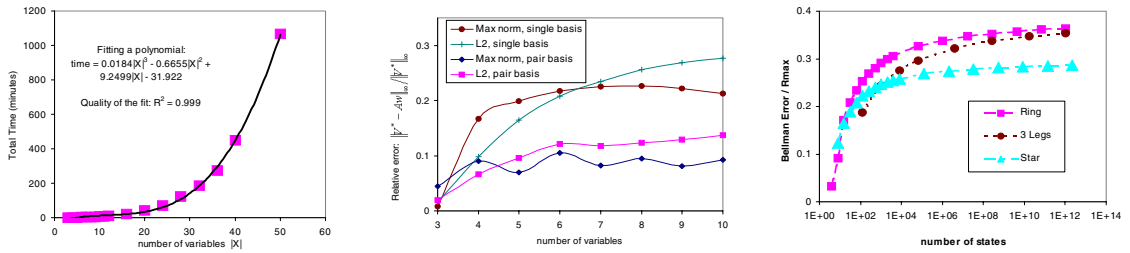


Figure 2: Running times for policy iteration on variants of the SysAdmin problem.



(a) (b) (c)

Figure 3: (a) Fitting a polynomial to the running time of the unidirectional ring; (b) Relative error to optimal value function $\mathcal{V}^*$ and comparison to $\mathcal{L}_2$ projection; (c) For large models, measuring Bellman error after convergence.

some asymmetry), a zero reward is given to faulty machines, and the discount factor is $\gamma = 0.95$. Note that the additive structure of the reward function makes it unsuitable for the tree-structured representation used by Boutilier *et al.* [1999].

The basis functions included independent indicators for each machine, with value 1 if it is working and zero otherwise (each, a restricted domain function of a single variable), and the constant basis, whose value is 1 for all states.

We implemented the factored policy and value iteration algorithms in Matlab, using CPLEX as the LP solver. In our experiments, most of the time was spent in Matlab generating the LP constraints; CPLEX is a remarkably efficient and reliable solver, allowing even large LPs to be solved very quickly. We present only results for policy iteration. The time per iteration was about equal for policy and value iteration, but policy iteration converged in many fewer iterations, only about 4 or 5 iterations in the models we tested.

To evaluate the complexity of the algorithm, tests were performed with increasing number of states, that is, increasing number of machines on the network. Fig. 2 shows the running time for increasing problem sizes, for various architectures. The simplest one is the "Star", where the backprojection of each basis function has domain restricted to two variables and the largest factor in the cost network has domain restricted to two variables. The most difficult one was the "Bidirectional Ring", where factors contain five variables.

Note, the number of states is growing exponentially (indicated by the log scale in Fig. 2), but running times increase only logarithmically in the number of states, i.e., polynomially in the number of variables. This is illustrated by Fig. 3(a), where we fit a 3rd order polynomial to the running times for the "unidirectional ring". Note, the problem size grows quadratically with the number of variables: adding a machine to the network also adds the possible action of fixing that machine. For this problem, the computation cost of our algorithm empirically grows approximately as $O\left((|\mathbf{X}| \cdot |A|)^{1.5}\right)$.

For further evaluation, we measured the error in our approximate value function relative to the true optimal value function $\mathcal{V}^*$. Note that it is only possible to compute $\mathcal{V}^*$ for small problems; in our case, we were only able to go up to 10 machines. For comparison, we also evaluated the error in the approximate value function produced by the $\mathcal{L}_2$-projection algorithm of Koller and Parr. As we discussed above, the $\mathcal{L}_2$ projections in factored MDPs by Koller and Parr [2000] are difficult and time consuming; hence, we were only able to compare the two algorithms for smaller problems, where an equivalent $\mathcal{L}_2$-projection can be implemented using an explicit state space formulation. Results for both algorithms are presented in Fig. 3(b), showing the relative error of the approximate solutions to the true value function for increasing problem sizes. The results indicate that, for larger problems, the max-norm formulation generates a better approximation of the true optimal value function $\mathcal{V}^*$ than the $\mathcal{L}_2$-projection. Here, we used two types of basis functions: the same single variable basis, and pairwise basis, which also includes indicators for neighboring pairs of machines. As expected, pairwise basis generated better approximations.

For these small problems, we can also compare the actual value of the policy generated by our algorithm to the value of the optimal policy. Here, the value of the policy generated by our algorithm is much closer to the value of the optimal policy than the error implied by the difference between our approximate value function and $\mathcal{V}^*$. For example, for the "Star" architecture with one server and up to 6 clients, our approximation with single variable basis functions had relative error of 12%, but the policy we generated had the same value as the optimal policy. In this case, the same was true for the policy generated by the $\mathcal{L}_2$ projection. In an "Unidirectional Ring" with 8 machines and pairwise basis, the relative error between our approximation and $\mathcal{V}^*$ was about 10%, but the resulting policy only had a 6% loss over the optimal policy. For the same problem, the $\mathcal{L}_2$ approximation has a value function error of 12%, and a true policy loss was 9%. In other words, both methods induce policies that have lower errors than the

errors in the approximate value function (at least for small problems). However, our algorithm continues to outperform the $\mathcal{L}_2$ algorithm, even relative to actual policy loss.

For the large models, we can no longer compute the correct value function, so we cannot evaluate our results by computing $\|\mathcal{V}^* - Aw\|_\infty$. However, the *Bellman error*, defined as $\mathrm{BellmanErr}(\mathcal{V}) = \|\mathcal{T}^*\mathcal{V} - \mathcal{V}\|_\infty$, can be used to provide a bound: $\|\mathcal{V}^* - Aw\|_\infty \leq \frac{\mathrm{BellmanErr}(Aw)}{1-\gamma}$ [Williams and Baird, 1993]. Thus, we use the Bellman error to evaluate our answers for larger models. Fig. 3(c) shows that the Bellman error increases very slowly with the number of states.[1]

Finally, we can look at the actual policies generated in our experiments. First, we noted that these tended to be short, the length of the final decision list policy grew approximately linearly with the number of machines. Furthermore, the policy itself is often fairly intuitive. In the "Ring and Star" architecture, for example, the decision list says: If the server is faulty, fix the server; else, if another machine is faulty, fix it.

6 Conclusions

In this paper, we presented new algorithms for approximate value and policy iteration. Unlike previous approaches, our algorithms directly minimize the $\mathcal{L}_\infty$ error, and therefore have better theoretical performance guarantees than algorithms that optimize the $\mathcal{L}_2$-norm.

We have shown that the $\mathcal{L}_\infty$ error can be minimized using linear programming, and provided an approach for representing this LP compactly for factored MDPs, allowing these algorithms to be applied efficiently even for MDPs with exponentially large state spaces. Our algorithms are more efficient and substantially easier to implement than previous algorithms based on the $\mathcal{L}_2$-projection.

We have presented results on a structured problem, representing a simplified version of a maintenance task. Our results show that our methods scale effectively to very large factored MDPs, and that our approach can exploit problem-specific structure to efficiently generate approximate solutions to complex problems.

The success of our algorithm depends on our ability to capture the most important structure in the value function using a linear factored approximation. This ability, in turn, depends on the choice of the basis functions and on the properties of the domain. The algorithm currently requires the designer to specify the factored basis functions. This is a limitation compared to other algorithms (e.g., [Dearden and Boutilier, 1997]), which are fully automated. However, our experiments indicate that a few simple rules are often quite successful in designing a basis. First, we ensure that the reward function is representable by our basis. A simple basis that, in addition, contains a separate set of indicators for each variable is often successful. We can also add indicators over pairs of variables; most simply, we can choose these according to the DBN transition model, where an indicator is added between variables X_i and each one of the variables in $\mathit{Parents}(X_i)$, thus representing one-step influences. This procedure can be extended, adding more basis to represent more influences as required. Thus, the structure of the DBN gives us indications of how to choose the basis functions. Other sources of prior knowledge can also be included for further specifying the basis.

The quality of our approximation also depends strongly on the structure of the domain. As we discussed above, our approximation is most successful in systems where variables are tightly coupled to only a small number of other variables. We believe that, for systems of this type, an approximation as a factored linear value function can be very accurate.

There are many possible extensions to this work. In particular, we hope to extend our algorithms to utilize other types of structure in the representation. One interesting direction involves factored action models, where multiple actions are taken simultaneously. Another involves the use of asymmetries (context-specificity) in the value function, as in the work of Dearden and Boutilier [1997], providing a complementary source of structure to the factorization used in our work.

Acknowledgments: We are very grateful to Dirk Ormoneit and Uri Lerner for many useful discussions. This work was supported by the ONR under the MURI program "Decision Making Under Uncertainty" and by the Sloan Foundation.

References

[Bellman *et al.*, 1963] R. Bellman, R. Kalaba, and B. Kotkin. Polynomial approximation – a new computational technique in dynamic programming. *Math. Comp.*, 17(8):155–161, 1963.

[Bertele and Brioschi, 1972] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, New York, 1972.

[Boutilier *et al.*, 1999] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 1999.

[Cheney, 1982] E. W. Cheney. *Approximation Theory*. Chelsea Publishing Co., New York, NY, 2nd edition, 1982.

[Dean and Kanazawa, 1989] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[Dearden and Boutilier, 1997] R. Dearden and C. Boutilier. Abstraction and approximate decision theoretic planning. *Artificial Intelligence*, 89(1):219–283, 1997.

[Dechter, 1999] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artif. Intel.*, 113(1–2):41–85, 1999.

[Koller and Parr, 1999] D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs. In *IJCAI*, 1999.

[Koller and Parr, 2000] D. Koller and R. Parr. Policy iteration for factored mdps. In *Proc. of Uncertainty in AI (UAI-00)*, 2000.

[Puterman, 1994] M. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. Wiley, New York, 1994.

[Simon, 1981] Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, Massachusetts, second edition, 1981.

[Stiefel, 1960] E. Stiefel. Note on Jordan elimination, linear programming and Tchebycheff approximation. *Numerische Mathematik*, 2:1 – 17, 1960.

[Tsitsiklis and Van Roy, 1996] J. N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

[Williams and Baird, 1993] R. Williams and L. Baird. Tight performance bounds on greedy policies based on imperfect value functions. Tech. Report, Northeastern Univ., Massachusetts, 1993.

[1] Note that computing the Bellman error involves a maximization over the state space. Thus, the complexity of this computation grows exponentially with the number of variables. However, this maximization can be performed by a cost network using the same construction as in our max-norm projection.

UNCERTAINTY AND PROBABILISTIC REASONING

FACTORED MARKOV DECISION PROCESSES

Solving Factored MDPs via Non-Homogeneous Partitioning

Kee-Eung Kim and **Thomas Dean**
Department of Computer Science
Brown University
Providence, RI 02912-1910
{kek,tld}@cs.brown.edu

Abstract

This paper describes an algorithm for solving large state-space MDPs (represented as *factored* MDPs) using search by successive refinement in the space of non-homogeneous partitions. Homogeneity is defined in terms of bisimulation and reward equivalence within blocks of a partition. Since homogeneous partitions that define equivalent reduced state-space MDPs can have a large number of blocks, we relax the requirement of homogeneity. The algorithm constructs approximate aggregate MDPs from non-homogeneous partitions, solves the aggregate MDPs exactly, and then uses the resulting value functions as part of a heuristic in refining the current best non-homogeneous partition. We outline the theory motivating the use of this heuristic and present empirical results and comparisons.

1 Introduction

Markov Decision Processes (MDPs) employing representations that factor states, actions and their associated transition and reward functions in terms of component functions of state variables (fluents) have surfaced as plausible models for planning under uncertainty [Boutilier *et al.*, 1999]. Since the number of states in these factored MDPs (FMDPs) is exponential in the number of fluents, traditional iterative methods that require enumerating states are typically not effective. In this paper, we solve FMDPs by constructing and refining non-homogeneous partitions of the state space. In a homogeneous partition, any two states in a block have the same reward and the same distribution with respect to transitions to other blocks. Non-homogeneous partitions allow variation among the states in a block with regard to rewards and block transition distributions. From a non-homogeneous partition, we construct an aggregate MDP by averaging the transition probabilities and the rewards within blocks.

The algorithm described in this paper solves FMDPs by successive refinement in the space of non-homogeneous partitions. We use factored representations to encode the partitions and the aggregate MDPs we construct from them. We can solve the aggregate MDPs in time polynomial in the number of blocks in the partition. The resulting optimal policies (optimal for the aggregate MDPs) also serve as policies for the original MDP and the value function for the optimal policy in the aggregate MDP serves as an estimate for the value of the policy in the original MDP. Given this estimate and the current partition, we choose the refinement that yields the greatest improvement and iterate. In the remainder of this paper, we present some background, provide a theoretical motivation for our refinement heuristic, and describe the results of a series of experiments.

2 Factored MDPs (FMDPs)

Definition 1 (FMDP) *An* FMDP *is defined as a tuple* $M = \{\vec{X}, A, T, R\}$ *where*

- $\vec{X} = [X_1, \ldots, X_n]$ *is a vector of fluents which collectively define the state. We use* Ω_{X_i} *to denote the set of values for* X_i. Ω_{X_i} *is the sample space of* X_i *when* X_i *is considered as a random variable. Thus, the state space for* M *is* $\Omega_{\vec{X}} = \prod_i \Omega_{X_i}$. *We use lower case letters* $\vec{x} = [x_1, \ldots, x_n]$ *to denote a particular instantiation of the fluents, and* $X_{i,t}$ *and* $x_{i,t}$ *denote, respectively, a fluent and its value at a particular time* t.
- A *is the set of actions.*
- T *is the set of transition probabilities, represented as the set of conditional probability distributions, one for each action and fluent:*

$$T(\vec{X}_t, a, \vec{X}_{t+1}) = \prod_{i=1}^{n} P(X_{i,t+1} | \text{pa}(X_{i,t+1}), a)$$

  where $\text{pa}(X_{i,t+1})$ *denotes the set of parents of* $X_{i,t+1}$ *in the graphical model (see below). Note that* $\forall i, \text{pa}(X_{i,t+1}) \subseteq \{X_{1,t}, \ldots, X_{n,t}\}$.
- $R : \vec{X} \rightarrow \Re$ *is the reward function. Without loss of generality, we define the reward to be determined by both state and action (*$R : \vec{X} \times A \rightarrow \Re$*).*

Figure 1 shows an example of a graphical model for the dynamics governing an action in a toy robot domain with five boolean variables. The robot has to deliver a cup of coffee whenever requested. Unfortunately, the coffee bar is outside the building and the robot receives a small amount of punishment if it gets wet. Each fluent denotes a particular aspect of

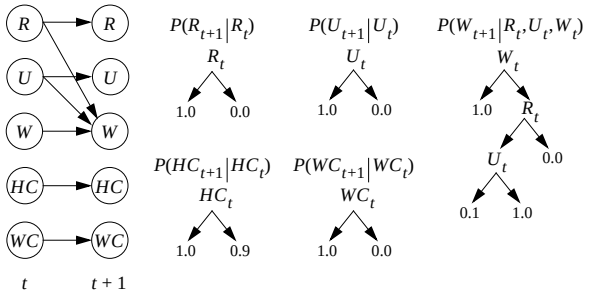


Figure 1: Graphical representation of an FMDP describing a toy robot domain.

the world: the weather outside being rainy (R), the robot having an umbrella (U), the robot being wet (W), the robot holding coffee (HC), and someone wanting coffee (WC). Note that the probabilistic effect of each fluent at time $t+1$ is conditioned on a small number of fluents (the set of parent fluents, *e.g.*, pa$(W_{t+1}) = \{R_t, U_t, W_t\}$) at time t. The conditional probabilities are stored as trees rather than tables for further savings in the size of the representation. Thus, we complete the description of the state dynamics by specifying conditional probability trees (CPTs) [Boutilier *et al.*, 1995] for each fluent and action. Similar representations have been used to model Markovian processes with factored state spaces, *e.g.*, the Two-stage Temporal Bayesian Network (2TBN) [Dean and Kanazawa, 1989] and the Dynamic Bayesian Network (DBN) [Forbes *et al.*, 1995]. The reward function R is also designated in terms of trees.

FMDPs exploit conditional independence among the fluents given their parents in the graphical model to achieve economy of representation. However, the underlying structure in the domain that makes compact representations possible does not lead necessarily to an efficient FMDP algorithm. There are algorithms modeled after classical MDP algorithms that use dynamic programming to iteratively update a value function represented in a factored form; however, the size of the value functions so represented can easily explode. There are also algorithms that compress the value functions in an effort to prevent an explosion, *e.g.*, using a tree representation for value functions with pruning techniques [Boutilier and Dearden, 1996], or using Algebraic Decision Diagrams (ADDs) for further reduction [Hoey *et al.*, 1999]. These algorithms are examples of Value Function Approximation (VFA) algorithms applied to FMDPs [Gordon, 1995; Tsitsiklis and Van Roy, 1996; Koller and Parr, 1999].

3 Stochastic Bisimulation Equivalence

Dean and Givan [1997] introduce the notion of *stochastic bisimulation homogeneity* for FMDPs. It is an extension of the state equivalence relation for minimizing Finite State Automata (FSA) to that of probabilistic FSAs. Dean and Givan's model minimization algorithm for FMDPs partitions the state space into *stable* blocks.

Definition 2 (Stable Block and Homogeneous Partition) *A block C of a partition P is said to be* stable with respect to a block B of P and an action a *if and only if every state in C has the same transition probability of ending up in block B by action a. Mathematically,*

$$\exists c \in [0,1] \text{ such that } \forall \vec{x}_t \in C, T(\vec{x}_t, a, B) = c$$

where

$$T(\vec{x}_t, a, B) = \sum_{\vec{x}_{t+1} \in B} T(\vec{x}_t, a, \vec{x}_{t+1}).$$

We say that C is stable *if C is stable with respect to every block of P and action in A. P is said to be* homogeneous *if and only if every block is stable.*

Given a homogeneous partition of an FMDP, we can define an aggregate MDP that is *equivalent* to the original FMDP. Every state within a block of a homogeneous partition has the same state dynamics with respect to any policy, *i.e.*, the transition probability of moving into a block is the same for every state in the same block. Thus, if every state within a block of a homogeneous partition has the same reward, the partition yields a reduced model of the original FMDP.

Givan *et al.* [2000] introduce ϵ*-homogeneity* for finding an approximately homogeneous partition with few blocks. We say that a block C is ϵ*-stable* with respect to B and a if

$$\exists c \in [0,1] \text{ such that } \forall \vec{x}_t \in C, |T(\vec{x}_t, a, B) - c| \leq \epsilon.$$

If every block of partition P is ϵ-stable with respect to every other block of P and every action, we say P is an ϵ-homogeneous partition. Qualitatively speaking, by relaxing the equality to be *approximately equal with error within* ϵ, we get a smaller partition than if we required strict homogeneity. The error in the optimal value function computed from an ϵ-homogeneous partition is discussed in the work of Singh and Yee [1994] and White and Eldeib [1994]. Most of the previous work on computing approximate solutions of FMDPs follow similar analyses [Boutilier and Dearden, 1996; St-Aubin *et al.*, 2000].

4 Non-Homogeneous Partitions for FMDPs

There are two important questions remaining to be answered concerning partition refinement techniques for FMDPs. First, what class of functions should we allow to represent block formulae? If we restrict the representational power of the formulae too much, we may end up with large partitions but with easily manipulable block formulae. If we allow arbitrary functions, manipulating block formulae is NP-hard (see [Goldsmith and Sloan, 2000] for recent analysis.) though we can compute the coarsest homogeneous partition which may be small. Second, what if the coarsest homogeneous partition is still exponentially large compared to the description size of the FMDP? Calculating ϵ-homogeneous partition helps, but note that we don't have a fine control over the size of the partition — all we know is that increasing ϵ will generally reduce the size of the partition and decreasing it will generally do the opposite.

Based on the above observation, we describe a new state aggregation algorithm that does not search for the coarsest homogeneous partition. In fact, the technique is not driven

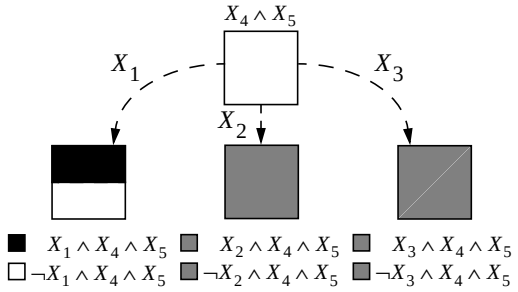


Figure 2: Chopping the block $X_4 \wedge X_5$ with respect to X_1, X_2 or X_3. Note that the blocks resulting from the CHOP operator are not stable in general.

by the notion of stochastic bisimulation homogeneity. Given an initial non-homogeneous partition of the state space, the algorithm iteratively decides which block to refine so that it produces at the end an approximately optimal policy without generating a huge partition. Thus, given a partition, the algorithm has to determine which block to refine with respect to which fluent. Note that the partitions we consider are not ϵ-homogeneous given that the transition probabilities can widely differ.

We show that the optimal value functions calculated on non-homogeneous partitions provide the basis for an effective heuristic for selecting the block-fluent pair to chop. First, we define how we construct the MDP from a non-homogeneous partition.

Definition 3 (MDP from Non-Homogeneous Partition) *Given an* FMDP $M = (\vec{X}, A, T, R)$ *and a non-homogeneous partition P of the state space $\Omega_{\vec{X}}$, the aggregate* MDP *induced by P, denoted as $M_P = (P, A, T_P, R_P)$, is defined as*

$$T_P(C, a, B) = \frac{1}{|C|} \sum_{\vec{x} \in C, \vec{x}' \in B} T(\vec{x}, a, \vec{x}')$$

$$R_P(C, a) = \frac{1}{|C|} \sum_{\vec{x} \in C} R(\vec{x}, a)$$

for all $B, C \in P$ and $a \in A$. V_P^* denotes the optimal value function of M_P, which is a mapping from $\Omega_{\vec{X}}$ to $\Re$. Note that for any $\vec{x}$ and $\vec{x}'$ in the same block of P, $V_P^*(\vec{x}) = V_P^*(\vec{x}')$ and $\pi^*(\vec{x}) = \pi^*(\vec{x}')$.

Given a partition P, the algorithm selects block C and fluent X for generating a refined partition P'. P' has the same blocks as P except for C which is replaced by CHOP(P, C, X). Figure 2 illustrates how the CHOP operator generates refined blocks of C. Recalling that n is the number of fluents in the domain, we note that there are at most $n|P|$ different refined partitions that can be obtained by the CHOP operator. For each refined partition P' generated by the CHOP operator, the algorithm constructs $M_{P'}$ and calculates the optimal value function $V_{P'}^*$.

We will present the algorithm shortly. But first, we would like to know how good it is to use M_P to solve the original FMDP. We prove that given any non-homogeneous partition

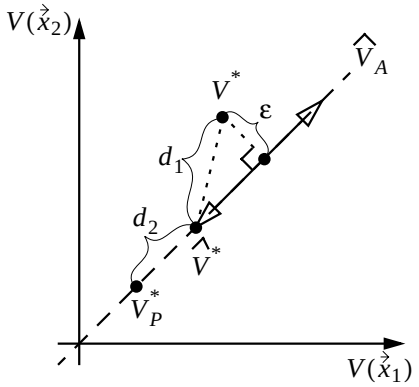


Figure 3: Value functions for an MDP constructed from a non-homogeneous partition. The figure illustrates an example where the MDP has two states and the partition has one block. The dashed line is the set of representable approximate value functions. $d_1 \leq 2(1 + \frac{\gamma}{1-\gamma})\epsilon$ and $d_2 \leq \frac{\|LV_P^* - V_P^*\|_\infty}{1-\gamma}$.

P, the optimal value function of M_P, which we define as V_P^*, is within a bounded distance from the true value function V^*. To this end, we introduce a definition and a theorem from Gordon [1995] that are used in proving Theorem 2.

Definition 4 (Averager) *An approximation function is an* averager *if, given the target vector U, the approximation function generates $\hat{U}$ defined as*

$$\hat{U}(i) = \beta_i c_i + \sum_j \beta_{ij} U(j)$$

where c_i is a constant and β_i and β_{ij} are non-negative constants such that $\forall i, \beta_i + \sum_j \beta_{ij} = 1$.

Theorem 1 (Gordon) *Let V^* be the optimal value function for an* MDP M*, L the update (Bellman backup) operator for value iteration,*

$$LV(\vec{x}) = \max_a \big[R(\vec{x}, a) + \gamma \sum_{\vec{x}' \in \Omega_{\vec{X}}} T(\vec{x}, a, \vec{x}') V(\vec{x}')\big]$$

and A the mapping for an averager. Let V^A be any fixed point of A. Given $\|V^* - V^A\|_\infty = \epsilon$, the iteration of $L \circ A$ converges to a value function $V^{L \circ A}$ so that $\|V^* - V^{L \circ A}\| \leq 2\gamma\epsilon/(1-\gamma)$. Approximate value iteration using A returns $AV^{L \circ A}$ which satisfies $\|V^* - AV^{L \circ A}\| \leq 2\epsilon + 2\gamma\epsilon/(1-\gamma)$.

Theorem 2 (Bounded Distance between V^* and V_P^*) *Given an* FMDP $M = (\vec{X}, A, T, I, R)$ *and a partition P of the state space $\Omega_{\vec{X}}$, the optimal value function of M given as V^* and the optimal value function of M_P given as V_P^* satisfy the bound on the distance*

$$\|V^* - V_P^*\|_\infty \leq 2\Big(1 + \frac{\gamma}{1-\gamma}\Big)\epsilon + \frac{\|LV_P^* - V_P^*\|_\infty}{1-\gamma} \quad (1)$$

where $\epsilon = \min_{V_P} \|V^* - V_P\|_\infty$ (V_P being any value function such that for any block $B \in P$ and any $\vec{x}$ and $\vec{x}'$ in B, $V_P(\vec{x}) = V_P(\vec{x}')$).

Proof Recall from the definition that $\forall \vec{x} \in C$,

$$V^*(\vec{x}) = \max_a \big[R(\vec{x}, a) + \gamma \sum_{\vec{x}' \in \vec{X}} T(\vec{x}, a, \vec{x}') V^*(\vec{x}')\big]$$

and

$$V_P^*(\vec{x}) = \max_a \big[\frac{1}{|C|} \sum_{\vec{x} \in C} R(\vec{x}, a) + \gamma \frac{1}{|C|} \sum_{B \in P} \sum_{\vec{x}' \in B} T(\vec{x}, a, \vec{x}') V_P^*(B)\big].$$

We define another value function

$$\widehat{V}^*(\vec{x}) = \frac{1}{|C|} \sum_{\vec{x} \in C} \max_a \big[R(\vec{x}, a) + \gamma \sum_{B \in P} \sum_{\vec{x}' \in B} T(\vec{x}, a, \vec{x}') \widehat{V}^*(B)\big].$$

To calculate the maximum distance between V^* and $\widehat{V}^*$, we define an averager A_P as follows:

$$\widehat{V}(\vec{x}) = (A_P V)(\vec{x}) = \sum_{\vec{x}' \in C} \frac{1}{|C|} V(\vec{x}') \text{ for } \forall C \in P, \forall \vec{x} \in C.$$

Note that each fixed point of A_P assigns the same value to states in the same block. For the example shown in Figure 3, the straight dotted line satisfying $\widehat{V}_A(\vec{x}_1) = \widehat{V}_A(\vec{x}_2)$ is the set of fixed points for A_P since $\vec{x}_1$ and $\vec{x}_2$ belong to the same block. Since P is a refinement of the reward partition, $\widehat{V}^*$ is the fixed point of the mapping $L \circ A_P$, and applying the result from Gordon [1995], we have that

$$\|\widehat{V}^* - V^*\|_\infty \leq 2(1 + \frac{\gamma}{1-\gamma})\epsilon \tag{2}$$

where ϵ is the distance between V^* and the closest fixed point of A_P. The distance between $\widehat{V}^*$ and V_P^* is calculated as follows: For any $\vec{x} \in C$,

$$\begin{aligned}
&|\widehat{V}^*(\vec{x}) - V_P^*(\vec{x})| \\
&= \Big|\frac{1}{|C|} \sum_{\vec{x} \in C} \max_a \big[R(\vec{x}, a) + \gamma \sum_{B \in P} \sum_{\vec{x}' \in B} T(\vec{x}, a, \vec{x}') \widehat{V}^*(B)\big] \\
&- \frac{1}{|C|} \sum_{\vec{x} \in C} \max_a \big[R(\vec{x}, a) + \gamma \sum_{B \in P} \sum_{\vec{x}' \in B} T(\vec{x}, a, \vec{x}') V_P^*(B)\big] \\
&+ \frac{1}{|C|} \sum_{\vec{x} \in C} \max_a \big[R(\vec{x}, a) + \gamma \sum_{B \in P} \sum_{\vec{x}' \in B} T(\vec{x}, a, \vec{x}') V_P^*(B)\big] \\
&- \max_a \big[\frac{1}{|C|} \sum_{\vec{x} \in C} R(\vec{x}, a) + \gamma \frac{1}{|C|} \sum_{B \in P} \sum_{\vec{x}' \in B} T(\vec{x}, a, \vec{x}') V_P^*(B)\big]\Big| \\
&\leq \gamma \|\widehat{V}^* - V_P^*\|_\infty + \|(A_P \circ L) V_P^* - V_P^*\|_\infty \\
&= \gamma \|\widehat{V}^* - V_P^*\|_\infty + \|(A_P \circ L) V_P^* - A_P V_P^*\|_\infty \\
&\leq \gamma \|\widehat{V}^* - V_P^*\|_\infty + \|L V_P^* - V_P^*\|_\infty
\end{aligned}$$

Since the above inequality holds for $\forall C \in P$, we have

$$\|\widehat{V}^* - V_P^*\|_\infty \leq \frac{\|L V_P^* - V_P^*\|_\infty}{1 - \gamma} \tag{3}$$

1. The number of iterations N, initial partition P of the state space (reward partition), and the optimal value function V_P^* for M_P.
2. For each block $C \in P$ and fluent $X_i, 1 \leq i \leq n$, compute $M_{P'}$ where P' is the same as P except C is replaced by CHOP(P, C, X_i).
3. For each P', compute $V_{P'}^*$ and then select $P^* = \arg\max_{P'} \|V_{P'}^* - V_P^*\|$.
4. If Step 2∼3 has been run N times, halt and output $\pi_{P^*}^*$.
5. Set $P = P^*$ and go to Step 2.

Figure 4: The algorithm for finding a non-homogeneous partition for an approximately optimal policy

By combining Equation 2 and Equation 3, we have shown that

$$\begin{aligned}
\|V^* - V_P^*\|_\infty &\leq \|V^* - \widehat{V}^*\|_\infty + \|\widehat{V}^* - V_P^*\|_\infty \\
&\leq 2\Big(1 + \frac{\gamma}{1-\gamma}\Big)\epsilon + \frac{\|L V_P^* - V_P^*\|_\infty}{1-\gamma}
\end{aligned}$$

□

There are two remarks on the bound in the above theorem. First, $\epsilon \equiv \min_{V_P} \|V^* - V_P\|_\infty$ that appears in the first additive term in Equation 1 measures how fine is the partition P. ϵ is the minimum error that we can achieve by assigning values to the components of V_P, with the restriction that the values should be the same for the components that belong to the same block in P. Thus, we naturally expect that a finer partition will achieve a smaller ϵ, although it depends on how the state space is partitioned. At least, given a partition P and its refinement $P' \subseteq P$, and letting ϵ_P and $\epsilon_{P'}$ be the ϵ of P and P' respectively, we can say that $\epsilon_P \geq \epsilon_{P'}$. Note also that ϵ becomes 0 for a homogeneous partition. Second, $\|L V_P^* - V_P^*\|_\infty$ in the second additive term also vanishes as the partition becomes finer. Although there is no guarantee that the error is monotonically smaller for a finer partition and larger for a coarser partition, at least this error serves as an upper bound on the distance.

The algorithm that we describe in the following selects the *best* refined partition P' from the current partition P given by the formula

$$\arg\max_{P'} \|V_{P'}^* - V_P^*\|.$$

Theorem 2 indicates that by choosing a refinement P' of P maximizing the distance between $V_{P'}^*$ and V_P^* there is reason to believe that we are making significant progress in approaching V^*. This is the same sort of argument used to justify the use of discretization strategies for continuous state-space MDPs.

The algorithm runs iteratively by refining the partition for a pre-determined number of iterations so that we end up with a fixed-size policy. Figure 4 shows the algorithm. Note that constructing $M_{P'}$ can be done efficiently. The new transition probability matrix $T_{P'}$ typically has many components

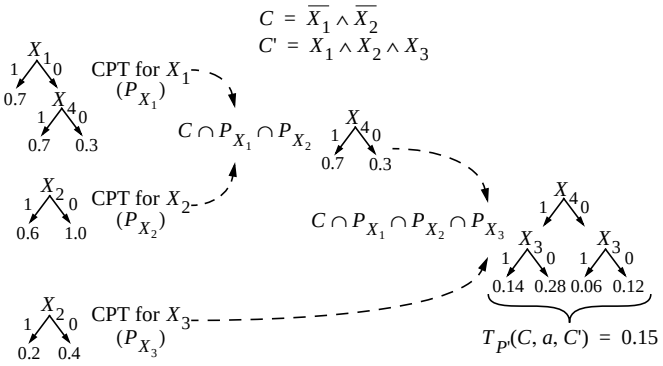


Figure 5: Calculating $T_{P'}(C, a, C')$ with decision trees.

the same as T_P. By reusing the components already calculated, we can construct the $M_{P'}$s without computing transition probabilities and rewards for all blocks. For example, if the partition P' was obtained by chopping a block $B \in P$ with respect to the fluent X_i so that $B = \{B_1, \ldots, B_{|\Omega_{X_i}|}\}$, we have

$$T_{P'}(C, a, C') = T_P(C, a, C'), \ \ \forall a \in A, \forall C \notin B, \forall C' \notin B.$$

The remaining components are computed without explicitly enumerating all of the states in the blocks. Figure 5 shows how we calculate the component $T_{P'}(C, a, C')$ using decision trees when $C \in B$ or $C' \in B$. To compute the component, we first graft CPTs and multiply the probabilities in the terminal nodes. Since we are only interested in the transition probability related to blocks C and C', we can eliminate the branches of the constructed tree that do not intersect with block C. This greatly reduces the size of the tree. The final step is to take the average of the items in the terminal nodes in the tree, weighted by the sizes of the blocks that the terminal nodes represent.

The reward function $R_{P'}$ for $M_{P'}$ is obtained similarly.

5 Experiments

The test problems used in our experiments are adopted from Hoey *et al.* [1999] and involve domains with 6 to 17 binary variables (fluents). The initial probabilities are given as a uniform distribution. For each domain, we show the performance of the policy derived from the non-homogeneous partition and the cumulative elapsed time at each iteration. We use CPLEX 6.5 for calculating optimal value functions of aggregate MDPs induced by non-homogeneous partitions, and the CUDD package [Somenzi, 1998] to implement structured versions of iterative algorithms (similar to SPUDD [Hoey *et al.*, 1999]) for evaluating the policies and calculating the optimal value functions. All experiments are run on a Sun Ultra10 with 256Mb of memory.

Figure 6 through Figure 8 illustrate the performance of the algorithm in Figure 4 on three benchmark domains. We ran the algorithm for 100 iterations, except for the COFFEE domain since it consisted of only 64 states. For each figure, the graph on the left side shows the actual performance (actual value of the optimal policy calculated from the aggregated MDP) and the heuristic value (the optimal value of the aggregated MDP) after each iteration. *The values are obtained assuming uniform starting probabilities on the states.* The graphs on the right sides show the plots of cumulative elapsed time (in seconds) after each iteration. Note that in some domains, there is a small gap between the actual performance and the heuristic value even when the policy from the aggregated MDP reached the optimal performance. This is due to the fact that the evaluation is done through an iterative method with the stopping condition $\|V_{i+1} - V_i\| \leq \delta$, which we set δ to 0.01.

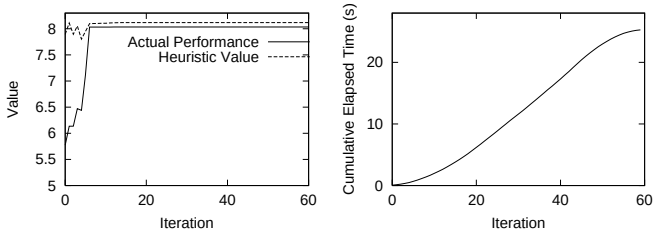


Figure 6: COFFEE domain (6 variables, 64 states). The non-homogeneous partitioning algorithm found the optimal policy after 6 chops, totaling 10 blocks in the partition. The optimal value is 8.12.

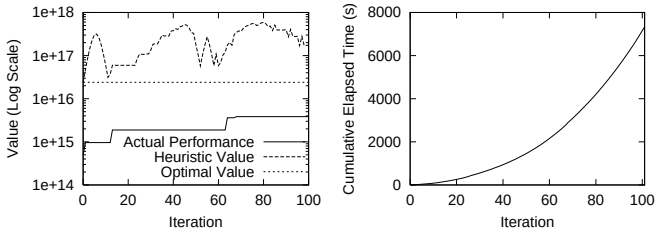


Figure 7: EXPON domain (12 variables, 4096 states). After 100 chops (113 blocks), the non-homogeneous partitioning algorithm yields an approximately optimal policy. The optimal value is 2.4×10^{16}.

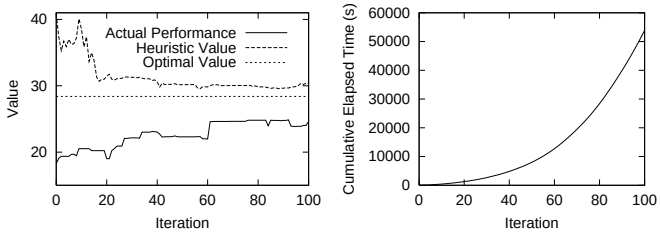


Figure 8: FACTORY domain (17 variables, 131,072 states). After 100 chops (126 blocks), the non-homogeneous partitioning algorithm yields an approximately optimal policy. The optimal value is 28.4.

The optimal value function of the EXPON domain (Figure 7) has thousands of internal nodes even when represented as an ADD. The non-homogeneous partitioning algorithm is not able to find the optimal policy with partition size less than or equal to 113, however, the policy at the end of 100th itera-

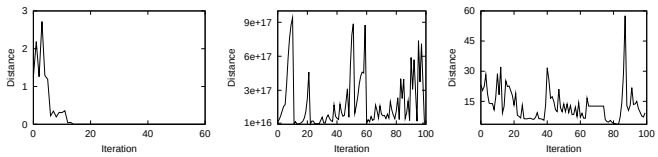


Figure 9: Distance plots between two value functions of successive aggregate MDPs ($\|V^*_{P*} - V^*_P\|$ in Figure 4). From left to right: COFFEE, EXPON, FACTORY.

tion performs 10 times better than the initial policy from the aggregate MDP induced by the reward partition. The size of the ADD representation for the optimal value function is also quite large in the case of the FACTORY domain (Figure 8). After 100 chops, totaling 126 blocks, the policy from the aggregated MDP has a value of 24.8 which is 87% of the optimal value.

We also experimented on the LINEAR domain (14 variables, 16,384 states). The optimal policy of the aggregate MDP from the reward partition yields the optimal value, *i.e.*, the optimal policy was obtained without any split. It takes 23.28 seconds for 1 iteration. When the algorithm finds out that every refinement P' of the current partition P yields $\|V^*_{P'} - V^*_P\| = 0$, then it knows the current optimal policy from the aggregated MDP is indeed optimal. Meanwhile, our implementation of the structured value iteration using CUDD package (which is not as fully optimized as SPUDD) takes 43.58 seconds (15 leaves).

Figure 9 shows the distances between two value functions of successive aggregate MDPs for each domain. We excluded the LINEAR domain since the distances were zero from the onset. Note that in the COFFEE domain, the distance decreases sharply after the optimal policy from the aggregate MDP is actually optimal. However, the distances after that are not zero, which implies the partition is still non-homogeneous. We do not observe such behavior in the EXPON and FACTORY domain since the algorithm was not able to yield the optimal policy. Although the distances between consecutive value functions are large, we note that the actual performance does not change much in the two domains. It is natural that a big difference in value functions does not necessarily imply a big difference in the actual performance. Sometimes, a block containing states with highly varing transition probabilities is not so important to reach for optimal performance, hence chopping the block does not greatly improve the actual performance of the policy.

We also compare the performance of non-homogeneous partitioning algorithm to APRICODD [St-Aubin *et al.*, 2000]. Figure 10 and Figure 11 summarize the comparison of the two algorithms on the larger domains, EXPON and FACTORY. By trial and error, we tuned the pruning parameter of the APRICODD algorithm so that the size of the approximate value function from the APRICODD algorithm is comparable to that of the non-homogeneous partition at the end of 100 chops. We used "sliding-tolerance" pruning technique with different parameters. We allow two to three times more nodes in the decision diagrams than the number of blocks in the non-homogeneous partition. Often, increasing the pruning

	Perf.	Blocks		Time
Non-homogeneous	16 %	113		7333 s
	Perf.	Nodes	Leaves	Time
APRICODD (0.4)	48 %	320	65	630 s
APRICODD (0.5)	23 %	246	33	73 s

Figure 10: Comparison of the non-homogeneous partitioning algorithm and the APRICODD on EXPON domain. The number inside the parenthesis is the pruning parameter for the "sliding-tolerance" pruning. The Perf. is the ratio between the performance of the optimal policy and that of the approximate policy assuming uniform starting distribution on states. We also present the number of nodes and leaves in the ADD representation of approximate value function from the APRICODD.

	Perf.	Blocks		Time
Non-homogeneous	87 %	126		55402 s
	Perf.	Nodes	Leaves	Time
APRICODD (0.2)	67 %	342	73	920 s
APRICODD (0.3)	26 %	252	65	893 s

Figure 11: Comparison of the non-homogeneous partitioning algorithm and the APRICODD on FACTORY domain.

parameter so that we get smaller value functions from APRICODD results in non-converging behavior. In fact, APRICODD on the EXPON domain with pruning parameter 0.5 does not converge. In this case, we stopped the algorithm when the value functions between consecutive iterations were oscillating (500 iterations). Note that we are still allowing the APRICODD to search in the space of a much richer representation for value functions — an ADD with 246 nodes can represent a much finer partition than a partition with 113 blocks. In terms of the number of blocks, the non-homogeneous partitioning algorithm performs comparable to the APRICODD algorithm. Note also that our implementation of APRICODD is not fully optimized, and that the running times may be significantly greater than those reported by the original authors.

We are currently exploring heuristics for efficiently selecting block-fluent pairs to chop. We note that solving continuous MDPs faces a similar problem, and we are experimenting on discretization heuristics such as Munos and Moore [Munos and Moore, 1999]. A preliminary result shows that this heuristic is highly effective, yielding approximately 50-fold speed up in some domains. A preliminary report on this experiment appears in [Kim, 2001].

The above experiments show two advantages of using the non-homogeneous partitioning algorithm. First, while in some domains the coarsest homogeneous partition may be quite large, it may not be critical to compute a homogeneous partition to obtain an optimal (or near optimal) policy. Structured value iteration algorithms such as SPUDD or APRICODD may incur a large cost in representing an optimal or near optimal value function, whereas the non-homogeneous partitioning algorithm does not necessarily face such a problem. Second, the algorithm provides a new approach to finding an approximately optimal policy, which allows us to specify the desired size of the policy *a priori*.

6 Conclusion and Related Work

To achieve economy of representation many algorithms aggregate states that have the same (or roughly the same) value; they do so using a variety representations ranging from simple set representations [Bertsekas and Castañon, 1989], decision trees [Boutilier *et al.*, 1995; Boutilier and Dearden, 1996] and algebraic decision diagrams [Hoey *et al.*, 1999; St-Aubin *et al.*, 2000], to linear combinations of simple basis functions [Koller and Parr, 1999] and neural networks [Bertsekas and Tsitsiklis, 1996].

While all that is necessary is that the states that are grouped together have the same value, it is often the case that the aggregated states have other properties in common. Structured value iteration [Boutilier *et al.*, 1995] and structured model reduction [Dean and Givan, 1997] group states according to stochastic bisimulation equivalence. This is a sufficient but not necessary criterion and in some cases it can result in partitions that are larger than strictly necessary. In the case of (exact) structured value iteration the leaves of the decision tree constitute the blocks of a partition that is reward and transition homogeneous. The structured model reduction algorithm successively refines an initial partition by splitting non-homogeneous blocks and is guaranteed to terminate with the coarsest homogeneous partition; unfortunately, this partition could have a number of blocks exponential in the number of fluents. This paper provides a method for splitting blocks that provides a global perspective (rather than considering each block in isolation) and for which the objective is to find a non-homogeneous partition that yields an aggregate MDP and corresponding optimal value function that is "close" to the original MDP and its optimal value function. The preliminary experimental results in this paper indicate that our algorithm achieves good performance using a compact representation encoded in terms of a non-homogeneous partition that is a fraction of the size of the representations required by other approaches.

Finding a non-homogeneous partitioning algorithm with a richer representation for blocks (such as ADDs) seems promising, and this task remains as our future work.

References

[Bertsekas and Castañon, 1989] Dimitri P. Bertsekas and David A. Castañon. Adaptive aggregation for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, 34(6):589–598, 1989.

[Bertsekas and Tsitsiklis, 1996] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[Boutilier and Dearden, 1996] Craig Boutilier and Richard Dearden. Approximating value trees in structured dynamic programming. In *Proceedings ICML-96*, 1996.

[Boutilier *et al.*, 1995] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Exploiting structure in policy construction. In *Proceedings IJCAI-95*, pages 1104–1111, 1995.

[Boutilier *et al.*, 1999] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11, 1999.

[Dean and Givan, 1997] Thomas Dean and Robert Givan. Model minimization in Markov decision processes. In *Proceedings AAAI-97*, 1997.

[Dean and Kanazawa, 1989] Thomas Dean and Keiji Kanazawa. A Model for Reasoning about Persistence and Causation. *Computational Intelligence*, pages 143–150, 1989.

[Forbes *et al.*, 1995] Jeff Forbes, Tim Huang, Keiji Kanazawa, and Stuart Russell. The BATmobile: Towards a Bayesian automated taxi. In *Proceedings IJCAI-95*, 1995.

[Givan *et al.*, 2000] Robert Givan, Sonia Leach, and Thomas Dean. Bounded-parameter Markov decision processes. *Artificial Intelligence*, 122:71–109, 2000.

[Goldsmith and Sloan, 2000] Judy Goldsmith and Robert H. Sloan. The complexity of model aggregation. In *Proceedings AIPS-2000*, 2000.

[Gordon, 1995] Geoffrey J. Gordon. Stable function approximation in dynamic programming. Technical Report CMU-CS-103, School of Computer Science, Carnegie Mellon University, 1995.

[Hoey *et al.*, 1999] Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings UAI-99*, 1999.

[Kim, 2001] Kee-Eung Kim. *Representations and Algorithms for Large Stochastic Planning Problems*. PhD thesis, Brown University, 2001. In preparation.

[Koller and Parr, 1999] Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured MDPs. In *Proceedings IJCAI-99*, 1999.

[Munos and Moore, 1999] Rémi Munos and Andrew Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *Proceedings IJCAI-99*, 1999.

[Singh and Yee, 1994] Satinder P. Singh and Richard C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16:227–233, 1994.

[Somenzi, 1998] Fabio Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0*. Department of Electrical and Computer Engineering, University of Colorado at Boulder, 1998.

[St-Aubin *et al.*, 2000] Robert St-Aubin, Jesse Hoey, and Craig Boutilier. APRICODD: Approximate policy construction using decision diagrams. In *Proceedings NIPS-2000*, 2000.

[Tsitsiklis and Van Roy, 1996] John N. Tsitsiklis and Benjamin Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

[White and Eldeib, 1994] Chelsea White and Hany Eldeib. Markov decision processes with impricise transition probabilities. *Operations Research*, 42(4), 1994.

Symbolic Dynamic Programming for First-Order MDPs

Craig Boutilier
Dept. of Computer Science
University of Toronto
Toronto, ON, M5S 3H5
cebly@cs.toronto.edu

Ray Reiter
Dept. of Computer Science
University of Toronto
Toronto, ON. M5S 3H5
reiter@cs.toronto.edu

Bob Price
Department of Computer Science
University of British Columbia
Vancouver, BC, V6T 1Z4
bprice@cs.toronto.edu

Abstract

We present a dynamic programming approach for the solution of first-order Markov decisions processes. This technique uses an MDP whose dynamics is represented in a variant of the situation calculus allowing for stochastic actions. It produces a *logical description* of the optimal value function and policy by constructing a set of first-order formulae that *minimally* partition state space according to distinctions made by the value function and policy. This is achieved through the use of an operation known as *decision-theoretic regression*. In effect, our algorithm performs value iteration without explicit enumeration of either the state or action spaces of the MDP. This allows problems involving relational fluents and quantification to be solved without requiring explicit state space enumeration or conversion to propositional form.

1 Introduction

Markov decision processes (MDPs) have become the *de facto* standard model for decision-theoretic planning problems. However, classic dynamic programming algorithms for MDPs [Puterman, 1994] require explicit state and action enumeration. For example, the classical representation of a value function is as a table or vector associating a value with each system state; these are produced by iterating over the state space. Since state spaces grow exponentially with the number of domain features, the direct application of these models to AI planning problems is limited. As a consequence, much MDP research in AI has focussed on representations and algorithms that allow complex planning problems to be specified concisely and solved effectively. Techniques such as function approximation [Bertsekas and Tsitsiklis, 1996] and state aggregation [Boutilier *et al.*, 1999] have proven reasonably effective at solving MDPs with very large state spaces.

One such approach with a strong connection to classical planning is the *decision-theoretic regression (DTR)* model [Boutilier *et al.*, 2000a]. The state space of an MDP is characterized by a number of random variables (e.g., propositions) and the domain is specified using logical representations of actions that capture the regularity in the effects of actions. For instance, Bayesian networks, decision trees, algebraic decision diagrams (ADDs), and probabilistic extensions of STRIPS can all be used to concisely represent stochastic actions in MDPs. These representations are exploited in the construction of a logical representation of the optimal value function and policy, thereby obviating the need for explicit state space enumeration. This process can be viewed as automatic state space abstraction and has been able to solve fairly substantial problems. For instance, the SPUDD algorithm [Hoey *et al.*, 1999] has been used to solve MDPs with hundreds of millions of states optimally, producing logical descriptions of value functions that involve only hundreds of distinct values. This works suggests that very large MDPs, if described in a logical fashion, can often be solved optimally by exploiting the logical structure of the problem.

Unfortunately, existing DTR algorithms are all designed to work with *propositional* representations of MDPs, while many realistic planning domains are best represented in first-order terms, exploiting the existence of domain objects, relations over those objects, and the ability to express objectives and action effects using quantification. Existing DTR algorithms can only be applied to these problems by grounding or "propositionalizing" the domain.[1] Unfortunately such an approach is impractical: the number of propositions grows very quickly with the number of domain objects and relations, and even relatively simple domains can generate incredibly large numbers of propositions when grounded. The number of propositions has a dramatic impact on the complexity of these algorithms. Specifying and reasoning with intuitively simple domain properties involving quantification becomes problematic in a propositional setting. For instance, a simple objective such as $\exists x \phi(x)$ (e.g., we want some widget at Factory 1) becomes the unwieldy $\phi(c_1) \vee \ldots \vee \phi(c_n)$, where the c_i are (relevant) constants (e.g., widget-1 is at Factory 1, or ...). Thus grounding our domain description deprives one of the naturalness and expressive power of relational representations and quantification in specifying dynamics and objective functions. Finally, existing DTR algorithms require explicit action enumeration when performing dynamic programming, which is also problematic in first-order domains, since the number of ground actions also grows dramatically with domain size.

In this paper we address these difficulties by proposing

[1] This assumes a *finite domain*: if the domain is infinite, these algorithms cannot generally be made to work.

a decision-theoretic regression algorithm for solving *first-order MDPs (FOMDPs)*. We adopt the the representation for FOMDPs presented in [Reiter, 2001; Boutilier *et al.*, 2000b], in which stochastic actions and objective functions are specified using the situation calculus. We derive a version of value iteration [Bellman, 1957] that constructs first-order representations of value functions and policies by exploiting the logical structure of the MDP. The algorithm constructs a minimal partitioning of state space, represented by a set of first-order formulae, and associates values (or action choices) with each element of the partition.

As a consequence, our dynamic programming algorithm solves first-order MDPs without explicit state space *or* action enumeration, and without propositionalizing the domain. Furthermore, the technique we propose can be used to reason purely *symbolically* about value and optimal action choice. Our model can be viewed as providing a tight, seamless integration of classic knowledge representation techniques and reasoning methods with solution algorithms for MDPs.

This paper should be viewed as providing the theoretical foundations for first-order decision-theoretic regression. We are encouraged by the success of DTR methods for propositional MDPs, where it has been demonstrated that many MDPs have value functions and policies that can be represented very concisely using logical techniques. We have no doubt that the use of relations and quantification will ultimately enhance these methods tremendously.

We review MDPs in Section 2, and briefly describe our representation of FOMDPs in Section 3. We derive our symbolic dynamic programming technique in detail in Section 4 and discuss various implementation issues in Section 5. We conclude with a discussion of future directions.

2 Markov Decision Processes

We begin with the standard state-based formulation of MDPs. We assume that the domain of interest can be modeled as a fully-observable MDP [Bellman, 1957; Puterman, 1994] with a finite set of states $\mathcal{S}$ and actions $\mathcal{A}$. Actions induce stochastic state transitions, with $\Pr(s, a, t)$ denoting the probability with which state t is reached when action a is executed at state s. We also assume a real-valued reward function R, associating with each state s its immediate utility $R(s)$.[2]

A *stationary policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$ describes a particular course of action to be adopted by an agent, with $\pi(s)$ denoting the action to be taken in state s. The decision problem faced by the agent in an MDP is that of forming an *optimal policy* that maximizes expected total accumulated reward over an infinite horizon (i.e., the agent acts indefinitely). We compare policies by adopting *expected total discounted reward* as our optimality criterion, wherein future rewards are discounted at a rate $0 \leq \gamma < 1$, and the *value of a policy* π, denoted $V_\pi(s)$, is given by the expected total discounted reward accrued, that is, $E(\sum_{t=0}^{\infty} \gamma^t R(s^t)|\pi, s)$. Policy π is *optimal* if $V_\pi \geq V_{\pi'}$ for all $s \in \mathcal{S}$ and policies π'. The *optimal value function* V^* is the value of any optimal policy.

[2] We ignore actions costs for ease of exposition. These impose no additional complications on our model.

Value iteration [Bellman, 1957] is a simple iterative approximation algorithm for constructing optimal policies. It proceeds by constructing a series of n-stage-to-go *value functions* V^n. Setting $V^0 = R$, we recursively define n-stage-to-go *Q-functions*:

$$Q^n(a, s) = R(s) + \left\{ \gamma \sum_{t \in \mathcal{S}} \Pr(s, a, t) \cdot V^{n-1}(t) \right\} \quad (1)$$

and value functions:

$$V^n(s) = \max_a Q^n(a, s) \quad (2)$$

The Q-function $Q^n(a, s)$ denotes the expected value of performing action a at state s with n stages to go and acting optimally thereafter. The sequence of value functions V^n produced by value iteration converges linearly to V^*. For some finite n, the actions that maximize Eq. (2) form an optimal policy, and V^n approximates its value. We refer to Puterman [1994] for a discussion of stopping criteria.

The definition of a Q-function can be based on any value function. We define $Q^V(a, s)$ exactly as in Eq. (1), but with arbitrary value function V replacing V^{n-1} on the right-hand side. $Q^V(a, s)$ denotes the value of performing a at state s, then acting in such a way as to obtain value V subsequently.

3 First-Order Representation of MDPs

Most planning domains are specified in terms of a set of random variables, which jointly determine the state of the system. For example, the system state may be the assignment of truth values to a set of propositional variables. In addition, these variables may themselves be structured, built from various relations, functions, and domain objects, that naturally lend themselves to a first-order representation. Representing and solving MDPs under such circumstances is generally impractical using classic state-based transition matrices and dynamic programming algorithms. The difficulty lies in the need to explicitly enumerate state and action spaces. State spaces grow exponentially with the number of propositional variables need to characterize the domain. Furthermore, in a first-order domain, the number of induced propositional variables can grow dramatically with the number of domain objects of interest.[3] Moreover, we are often interested in solving planning problems with infinite domains.

Several representations for propositionally-factored MDPs have been proposed, including probabilistic variants of STRIPS and dynamic Bayes nets [Boutilier *et al.*, 1999]. First-order representations have also been proposed for MDPs, including those of Poole [1997], and Geffner and Bonet [1998]. In this paper we adopt the first-order, situation calculus MDP representation developed by Reiter [2001], and by Boutilier *et al.* [2000b] for use in the DTGolog framework. This model has several unique features that make dynamic programming techniques viable. We first review this representational language and methodology, and then show how stochastic actions can be represented in this framework. We also introduce some notation to ease the specification of MDPs.

[3] An n-ary relation over a domain of size d induces d^n atoms.

3.1 The Situation Calculus

The situation calculus [McCarthy, 1963] is a first-order language for axiomatizing dynamic worlds. In recent years, it has been considerably extended beyond the "classical" language to include processes, concurrency, time, etc., but in all cases, its basic ingredients consist of *actions*, *situations* and *fluents*.

Actions

Actions are first-order terms consisting of an action function symbol and its arguments. For example, the action of putting block b on the table might be denoted by the action term $putTbl(b)$.

Situations

A *situation* is a first-order term denoting a sequence of actions. These are represented using a binary function symbol do: $do(\alpha, s)$ denotes the sequence resulting from adding the action α to the sequence s. The special constant S_0 denotes the *initial situation*, namely the empty action sequence. Thus, $do(\alpha, s)$ is like LISP's $cons(\alpha, s)$ and S_0 is like LISP's ( ). In a blocks world, the situation term

$$do(stack(A, B), do(putTbl(B), do(stack(C, D), S_0)))$$

denotes the sequence of actions

$$[stack(C, D), putTbl(B), stack(A, B)].$$

Foundational axioms for situations are given in [Pirri and Reiter, 1999].

Fluents

Relations whose truth values vary from state to state are called *fluents*, and are denoted by predicate symbols whose last argument is a situation term. For example, $BIn(b, Paris, s)$ is a relational fluent meaning that in that state reached by performing the action sequence s, box b is in Paris.

Axiomatizing a Domain Theory

A domain theory is axiomatized in the situation calculus with four classes of axioms [Pirri and Reiter, 1999]:

1. **Action precondition axioms**: There is one axiom for each action function $A(\vec{x})$, with syntactic form

   $$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$$

   Here, $\Pi_A(\vec{x}, s)$ is a formula with free variables among $\vec{x}, s$. These characterize the preconditions of action A.
2. **Successor state axioms**: There is one such axiom for each fluent $F(\vec{x}, s)$, with syntactic form

   $$F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s),$$

   where $\Phi_F(\vec{x}, a, s)$ is a formula with free variables among $a, s, \vec{x}$. These characterize the truth values of the fluent F in the next situation $do(a, s)$ in terms of the current situation s, and they embody a solution to the frame problem for deterministic actions [Reiter, 1991].
3. **Unique names axioms for actions**: These state that the actions of the domain are pairwise unequal.
4. **Initial database**: This is a set of first-order sentences whose only situation term is S_0 and it specifies the initial state of the domain. The initial database will play no role in this paper.

Regression in the Situation Calculus

The regression of a formula ψ through an action a is a formula ψ' that holds prior to a being performed iff ψ holds after a. Successor state axioms support regression in a natural way. Suppose that fluent F's successor state axiom is $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$. We inductively define the regression of a formula whose situation arguments all have the form $do(a, s)$ as follows:

$$Regr(F(\vec{x}, do(a, s))) = \Phi_F(\vec{x}, a, s)$$
$$Regr(\neg\psi) = \neg Regr(\psi)$$
$$Regr(\psi_1 \wedge \psi_2) = Regr(\psi_1) \wedge Regr(\psi_2)$$
$$Regr((\exists x)\psi) = (\exists x) Regr(\psi)$$

3.2 Stochastic Actions and the Situation calculus

For the purposes of representing probabilistic uncertainty, the above ontology and axiomatization for the situation calculus might appear to be inadequate, because all actions must be deterministic. One can see this requirement most clearly in the syntactic form of successor state axioms where a fluent's truth value in the next situation is uniquely determined by the current situation; thus, the next state is uniquely determined by the present state and the action performed. How then can stochastic actions be represented in the situation calculus? The trick is to decompose stochastic actions into deterministic primitives under nature's control—she chooses the deterministic action that actually gets executed, with some specified probability, when an agent performs a stochastic action. We then formulate situation calculus domain axioms using these deterministic choices [Bacchus *et al.*, 1995; Reiter, 2001; Boutilier *et al.*, 2000b].

We illustrate this approach with a simple example in a logistics domain consisting of cities, trucks, and boxes: boxes can be loaded onto and unloaded from trucks, and trucks can be driven between cities.

Nature's Choices for Stochastic Actions: For each stochastic action we must specify the deterministic choices available to nature. For instance, the stochastic *load* action can succeed (denoted by *loadS*) or fail (*loadF*):

$$choice(load(b, t), a) \equiv a = loadS(b, t) \vee a = loadF(b, t)$$

Similarly, the stochastic *unload* and *drive* actions also decompose into successful or unsuccessful alternatives chosen by nature with known probabilities.

$$choice(unload(b, t), a) \equiv a = unloadS(b, t) \vee a = unloadF(b, t)$$
$$choice(drive(t, c), a) \equiv a = driveS(t, c) \vee a = driveF(t, c)$$

Probabilities for Nature's Choices: For each of nature's choices $n(\vec{x})$ associated with action $A(\vec{x})$, we specify the probability $prob(n(\vec{x}), A(\vec{x}), s)$ with which it is chosen, given that $A(\vec{x})$ was performed in situation s:

$$prob(loadS(b, t), load(b, t), s) = 0.99$$
$$prob(loadF(b, t), load(b, t), s) = 0.01$$
$$prob(unloadS(b, t), unload(b, t), s) = p \equiv$$
$$Rain(s) \wedge p = 0.7 \vee \neg Rain(s) \wedge p = 0.9$$
$$prob(unloadF(b, t), unload(b, t), s) =$$

$$1 - \mathit{prob}(\mathit{unloadS}(b,t), \mathit{unload}(b,t), s)$$

$$\mathit{prob}(\mathit{driveS}(t,c), \mathit{drive}(t,c), s) = 0.99$$

$$\mathit{prob}(\mathit{driveF}(t,c), \mathit{drive}(t,c), s) = 0.01$$

Here we see that unloading is less likely to succeed when it is raining.

Action Preconditions for Deterministic Actions:

$$\mathit{Poss}(\mathit{loadS}(b,t), s) \equiv (\exists c).\mathit{BIn}(b,c,s) \wedge \mathit{TIn}(t,c,s)$$

$$\mathit{Poss}(\mathit{loadF}(b,t), s) \equiv (\exists c).\mathit{BIn}(b,c,s) \wedge \mathit{TIn}(t,c,s)$$

$$\mathit{Poss}(\mathit{unloadS}(b,t), s) \equiv \mathit{On}(b,t,s)$$

$$\mathit{Poss}(\mathit{unloadF}(b,t), s) \equiv \mathit{On}(b,t,s)$$

$$\mathit{Poss}(\mathit{driveS}(t,c), s) \equiv \mathit{true}$$

$$\mathit{Poss}(\mathit{driveF}(t,c), s) \equiv \mathit{true}$$

Nature's choices $n_j(\vec{x})$ for action $A(\vec{x})$ need not have common preconditions, but often they do, as above.

Successor State Axioms:

$$\mathit{BIn}(b,c,\mathit{do}(a,s)) \equiv (\exists t)[\mathit{TIn}(t,c,s) \wedge a = \mathit{unloadS}(b,t)] \vee \mathit{BIn}(b,c,s) \wedge \neg(\exists t) a = \mathit{loadS}(b,t)$$

$$\mathit{TIn}(t,c,\mathit{do}(a,s)) \equiv a = \mathit{driveS}(t,c) \vee \mathit{TIn}(t,c) \wedge \neg(\exists c') a = \mathit{driveS}(t,c')$$

$$\mathit{On}(b,t,\mathit{do}(a,s)) \equiv a = \mathit{loadS}(b,t) \vee \mathit{On}(b,t,s) \wedge a \neq \mathit{unloadS}(b,t)$$

$$\mathit{Rain}(\mathit{do}(a,s)) \equiv \mathit{Rain}(s)$$

There are two important points to note about this example:

1. By virtue of decomposing stochastic actions into deterministic primitives under nature's control, we get perfectly conventional situation calculus action precondition and successor state axioms *that do not refer to stochastic actions.* Stochastic actions have a status different from deterministic actions, and cannot participate in situation terms.[4]
2. *Nowhere do these axioms restrict the domain of discourse to some prespecified set of trucks, boxes, or cities.* There are even models of these axioms with infinitely many—even uncountably many—individuals. If one were to solve an MDP for which this axiomatization is valid, one would obtain, in fact, a solution that applies to an entire class of MDPs with arbitrary domains of trucks, boxes and cities.

3.3 Some Additional Notation

In what follows we use the notion of a *state formula*, $\psi(\vec{x}, s)$, whose only free variables are non-situation variables $\vec{x}$ and a situation variable s. Intuitively, a state formula refers only to properties of the situation s. A set of state formulae $\{\psi_i(\vec{x}, s)\}$ *partitions* state space iff $\models (\forall \vec{x}, s).\psi_i(\vec{x}, s) \supset \neg\psi_j(\vec{x}, s)$, for all i, $j \neq i$, and $\models (\forall \vec{x}, s).\bigvee_i \psi_i(\vec{x}, s)$.

[4]Note that when nature's choices for a specific action do not have identical preconditions, care must be taken in the axiomatization to ensure the probabilities sum to one in every situation.

The Case Notation

To simplify the presentation, we introduce the notation

$$t = \mathit{case}[\phi_1, t_1; \cdots; \phi_n, t_n]$$

as an abbreviation for the formula

$$\bigvee_{i \leq n} \{\phi_i \wedge t = t_i\}$$

where the ϕ_i are state formulae and the t_i are terms. We sometimes write this $\mathit{case}[\phi_i, t_i]$. Often the t_i will be constants and the ϕ_i will partition state space. We introduce the following operators on case statements (whose use will be important in the next section):

$$\mathit{case}[\phi_i, t_i : i \leq n] \otimes \mathit{case}[\psi_j, v_j : j \leq m] = \mathit{case}[\phi_i \wedge \psi_j, t_i \cdot v_j : i \leq n, j \leq m]$$

$$\mathit{case}[\phi_i, t_i : i \leq n] \oplus \mathit{case}[\psi_j, v_j : j \leq m] = \mathit{case}[\phi_i \wedge \psi_j, t_i + v_j : i \leq n, j \leq m]$$

$$\mathit{case}[\phi_i, t_i : i \leq n] \ominus \mathit{case}[\psi_j, v_j : j \leq m] = \mathit{case}[\phi_i \wedge \psi_j, t_i - v_j : i \leq n, j \leq m]$$

$$\mathit{case}[\phi_i, t_i : i \leq n] \cup \mathit{case}[\psi_j, v_j : j \leq m] = \mathit{case}[\phi_1, t_1; \cdots \phi_n, t_n; \psi_1, v_1; \cdots \psi_m, v_m]$$

Representing Probabilities with Case Notation

Let $A(\vec{x})$ be a stochastic action type with possible outcomes $n_1(\vec{x}), \cdots, n_k(\vec{x})$. We assume the probabilities of these outcomes are specified using case notation. Specifically, the choice probabilities for $n_j(\vec{x})$ are given as:

$$\mathit{prob}(n_j(\vec{x}), A(\vec{x}), s) = \mathit{case}[\phi_1^j(\vec{x}, s), p_1^j; \cdots; \phi_n^j(\vec{x}, s), p_n^j],$$

where the ϕ_i^j partition state space, and p_i^j is the probability of choice $n_i(\vec{x})$ being realized under condition $\phi_i^j(\vec{x}, s)$ when the agent executes stochastic action $A(\vec{x})$.

Our *unload* stochastic action above is represented in case notation as:

$$\mathit{prob}(\mathit{unloadS}(b,t), \mathit{unload}(b,t), s) = \mathit{case}[\mathit{Rain}(s), 0.7; \neg\mathit{Rain}(s), 0.9]$$

$$\mathit{prob}(\mathit{unloadF}(b,t), \mathit{load}(b,t), s) = \mathit{case}[\mathit{Rain}(s), 0.3; \neg\mathit{Rain}(s), 0.1].$$

Notice that when the probability of nature's choice is situation-independent, (e.g., as in *loadS*), then only a single "case" is present (e.g., $case[\mathit{true}, 0.99]$).

Specifying Rewards and Values with Case Notation

An MDP optimization theory contains axioms specifying the reward function. In their simplest form, reward axioms use the function $R(s)$ to assert costs and rewards as a function of the action taken, properties of the current situation, or both (note that the action taken can be recovered from the situation term). In what follows, we assume a simple "state-based" reward model in which only relational fluents determine reward, and we assume that this reward function is specified using case notation:

$$R(s) = \mathit{case}[\xi_1(s), r_1; \cdots; \xi_m(s), r_m],$$

where the $\xi_i(s)$ partition state space. For example, rewarding the presence of *some* box in Paris can be specified using

$$R(s) = \mathit{case}[(\exists b)\mathit{BIn}(b, Paris, s), 10; \neg(\exists b)\mathit{BIn}(b, Paris, s), 0]$$

The restriction to state-based reward is simply to keep the exposition simple. Action costs are easily modeled and are used in our prototype implementation.

We also use the case notation to represent value functions in a similar fashion, concisely writing V in the form

$$V(s) = \mathit{case}[\beta_1(s), v_1; \cdots \beta_n(s), v_n].$$

This use of case statements can be viewed as embodying a form of state space *abstraction*: rather than assigning values on a state-by-state basis, we distinguish states according to the conditions β_i. Those states satisfying β_i can be treated as an *abstract state*. In this way, we can often represent value functions (and policies and Q-functions similarly) without state enumeration, exploiting the logical structure of the function. This is similar to the abstraction models discussed in [Boutilier *et al.*, 1999], but with the ability to partition state space using first-order formulae.

4 Dynamic Programming with FOMDPs

Logical representations for MDPs provide natural and compact specifications of planning domains, obviating the need for explicit state space enumeration. Logical descriptions exploiting regularities in value functions and policies can also be very compact. Solving an FOMDP can be made much more efficient if the logical structure of value functions can be discovered through inference using the the logical MDP specification, with expected value computations performed once per abstract state instead of once per state. Thus a dynamic programming algorithm that works directly with symbolic representations of value functions offers great potential computational benefit. In this section, we generalize the notion of decision-theoretic regression from propositional MDPs to FOMDPs, and construct a *first-order value iteration* algorithm.

4.1 First-Order Decision-Theoretic Regression

Suppose we are given a value function V. The *first-order decision theoretic regression (FODTR)* of V through action type $A(\vec{x})$ is a logical description of the Q-function $Q^V(A(\vec{x}), s)$. In other words, given a set of abstract states corresponding to regions of state space where V is constant, we wish to produce a corresponding abstraction for $Q^V(A(\vec{x}), s)$. This is analogous to classical goal regression, the key differences being that action $A(\vec{x})$ is stochastic.

Let $A(\vec{x})$ be a stochastic action with corresponding nature's choices $n_j(\vec{x}), j \leq k$. Ignoring preconditions momentarily, $Q^V(A(\vec{x}), s)$ is defined classically as

$$Q^V(A(\vec{x}), s) = R(s) + \gamma \cdot \{\textstyle\sum_{t \in \mathcal{S}} Pr(s, A(\vec{x}), t) \cdot V(t)\}$$

Since different successor states arise only through different nature's choices, the situation calculus analog of this is:

$$Q^V(A(\vec{x}), s) = R(s) + \gamma \cdot \sum_j \mathit{prob}(n_j(\vec{x}), A(\vec{x}), s) \cdot V(\mathit{do}(n_j(\vec{x}), s)) \quad (3)$$

As described earlier, we assume that the functions $R(s)$, $prob(n, A, s)$ and $V(s)$ are all described with case statements. Respectively denote these by $\mathit{rCase}(s)$, $\mathit{pCase}(n, s)$ and $\mathit{vCase}(s)$. Then after substituting these case expressions into Eq. (3) and appealing to the case addition and multiplication operators of Section 3.3, we obtain

$$Q^V(A(\vec{x}), s) = \mathit{rCase}(s) \oplus \gamma \cdot [\oplus_j \{\mathit{pCase}(n_j(\vec{x}), s) \otimes \mathit{vCase}(\mathit{do}(n_j(\vec{x}), s))\}]$$

The only problem with this expression is that the formula $\mathit{vCase}(\mathit{do}(n_j(\vec{x}), s))$ refers not to the current situation s, but to the future situation $\mathit{do}(n_j(\vec{x}), s))$, but this is easily remedied with regression:

$$Q^V(A(\vec{x}), s) = \mathit{rCase}(s) \oplus \gamma \cdot [\oplus_j \mathit{pCase}(n_j(\vec{x}), s) \otimes \mathit{Regr}(\mathit{vCase}(\mathit{do}(n_j(\vec{x}), s)))]$$

We emphasize the critical nature of this step. The representational methodology we adopt—treating stochastic actions using *deterministic* nature's choices—allows us to apply regression directly to derive *properties of the pre-action state that determine the value-relevant properties of the post-action state*. Specifically, classical regression can be applied *directly* to the case statement $\mathit{vCase}(\mathit{do}(n_j(\vec{x}), s))$ because the $n_j(\vec{x})$ are deterministic.

Because sums and products of case statements are also case statements, the above expression for $Q^V(A(\vec{x}), s)$ is a case statement, say $\mathit{case}[\alpha_i(\vec{x}, s), q_i]$, that characterizes the Q-function for action $A(\vec{x})$ with respect to V. Thus from a logical description of V we can derive one for Q. Conceptually, this can be viewed as transforming the abstraction of state space suitable for V into one suitable for Q. It is not hard to show that if the state formulae in V's case statement partition the state space, then so do the α_i defining Q. This is key to avoiding state and action enumeration in dynamic programming.

The above derivation ignores action preconditions. To handle preconditions, $Q^V(A(\vec{x}), s)$ can no longer be treated as a function, but must be represented by a *relation* $Q^V(A(\vec{x}), q, s)$, meaning that A's Q-value in s is q. This relation holds only if $\mathit{Poss}(n_i(\vec{x}), s)$ holds for at least one of A's choices n_i; otherwise the Q-value is undefined:

$$Q^V(A(\vec{x}), q, s) \equiv [\textstyle\bigvee_i \mathit{Poss}(n_i(\vec{x}), s)] \wedge q = \mathit{case}[\alpha_i(\vec{x}, s), q_i]$$

Since $\bigvee_i \mathit{Poss}(n_i(\vec{x}), s)$ can be distributed into the case statement (by conjoining it with the α_i), the result is again a case statement for the Q-relation.

As an example consider value function V^0:

$$V(s) = \mathit{case}[\exists b.\mathit{BIn}(b, \mathit{Rome}, s), 10 \; ; \; \neg\exists t.\mathit{BIn}(b, \mathit{Rome}, s), 0]$$

That is, if some box b is in *Rome*, value is 10; otherwise value is 0. Suppose that reward R is identical to V^0 and our discount rate is 0.9. We use the $\mathit{unload}(b, t)$ action, described above, to illustrate FODTR. The regression of V^0 through $\mathit{unload}(b, t)$ results in a case statement (after simplification), denoting $Q^1(\mathit{unload}(b, t), q, s)$ with four elements:

$$\alpha_1(b, t, s) \equiv \exists b'\mathit{BIn}(b', \mathit{Rome}, s)$$

$$\alpha_2(b, t, s) \equiv \mathit{Rain}(s) \wedge \mathit{TIn}(t, \mathit{Rome}, s) \wedge \mathit{On}(b, t, s) \wedge \neg\exists b'\mathit{BIn}(b', \mathit{Rome}, s)$$

$$\alpha_3(b, t, s) \equiv \neg\mathit{Rain}(s) \wedge \mathit{TIn}(t, \mathit{Rome}, s) \wedge$$

$$On(b,t,s) \wedge \neg\exists b' BIn(b', Rome, s)$$
$$\alpha_4(b,t,s) \equiv (\neg TIn(t, Rome, s) \vee \neg On(b,t,s)) \wedge \exists b' BIn(b', Rome, s)$$

and the associated Q-values: $q_1 = 19$; $q_2 = 6.3$; $q_3 = 8.1$; $q_4 = 0$. Before simplification, the case statement consisted of 8 formulae, two of which were inconsistent and two pairs of which had identical Q-values.

An important property of FODTR is that it not only produces an abstraction of state space to describe $Q^V(A(\vec{x}), q, s)$, it also abstracts the action space as well. With a small number of logical formulae, it captures the Q-values $Q(a, q, s)$ for each situation s and *each instantiation* a of $A(\vec{x})$. While state space abstraction has been explored in the context of decision-theoretic regression for propositional representations of MDPs, little work has focused on abstracting the action space in this way.

Finally, although our example works with specific numerical values in the case statements, purely *symbolic* descriptions of value can also be reasoned with in this way. For example, if the Q-value of action *drive*(t, c) depends on the weight of truck t in the current situation, the value term in a case statement can be made to depend on this property of the situation (i.e., *weight*(t, s)). This can prove especially useful for reasoning with continuous (or hybrid) state and action spaces.

4.2 Symbolic Dynamic Programming

Value iteration consists of setting $V^0 = R$ and repeatedly applying Eq. (1) and Eq. (2) until a suitable termination condition is met. Since R is described symbolically and FODTR can be used to implement Eq. (1) logically, we need only derive a "logical implementation" of Eq. (2) in order to have a form of dynamic programming that can compute optimal policies for FOMDPs without explicit state or action enumeration (together with a method for termination testing and policy extraction).

In what follows, we assume that all values occurring in the case statements for $Q(A, v, s)$ are numerical constants, which means that the case statements for $prob(n, A, s)$, $V(s)$ and $R(s)$ all have this property.

Suppose we have computed n-stage-to-go Q-relations $Q(A(\vec{x}), v, s)$, one for each action type A, of the form $case[\alpha_i^A(\vec{x}, s), q_i^A]$, where the q_i^A are numerical constants. Letting $V(s)$ denote the n-stage-to-go value function, Eq. (2) can be written

$$V(s) = v \equiv (\exists a).Q(a, v, s) \wedge (\forall b) Q(b, w, s) \supset w \leq v \quad (4)$$

We assume that *some* stochastic action (e.g., a deterministic no-op) is executable in every situation, so that $V(s)$ will be a function. (If not, we can easily define it as a relation.) We now derive a series of expressions for the r.h.s. of this equivalence. Assuming domain closure for action types (i.e., all actions a are instances of some $A_i(\vec{x}_i)$, we have

$$V(s) = v \equiv [\bigvee_i (\exists \vec{x}_i) Q(A_i(\vec{x}_i), v, s)] \wedge \bigwedge_j (\forall \vec{y}_j, v').Q(A_j(\vec{y}_j), v', s) \supset v' \leq v$$

To minimize notational clutter, represent this generically by

$$V(s) = v \equiv [\bigvee_A (\exists \vec{x}) Q(A(\vec{x}), v, s)] \wedge \bigwedge_B (\forall \vec{y}, v').Q(B(\vec{y}), v', s) \supset v' \leq v$$

We are supposing that we have already determined the Q-values for each action type A, in the form of a case statement:

$$Q(A(\vec{x}), q, s) \equiv q = case[\alpha_i^A(\vec{x}, s), q_i^A] \quad (5)$$

Substitute Eq. (5) into the previous expression to get

$$V(s) = v \equiv \left[\bigvee_A (\exists \vec{x}) v = case[\alpha_i^A(\vec{x}, s), q_i^A]\right] \wedge \bigwedge_B (\forall \vec{y}, v').Q(B(\vec{y}), v', s) \supset v' \leq v$$

Since the q_i^A are constants, we can distribute the existential quantifiers into the case expression:

$$V(s) = v \equiv \left[\bigvee_A v = case[(\exists \vec{x})\alpha_i^A(\vec{x}, s), q_i^A]\right] \wedge \bigwedge_B (\forall \vec{y}, v').Q(B(\vec{y}), v', s) \supset v' \leq v$$

Writing $(\exists \vec{x})\alpha_i^A(\vec{x}, s)$ as $\gamma_i^A(s)$, and recalling the definition of the case union operator $\cup$ of Section 3.3, we have

$$V(s) = v \equiv v = \left[\bigcup_A case[\gamma_i^A(s), q_i^A]\right] \wedge \bigwedge_B (\forall \vec{y}, v').Q(B(\vec{y}), v', s) \supset v' \leq v$$

Suppose $\bigcup_A case[\gamma_i^A(s), q_i^A]$ has the form $case[\gamma_i(s), V_i : i \leq k]$. Therefore,

$$V(s) = v \equiv [\bigvee_{i=1}^{k} \gamma_i(s) \wedge v = V_i] \wedge \bigwedge_B (\forall \vec{y}, v').Q(B(\vec{y}), v', s) \supset v' \leq v$$

This simplifies to

$$V(s) = v \equiv \bigvee_{i=1}^{k} \gamma_i(s) \wedge \bigwedge_B (\forall \vec{y}, v')[Q(B(\vec{y}), v', s) \supset v' \leq V_i] \wedge v = V_i$$

Recalling the definition of the case notation, we get

$$V(s) = v \equiv v = case[\gamma_i(s) \wedge \bigwedge_B (\forall \vec{y}, v').Q(B(\vec{y}), v', s) \supset v' \leq V_i, V_i : i \leq k]$$

The only remaining task is to characterize the expressions $Q(B(\vec{y}), v', s) \supset v' \leq V_i$ in terms of the case statement for $Q(B(\vec{y}), v', s)$. Suppose this case statement is:

$$Q(B(\vec{y}), v', s) \equiv v' = case[\beta_j^B(\vec{y}, s), q_j^B]$$

Then it is easy to show that

$$Q(B(\vec{y}), v', s) \supset v' \leq V_i \equiv \bigwedge_j [\beta_j^B(\vec{y}, s) \supset q_j^B \leq V_i]$$

Substituting this last expression for $Q(B(\vec{y}), v', s) \supset v' \leq V_i$ in the above expression for $V(s)$ gives us

$$V(s) = v \equiv v = case[\gamma_i(s) \wedge \bigwedge_B (\forall \vec{y}). \bigwedge_j [\beta_j^B(\vec{y}, s) \supset q_j^B \leq V_i], V_i : i \leq k]$$

Next, because the q_j^B and V_i are numerical constants, we can distribute the universal quantifier as an existential quantifier in the antecedent of the implications, to get

$$V(s) = v \equiv v = case[\gamma_i(s) \wedge \bigwedge_B \bigwedge_j [(\exists \vec{y})\beta_j(\vec{y}, s) \supset q_j^B \leq V_i], V_i : i \leq k]$$

Next, recalling how the γ_i were introduced by unioning the case expressions for all the Q-values, we get

$$V(s) = v \equiv v = case[\gamma_i(s) \wedge \bigwedge_j [\gamma_j(s) \supset V_j \leq V_i], V_i : i \leq k]$$

Finally, we can again exploit the fact that the Vs are numerical constants (as opposed to symbolic terms), and therefore

can be compared. This allows us to write our final expression for V:

$$V(s) = \mathit{case} \begin{bmatrix} \gamma_1(s) \wedge \bigwedge_{\{i|V_i > V_1\}} \neg\gamma_i(s) & V_1 \\ \vdots & \\ \gamma_k(s) \wedge \bigwedge_{\{i|V_i > V_k\}} \neg\gamma_i(s) & V_k \end{bmatrix}$$

If we modify the definition of the $\cup$ operator so that it sorts the rows according to their V values, and merges rows with identical V values, we get the pleasing expression

$$V(s) = \mathit{case} \begin{bmatrix} \gamma_1(s) & V_1 \\ \gamma_2(s) \wedge \neg\gamma_1(s) & V_2 \\ \vdots & \\ \gamma_k(s) \wedge \neg\gamma_1(s) \wedge \neg\gamma_2(s) \wedge \cdots \wedge \neg\gamma_{k-1}(s) & V_k \end{bmatrix} \quad (6)$$

This determines a simple case statement that completely defines the value function $V^n(s)$ given the logical description of the relations $Q^n(A(\vec{x}), v, s)$. Together with the FODTR algorithm for producing Q-relations, this provides the means to construct the sequence of value functions that characterize value iteration in a purely symbolic fashion, eliminating the need for state and action enumeration. It is not hard to show that the case conditions defining V^n partition state space.

Finally, notice that we obtained the case expression (6) by a sequence of equivalence-preserving transformations from the definition (3) of the Q-function (suitably modified to accommodate action preconditions), and the definition (4) of the value function. Therefore, we have:

Theorem 1 *The case expression (6) is a correct representation for $V(s)$ relative to the specifications (3) and (4) for the Q-function and value function respectively.*

With these pieces in place, we can summarize first-order value iteration as follows: given as input a first-order representation of $R(s)$ (a case statement) and our action model, we set $V^0(s) = R(s)$, $n = 1$ and perform the following steps until termination:

1. For each action type $A(\vec{x})$ compute the case representation of $Q^n(A(\vec{x}), q, s)$ (using $V^{n-1}(s)$ as in Eq. (3)).
2. Compute the case representation of $V^n(s)$ (using the $Q^n(A(\vec{x}), q, s)$ as in Eq. (6)).
3. Increment n.

Termination of first-order value iteration is straightforward. Given the case statements C^n and C^{n-1} for value functions V^n and V^{n-1}, we form $C^n \ominus C^{n-1}$ and simplify the resulting case statement by removal of any inconsistent elements. If each case has a value term less that specified threshold ε, value iteration terminates. Extraction of an optimal policy is also straightforward: one simply needs to extract the maximizing actions from the set of Q-functions derived from the optimal value function. The optimal policy will thus be represented symbolically with a case statement.

4.3 An Illustration

To give a flavor of the form of first-order value functions, consider an example where the reward function is given by three statements:

$$(\exists b).BIn(b, \mathit{Paris}, s) \wedge \mathit{TypeA}(b); r = 10$$
$$(\exists b).BIn(b, \mathit{Paris}, s) \wedge \neg\mathit{TypeA}(b); r = 5$$
$$\neg(\exists b)BIn(b, \mathit{Paris}, s); r = 0$$

That is, we want a box of Type A in Paris, but will accept a box of another type if a Type A box is unavailable. Actions include the load, unload, and drive actions described above. We include action costs: the action *unload*(b, t) has cost 4, *load*(b, t) has cost 1, and *drive*(t, c) has cost 3. The optimal one-stage policy chooses only unloading or no-op (since with only one stage to go, driving and loading have no value). Our algorithm derives the following conditions for *unload*(b, t) to be executed:

$$\mathit{On}(b, t, s) \wedge \mathit{TIn}(t, \mathit{Paris}, s) \wedge$$
$$[\neg(\exists b')\mathit{BIn}(b', \mathit{Paris}, s) \vee$$
$$\mathit{TypeA}(b) \wedge \neg\mathit{Rain}(s) \wedge$$
$$\neg(\exists b').\mathit{BIn}(b', \mathit{Paris}, s) \wedge \neg\mathit{TypeA}(b')]$$

Thus a box b is unloaded if there is a box on some truck in Paris, and there is no box currently in Paris, or b is a Type A box and it's not raining, and there's no Type A box in Paris. No-op is executed if the negation of the condition above holds (since for a one-step backup there is no value yet discovered for driving or loading). It is important to note that this partitioning remains fixed (as does the partitioning for the resultant value function) regardless of the number of domain objects and extraneous relations in the problem description. Thus we get stronger abstraction than would be possible using a propositionalized version of the problem. Also note that this describes the conditions under which one performs any *instance* of the unload action. In this way our algorithm allows for action abstraction, allowing one to produce value functions and policies without explicit enumeration of action instances.

5 A (Very) Preliminary Implementation

We have implemented (in Prolog) the basic Bellman backup operator (i.e., single iterations of one-step value iteration) defined by Eq (6). The implementation is based entirely on a rewrite interpreter that applies programmer specified rewrite rules to situation calculus formulae until no further rewrites are possible. The program first computes the case statements for the Q-values for all the stochastic actions. Next, from these it computes the $\langle \gamma_i(s), V_i \rangle$ pairs required by the case statement (6), and finally, the case statement of (6) itself. Throughout, logical simplification is applied (also specified by rewrite rules) to all subformulas of the current formula.

From a practical point of view, the key component in efficiently implementing first-order DTR is logical simplification to ensure manageable formulae describing the partitions. Our current implementation performs only the most rudimentary logical simplification and does not always produce concise descriptions of the cases within partitions. Neither can it eliminate all inconsistent partitions. The main reason for these limitations is that the current implementation lacks a first-order theorem-prover. For the example MDPs we have looked at,

sophisticated theorem-proving appears not to be necessary, but simple-minded simplification rules that don't know very much about quantifiers are simply too weak.

We ran value iteration to termination under our implementation using the reward function that gives a reward of 10 for having any box in Paris, and zero reward otherwise (for simplicity, it is treated as a terminal reward, and is received only once). Because our simplifier did not include a theorem-prover, some of the intermediate computations were hand-edited to further simplify the resulting expressions. We obtained the following optimal value function:

$$\exists b \mathit{BIn}(\mathit{Paris}, b, s) : 10$$
$$\neg \mathit{Rain}(s) \wedge \exists b, t(\mathit{On}(b, t, s) \wedge \mathit{TIn}(t, Paris, s)) \wedge \neg \exists b \mathit{BIn}(\mathit{Paris}, b, s) : 5.56$$
$$\mathit{Rain}(s) \wedge \exists b, t(\mathit{On}(b, t, s) \wedge \mathit{TIn}(t, Paris, s)) \wedge \neg \exists b. \mathit{BIn}(\mathit{Paris}, b, s) : 4.29$$
$$\neg \mathit{Rain}(s) \wedge \exists b, t \mathit{On}(b, t, s) \wedge \neg \exists b \mathit{BIn}(\mathit{Paris}, b, s) \wedge \neg \exists b, t(\mathit{On}(b, t, s) \wedge \mathit{TIn}(t, Paris, s)) : 2.53$$
$$\neg \mathit{Rain}(s) \wedge \exists b, t, c(\mathit{BIn}(c, s) \wedge \mathit{TIn}(c, s)) \wedge \neg \exists b, t \mathit{On}(b, t, s) \wedge \neg \exists b \mathit{BIn}(\mathit{Paris}, b, s) : 1.52$$
$$\mathit{Rain}(s) \wedge \exists b, t \mathit{On}(b, t, s) \wedge \neg \exists b, t(\mathit{On}(b, t, s) \wedge \mathit{TIn}(t, Paris, s)) \wedge \neg \exists b \mathit{BIn}(\mathit{Paris}, b, s) : 1.26$$
$$\neg \exists b \mathit{BIn}(\mathit{Paris}, b, s) \wedge \neg \exists b, t \mathit{On}(b, t, s) \wedge [\mathit{Rain}(s) \vee \neg \exists b, t, c(\mathit{BIn}(c, s) \wedge \mathit{TIn}(c, s)) : 0.0$$

We emphasize again that this value function applies no matter how many domain objects there are.

Our algorithm is not competitive with state of the art propositional MDP solvers, largely because solvers such as SPUDD [Hoey *et al.*, 1999] use very efficient implementations of logical reasoning software. We are currently developing a version of the FODTR algorithm that uses a first-order theorem-prover to enhance its performance. Of course, at another level, one can argue that propositional MDP solvers cannot even get off the ground when (even trivial) planning problems have a large number of domain objects.

An important issue we hope to address in the near future is the use of hybrid representations of MDPs and value functions that allow one to adopt efficient data structures like ADDs or decision trees, but instantiate these structures with first-order formulae. This would allow the expressive power of our first-order model, but restrict the syntactic form of formulae somewhat so that simplification and consistency checking could be implemented more effectively for "typical" problem instances.

6 Concluding Remarks

We have described the first approach for solving MDPs specified in first-order logic by dynamic programming. By the careful integration of sophisticated KR methods with classic MDP algorithms, we have developed a framework in which MDPs can be specified concisely and naturally and solved without explicit state and action enumeration. Indeed, nothing in our model prevents its direct application to infinite domains. Furthermore, it permits the symbolic representation of value functions and policies.

A number of interesting directions remain to be explored. As mentioned, the practicality of this approach depends on the use of sophisticated simplification methods. We are currently incorporating several of these into our implementation. Other dynamic programming algorithms (e.g., modified policy iteration) can be implemented directly within our framework. Approximation methods based on merging partitions with similar values can also be applied with ease. Finally, the investigation of symbolic dynamic programming to continuous and hybrid domains offers exciting possibilities.

Acknowledgements: This research was supported by NSERC and IRIS Project BAC "Dealing with Actions." Thanks to the referees for their helpful suggestions on the presentation of this paper.

References

[Bacchus *et al.*, 1995] F. Bacchus, J. Y. Halpern, and H. J. Levesque. Reasoning about noisy sensors in the situation calculus. *IJCAI-95*, 1933–1940, Montreal, 1995.

[Bellman, 1957] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[Bertsekas and Tsitsiklis, 1996] D. P. Bertsekas and J.. N. Tsitsiklis. *Neuro-dynamic Programming*. Athena, Belmont, MA, 1996.

[Boutilier *et al.*, 1999] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *J. Art. Intel. Res.*, 11:1–94, 1999.

[Boutilier *et al.*, 2000a] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Art. Intel.*, 121:49–107, 2000.

[Boutilier *et al.*, 2000b] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. *AAAI-2000*, 355–362, Austin, TX, 2000.

[Geffner and Bonet, 1998] H. Geffner and B. Bonet. High-level planning and control with incomplete information using POMDPs. *Fall AAAI Symp. on Cognitive Robotics*, Orlando, FL, 1998.

[Hoey *et al.*, 1999] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. *UAI-99*, 279–288, Stockholm, 1999.

[McCarthy, 1963] J. McCarthy. Situations, actions and causal laws. Tech. report, Stanford Univ., 1963. Repr. *Semantic Information Processing* (M. Minsky ed.), MIT Press, Cambridge, 1968, 410-417.

[Pirri and Reiter, 1999] F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. *JACM*, 46(3):261–325, 1999.

[Poole, 1997] D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Art. Intel.*, 94(1–2):7–56, 1997.

[Puterman, 1994] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[Reiter, 1991] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, ed., *Artificial Intelligence and Mathematical Theory of Computation (Papers in Honor of John McCarthy)*, 359–380. Academic Press, 1991.

[Reiter, 2001] R. Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. MIT Press, Cambridge, MA, 2001.

UNCERTAINTY AND PROBABILISTIC REASONING

MARKOV DECISION PROCESSES

Adaptive Control of Acyclic Progressive Processing Task Structures

Stéphane Cardon **Abdel-Illah Mouaddib**
CRIL-IUT de Lens-Université d'Artois
Rue de l'université, S.P. 16
62307 Lens Cedex France
{cardon, mouaddib}@cril.univ-artois.fr

Shlomo Zilberstein
Computer Science Dept
University of Massachusetts
Amherst, MA 01003 USA
zilberstein@cs.umass.edu

Richard Washington
NASA Ames Research Ctr
MS 269-2 Moffet Field
CA 94035, USA
rwashington@arc.nasa.gov

Abstract

The progressive processing model allows a system to trade off resource consumption against the quality of the outcome by mapping each activity to a graph of potential solution methods. In the past, only semi-linear graphs have been used. We examine the application of the model to control the operation of an autonomous rover which operates under tight resource constraints. The task structure is generalized to directed acyclic graphs for which the optimal schedule can be computed by solving a corresponding Markov decision problem. We evaluate the complexity of the solution analytically and experimentally and show that it provides a practical approach to building an adaptive controller for this application.

1 Introduction

Planetary rovers are unmanned vehicles equipped with cameras and a variety of scientific sensors. They have proved to be a cost-effective mechanism in space exploration and will continue to play a major role in future NASA missions [Washington *et al.*, 1999]. Despite dramatic improvements in capabilities and computational power, space systems continue to operate with scarce resources such as power, computer storage, and communication bandwidth. They also have a limited life span. As a result, it is necessary to maximize operation time during periods of no communication with the control center. It is also necessary to manage resources carefully when deciding what activity should be performed and how to best accomplish the rover's tasks. The main objective of this work is to develop planning and monitoring algorithms that can optimize scientific return when operating with limited resources.

For this purpose, the progressive processing model provides a suitable framework [Mouaddib and Zilberstein, 1997; 1998]. Progressive processing allows a system to trade off computational resources against the quality of the result similar to other resource-bounded reasoning techniques such as *"flexible computation"* [Horvitz, 1987],*"anytime algorithms"* [Dean and Boddy, 1988; Zilberstein, 1996], *"imprecise computation"* [Hull *et al.*, 1996; Liu *et al.*, 1991] and *"design-to-time"* [Garvey and Lesser, 1993]. The specific characteristic of progressive processing is the mapping of the computational process (or a plan) into a hierarchical graph of alternative methods. Each method has an associated probabilistic description of the resource/quality tradeoff it offers.

In previous work, Mouaddib and Zilbertstein [Mouaddib and Zilberstein, 1997; 1998] presented a solution to the control problem of progressive processing. This problem is to decide at run-time which subset of methods should be executed so as to maximize the overall utility of a system. The reactive controller is adaptive and takes into account the remaining resources and possible changes in the environment. This is achieved by reformulating the control problem as a Markov decision process (MDP). An efficient implementation that can handle run-time modifications of the overall plan has been implemented [Mouaddib and Zilberstein, 1998]. In the past, only semi-linear task structures have been used. That is, each step of a task could be implemented by one of several alternatives, after which the next step is considered.

The rover application, however, requires several extensions of the previously developed controller. Figure 1 shows a sample plan that illustrates the problem. In the plan, the rover locates an object and aims the camera, possibly after moving closer to the object. It can take a picture at one of three different resolutions, and can compress it using either high or low compression methods. In between, depending on the analysis of the image, the rover may decide to investigate the chemical components of the object. This plan presents several new requirements that have not been previously addressed:

Task inter-dependency: Task execution may depend on the outcome of previous actions. For example, the ability to investigate the chemical components depends on the results of image analysis.

Non linearity: The overall task structure becomes a directed acyclic graph. Execution follows a linear path through the graph.

Multiple resources: The controller must optimize its operation with respect to multiple resources.

The contribution of this paper is three fold. First, we generalize progressive processing to non-linear task structures with one resource and construct an optimal controller for such plans. Second, we analyze the complexity of the solution both analytically and empirically and show that it is an effective approach. Third, we examine the implications of increasing

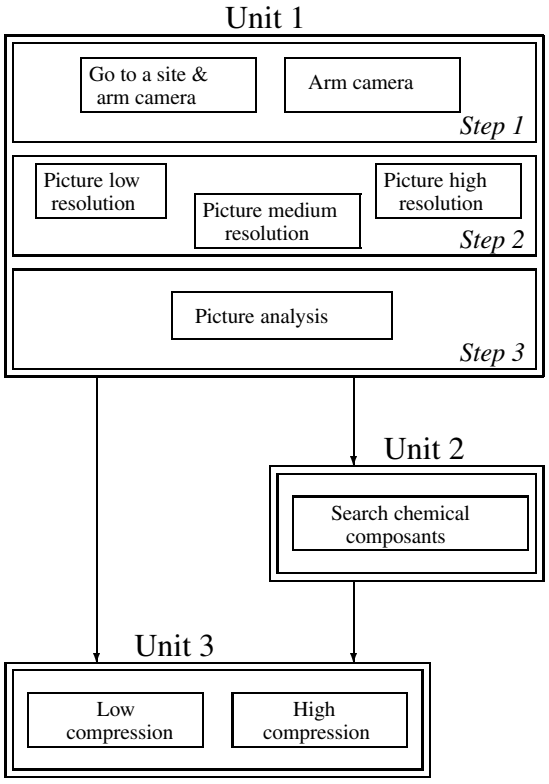


Figure 1: Planetary rover's plan

the number of resources and size of the plan on the complexity. We address this problem by introducing the notion of opportunity cost that can be estimated quickly for large, non-linear plans.

2 Progressive processing control problem

The work presented in this paper builds on the concepts of progressive processing units and plans. In this section, we define these notions and define our problem.

2.1 Preliminary definitions

Progressive processing is a reasoning technique that creates an incremental solution of a problem by using a hierarchy of steps, where each step contributes to the solution. This hierarchy can be viewed as the reasoning scheme which leads us to the solution. We denote by *PRU* (progressive processing unit) this hierarchy. Notice that we may associate a utility to the execution of a PRU's step. This utility is an abstract measure of the step's quality in the solution.

With this intuitive definition, a rover's plan may be seen as an ordered set of PRUs. More generally, the plan may be non-linear, i.e., a directed acyclic graph of PRUs. One unit has many successors. In this graph, a step will be considered for execution when the succession of steps executed so far has sufficient quality to enable this step.

For the example of planetary rovers, we have three PRUs. The first corresponds to going to a site, taking a picture and analyzing this image in order to find rocks. The second depends on the analysis results. If the site contains rocks, the rover may activate its APXS (alpha proton X-ray spectrometer) in order to determine the chemical components of the rocks. The last PRU corresponds to compressing the scientific data (images and chemical components).

We may transform this plan to a linear plan but, in that case, we lose generality and the inter-unit dependency information. In fact, the non-linearity of a plan and the notion of dependence allow for more realistic applications.

The planetary rover's problem (or progressive processing problem control) is to choose the set of executing tasks that maximize the global utility while respecting resource constraints.

2.2 Formal definitions

We formalize these definitions, focusing on one resource: operation time.

Definition 1 *A **plan** $\mathcal{P}$ is a directed acyclic graph of progressive processing units, $P_1, P_2, ..., P_{\mathcal{N}}$.*

Example: One plan is figure 1, where P_1 is unit 1, P_2 is unit 2 and P_3 is unit 3.

Definition 2 *A **progressive processing unit** (PRU), P_i, consists of a set of steps, $n_i = \{N_{i,1}, N_{i,2}, ..., N_{i,\#n_i}\}$, a deadline d_i and a set of successors, $s_i = \{succ(j_1), ..., succ(j_{\#s_i})\}$, where $succ(j_l) = (I^Q_{j_l}, P_{j_l})$ and $I^Q_{j_l}$ is the utility interval $[MinQ_{j_l}, MaxQ_{j_l}]$ corresponding to minimal and maximal utilities required to move to the next unit P_{j_l}.*

Example: PRU P_1 has three steps: $N_{1,1}$ is step 1, $N_{1,2}$ is step 2 and $N_{1,3}$ is step 3. It has two successors: P_2 and P_3.

Definition 3 *A **step** j of a unit i, $N_{i,j}$, consists of a set of tasks, $t_{i,j} = \{T_{i,j,1}, T_{i,j,2}, ..., T_{i,j,\#t_{i,j}}\}$ and a user-specified attribute, skippable, which indicates if this step may be skipped or not. We denote by $s_{i,j}$ the skippable attribute, and $s_{i,j} = 1$ when $N_{i,j}$ is skippable and 0 otherwise. We define $N_{i,0}$ to be the special step (with no tasks) that is used to show the beginning of executing a unit.*

Example: Step 2 of unit 1 has three tasks. The first $T_{1,2,1}$ is "Picture low resolution", the second $T_{1,2,2}$ is "Picture medium resolution" and the third $T_{1,2,3}$ is "Picture high resolution". This step can be skipped if we don't want to take a picture at a particular site.

Definition 4 *A **task** k of a step j of a unit i, $T_{i,j,k}$, consists of an executable module, a utility (or quality) and a discrete probability distribution $rc_{i,j,k}$ of resources consumed when the module is executed. The number of elements of $rc_{i,j,k}$ is $\#rc_{i,j,k}$.*

The probabilities can be determined from ground experimental rovers and simulations.

Example: Task 2 of step 2 of unit 1, $T_{1,2,2}$ consists of taking a medium resolution picture of the site. The distribution of resource consumption for this task, as used in our experiments, is $rc_{1,2,2} = \{(28, 0.2), (29, 0.6), (30, 0.2)\}$.

Definition 5 *The **progressive processing control problem** is to select at run-time the set of tasks that maximizes the global utility.*

2.3 Selecting tasks

When the rover has completed step $N_{i,j}$ (meaning that a task $T_{i,j,k}$ of that step has been executed or that the step has been skipped), it has three possible decisions:

- it can execute one task $T_{i,j+1,k'}$ of the next step $N_{i,j+1}$ if this step exists,
- it can skip the next step $N_{i,j+1}$, if it exists and it is skippable,
- it can move to a accessible successor unit $P_{succ(i)}$.

The optimal decision is the one that maximizes the global utility. The sequence of decisions will determine the set of tasks executed. The global utility is the cumulative reward of the executed tasks (reward is measured by the qualities associated to the tasks).

In this context, the progressive processing control problem is a state space where a transition from one state to an another is the decision made by the rover. The state represents the last executed step and the remaining resources. In fact, the choice of the next action (when the rover is in a particular state) depends only on the current state. This is the **Markov property**. So, the progressive processing problem control may be seen as a problem of controlling a Markov decision process.

2.4 Markov Decision Process Controller

We now define the corresponding Markov decision process.

States

A Markov decision process is a graph of states such that:

Definition 6 *A **state**, $[N_{i,j}; RR]$, consists of the last executed step $N_{i,j}$ and the remaining resources.*

Definition 7 *The **accumulated quality** is the sum of qualities of all executed task.*

Definition 8 ***Quality dependency of units**: A state $[N_{i',0}; RR']$ is the successor of $[N_{i,j}; RR]$ when the accumulated quality of $[N_{i,j}; RR']$ belongs to the utility interval $I^Q_{i'}$ and $(I^Q_{i'}, P_{i'}) \in succ_i$. We say in this case that $[N_{i',0}; RR']$ is an accessible state from $[N_{i,j}; RR']$.*

With the quality dependency, we express the dependence between the problems, including the precedence constraint (quality not null or belonging to a certain interval).

Example: To go to site 2, the rover should have executed $T_{1,3,1}$ (analysis of taken picture). So, we give a high reward to $T_{1,3,1}$ and $MinQ_2 = q_{1,3,1}$, where $(I^Q_2, P_2) \in succ_1$.

Definition 9 *A state is **terminal** when all the resources have been fully elapsed, or when there is no accessible successor.*

Transitions

We have three possible transition types: E, S, and D.

Definition 10 *A **transition** $\mathbf{E_{i,j,k}}$ (type E) is probabilistic. It consists of executing a task $T_{i,j,k}$ of the next step $N_{i,j}$. We denote it $[N_{i,j-1}; RR] \xrightarrow{E_{i,j,k}} [N_{i,j}; RR - \Delta R]$, where ΔR represents the resources consumed for the execution of $T_{i,j,k}$.*

Definition 11 *A **transition** $\mathbf{S_{i,j}}$ (type S) is deterministic. It consists of skipping the next step $N_{i,j}$. We denote it $[N_{i,j-1}; RR] \xrightarrow{S_{i,j}} [N_{i,j}; RR]$.*

Definition 12 *A **transition** $\mathbf{D_i}$ (type D) is deterministic. It consists of passing to an another unit. We denote it $[N_{i',j}; RR] \xrightarrow{D_i} [N_{i,0}; RR]$, where P_i is an accessible successor of $P_{i'}$.*

Transition rules

Now, let us define transition rules.

Pr(state f | state e, transition t) is the probability of moving from e to f when taking transition t.

Let $e = [N_{i,j-1}; RR]$ be the current state.

1. Transitions $E_{i,j,k}$
   Let ΔR be the resources consumed by task $T_{i,j,k}$.
   (a) $\Delta R \leq RR$, $Pr([N_{i,j}; RR - \Delta R] | e, E_{i,j,k}) = Pr(rc_{i,j,k} = \Delta R)$,
   (b) $\Delta R > RR$, $Pr([N_{i',0}; d_{i'} - d_i] | e, E_{i,j,k}) = Pr(rc_{i,j,k} > RR)$, where $[N_{i',0}; d_{i'} - d_i]$ is accessible from e.
2. Transitions $S_{i,j}$ (if $s_{i,j} = 1$, i.e., the step is skippable) $Pr([N_{i,j}; RR] | e, S_{i,j}) = 1$.
3. Transitions D_i
   $Pr([N_{i',0}; d_{i'} - d_i + RR] | e, D_i) = 1$, where $[N_{i',0}; d_{i'} - d_i + RR]$ is accessible from e.

2.5 State's value

The value associated to each state $V([N_{i,j}; RR])$ is a measure of utility of this state in the MDP. This value is calculated recursively by using the value of successor states.

For this, we denote $e = [N_{i,j-1}; RR]$, $\mathcal{E}_\mathcal{A} = \{$*states accessible from* $e\}$ and $ea_{i'} \in \mathcal{E}_\mathcal{A}$ an accessible state for which the associated task is in unit $P_{i'}$.

- Transitions $E_{i,j,k}$

$$\begin{aligned} v_E &= \sum_{\Delta R \leq RR} [Pr(rc_{i,j,k} = \Delta R) \\ &\qquad \cdot V([N_{i,j}; RR - \Delta R])] \\ &\quad + q_{i,j,k} \\ v_F &= \max_{\forall i', ea_{i'} \in \mathcal{E}_\mathcal{A}} [Pr(rc_{i,j,k} > RR) \\ &\qquad \cdot V([N_{i',0}; d_{i'} - d_i])] \end{aligned}$$

- Transitions $S_{i,j}$

$$v_S = s_{i,j} \cdot V([N_{i,j}; RR])$$

- Transitions D_i

$$v_D = \max_{\forall i', ea_{i'} \in \mathcal{E}_\mathcal{A}} V([N_{i',0}; d_{i'} - d_i + RR])$$

So, the value of one state is:

$$V(e) = \max_{\forall E_{i,j,k}, S_{i,j}, D_i} (v_E + v_F, v_S, v_D) \qquad (1)$$

3 Performance of the MDP controller

We have implemented the execution model described above, the policy-construction algorithm, and a simulator that allows plans to be executed and visualized in the rover domain. The performance of the resulting system is evaluated using the simulator. This section illustrates the results obtained using the system.

3.1 Analysis

Because of the one-to-one correspondence between the states of MDP and the states of the control, the optimal solution of the MDP is the optimal control.

Claim 1 (MDP size) *The number of states in the MDP is bounded by:*

$$\mathcal{N} \cdot \max_{\forall P_i} d_i \cdot \max_{\forall P_i} \#n_i \tag{2}$$

Proof: Note that two states with same remaining resources and associated step generate the same successors. So, to avoid an exponential complexity, we must notice already created states.

Since consumable resources are bounded, as well as the number of steps, we obtain a bounded number of states. Moreover, two states are different if and only if the associated steps are different or the remaining resources are different. So, we have the bound: $\mathcal{N} \cdot \max_{\forall P_i} d_i \cdot \max_{\forall P_i} \#n_i$. □

Claim 2 (solution time) *The time elapsed to calculate the optimal policy for the MDP is bounded by:*

$$\mathcal{N} \cdot (\max_{\forall P_i} d_i)^2 \cdot \max_{\forall P_i} \#n_i \cdot \left[\max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \cdot \max_{\forall P_i, \forall N_{i,j}, \forall T_{i,j,k}} \#rc_{i,j,k} + 2 \right] \tag{3}$$

Proof: The time consumed to solve the MDP is bounded by the number of transitions multiplied by the cost of searching already created states (if we don't want an exponential complexity).

The number of transitions from a state $[N_{i,j-1}; RR]$ is bounded by the number of transitions of each type: $\#rc_{i,j,k}$ of type E, and one each of type S and D. So, for that one state, the number of transitions generated is bounded by:

$$\left[\sum_{k=1}^{\#t_{i,j}} \#rc_{i,j,k} \right] + 2 \leq \max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \cdot \max_{\forall P_i, \forall N_{i,j}, \forall T_{i,j,k}} \#rc_{i,j,k} + 2.$$

Therefore, the bound on the number of transitions in the entire MDP is (using equation 2):

$$\mathcal{N} \cdot \max_{\forall P_i} d_i \cdot \max_{\forall P_i} \#n_i \cdot \left[\max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \cdot \max_{\forall P_i, \forall N_{i,j}, \forall T_{i,j,k}} \#rc_{i,j,k} + 2 \right]$$

Finally, the search of already created states involves d_i states in the worst case: since we store states by unit and by step, only those states associated with the specified unit and step need to be checked.

So, we obtain the following bound:

$$\mathcal{N} \cdot (\max_{\forall P_i} d_i)^2 \cdot \max_{\forall P_i} \#n_i \cdot \left[\max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \cdot \max_{\forall P_i, \forall N_{i,j}, \forall T_{i,j,k}} \#rc_{i,j,k} + 2 \right]$$

□

In conclusion, our controller has a complexity of $\mathcal{O}(\mathcal{N} \cdot \max_{\forall P_i} d_i)$ in space and $\mathcal{O}(\mathcal{N} \cdot (\max_{\forall P_i} d_i)^2)$ in policy-construction time. We also notice that if the deadline is not a function of N, we obtain a linear complexity.

3.2 Experimental results

We take a non-linear plan composed of $\mathcal{N}$ units and of depth $\log \mathcal{N}$. Each unit is composed of four steps. The first step consists of going to a site, the second of arming the camera, the third of taking the photo (low, medium or high resolution) and the last of compressing the obtained scientific data.

For our experiments, we set the average time to execute this unit to be 32 seconds. So, the deadline of each unit in plan is 32 seconds multiplied by the depth of this unit in the plan. If we assume that the number of nodes at each level of the plan increases by at least a constant factor $b > 1$, then the depth of the plan is $\mathcal{O}(\log \mathcal{N})$, and we obtain the complexity $\mathcal{O}(\mathcal{N} \cdot \log \mathcal{N})$ in space and $\mathcal{O}(\mathcal{N} \cdot (\log \mathcal{N})^2)$ in time.

However, we notice that we use exactly the resources needed to execute a task. What happens if we consume resources in fixed-size packets instead of exactly the needed resources?

$\mathcal{N}$	Value (size=1)	Value (size=2)	Error
20	94.38	90.80	3.79%
40	133.76	129.88	2.90%
60	166.76	160.71	3.63%
80	197.37	187.95	4.77%
100	219.37	209.51	4.49%

Table 1: Value computed as a function of $\mathcal{N}$, using packets of resources.

Table 1 shows state values computed for packet sizes of 1 and 2; little error is incurred by increasing the packet size. Figure 2 shows policy-creation time for plans of realistic size. Using a packet size of 2, the time used to solve the MDP is divided by 4 and the number of states created is divided by 2.

In fact, we can generalize this result:

Claim 3 ***If*** *EC is the number of states created and TC is the time consumed to create these states (for packet size equal to 1),* ***then*** *$\frac{EC}{u}$ is the maximal number of created states and $\frac{TC}{u^2}$ the maximal time elapsed (for packet size equal to u).*

$\frac{EC}{u}$ and $\frac{TC}{u^2}$ are bounds since equations 2 and 3 are bounds for EC and TC.

Proof: We suppose that EC and TC are number of created states and time consumed for packet size equal to 1. We now take packets of u resources.

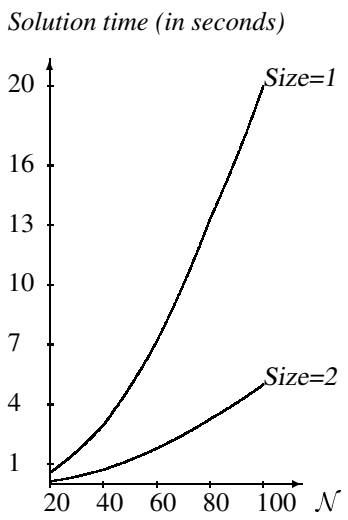


Figure 2: Solution time for packet size equal to 1 and 2

Notice that size of packet has no influence on the size of the problem $\mathcal{N}$. Moreover, it leaves the number of steps in unit and number of tasks in step invariant.

So, the packet size proportionally reduces the deadline of each unit ($d_i \rightarrow \frac{d_i}{u}$). Therefore, equation 2 implies that $\frac{EC}{u}$ is the maximal number of created states and equation 3 implies that $\frac{TC}{u^2}$ is the maximal time consumed to create these states.

The last term of equation 3 can be changed by packets, but it is just a constant. □

4 Dynamic operation and scalability

In the previous section, we presented an optimal solution to the control problem of acyclic progressive processing task structures. In this section, we extend the applicability of the solution to situations in which the global policy cannot be computed at run-time either by the control center or the rover itself. There are two primary motivations to avoid run-time construction of the policy.

1. **Dynamic operation:** We want to be able to modify the plan at run-time (e.g., insert or delete a PRU) without necessarily recomputing the control policy.
2. **Scalability:** We want to be able to track more resources, and the complexity of global policy construction grows exponentially with the number of resources.

While we focus in this paper on the dynamic operation, scalability is also enhanced by the proposed solution. Our approach is to exploit the fact that the units of the plan are largely independent. We try to capture the dependency of the execution of each unit on the remaining plan using a notion similar to opportunity cost [Zilberstein and Mouaddib, 1999].

4.1 Dynamic control with one resource

Definition 13 *Let $v_L([N_{i,j}; RR]$ represent the optimal local value of the state after step $N_{i,j}$ with remaining resources RR.*

Note that v_L is the same as V for a modified plan in which unit i has no successors. v_L is simply the value of the best policy for PRU i ignoring the remaining plan, hence it is called local value.

Claim 4 *The global value function can be reformulated as follows.*

$$V([N_{i,j}; RR]) = \max_{\Delta R}\{v_L([N_{i,j}; RR - \Delta R] + v_D([N_{i,j}; \Delta R])\} \quad (4)$$

Proof: This is an immediate result of the definition of v_L, V and v_D, and the additivity of the utility. □

For each PRU, we can easily compute the local value function v_L and construct a local policy off-line. The global value function (and policy) can be constructed at run-time, if we can obtain a fast estimate of v_D.

4.2 Estimating v_D

One way to think about v_D is as the value of terminating the work on a certain PRU with some level of resources, then using the resources for the remaining plan. This is exactly what we need to know. Estimating v_D is equivalent to estimating the global value of the remaining plan. We have examined several approximation schemes for v_D. Due to space limitations, we only sketch one simple approach.

- Construct an optimal local value function and local policy for each PRU (done once, off-line).
- For each level of RR, compute the expected value and expected resource consumption by the optimal local policy (done once, off-line).
- Go over the plan graph "backwards" from the leaves and compute an expected value for each node, for each level of RR at that node. The backup is relatively simple: maximizing for each level of RR at that node the combined parent/child value. In this process, it is necessary to take into account the difference between the deadlines of each parent/child pair. The quality dependency constraints are enforced with respect to the *expected* quality of the parent.

This algorithm is linear in the size of the graph. The computational savings and sub-optimality of the outcome are the result of using precalculated expected resource consumption and value for each PRU.

5 Conclusion

We have presented a solution to the problem of adaptive control of acyclic progressive processing tasks. This approach, which relies on solving a corresponding MDP, generalizes earlier work on MDPC [Mouaddib and Zilberstein, 1998]; it permits for the first time the treatment of non-linear progressive processing task structures. Moreover, our approach addresses effectively the high degree of uncertainty regarding resource consumption. In that sense, it improves on existing models for resource-bounded reasoning such as *"imprecise computation"* [Hull *et al.*, 1996; Liu *et al.*, 1991] and *"design-to-time"* [Garvey and Lesser, 1993]. Finally,

the model captures inter-task quality dependency similar to the enable and disable relationships in the *"design-to-time"* framework [Garvey *et al.*, 1993].

We also address in the paper a potential deficiency of the approach which may require a large amount of memory and time to create and solve the MDP and to store the resulting policy. Using packets of consumable resources, we managed to reduce the size of the MDP with a small loss of quality (4.5% for packet size equal to 2). Finally, we address the issue of dynamic operation and scalability by approximating the solution of the MDP.

The solution presented in this paper includes time as the only resource. Future work will allow us to represent a vector of resources [Mouaddib, 2000] such as storage capacity and power. An additional challenge is to develop more precise estimates of the value function v_D and adapt the approach to the case of multiple resources. A related project is using reinforcement learning to estimate v_D [Bernstein and Zilberstein, 2001].

More accurate models of resource usage can be obtained by using results from actual rover testing or high-fidelity simulations. We plan to make use of ongoing development of a high-fidelity rover simulation at NASA Ames Research Center to refine our models to make them more accurate and applicable to realistic rover problems, with the goal of testing our approach on actual rover hardware.

The PRU used in our experiments (Cf. 3.2) can be extended to solve a satellite scheduling problem. With this approach, more units can be scheduled than by traditional methods. Indeed, at this moment, a satellite in orbit around the world schedules the equivalent of only about 20 units. With our approach, it would be able to schedule 100, thus increasing its productivity 5-fold.

Acknowledgements

Support for this work was provided in part by the National Science Foundation under grants IRI-9634938, IRI-9624992, and INT-9612092, and by a Project of Plan Etat/Nord-Pas-De-Calais, and by IUT de Lens.

References

[Bernstein and Zilberstein, 2001] Daniel S. Bernstein and Shlomo Zilberstein. Reinforcement learning for weakly-coupled mdps and an application to planetary rover control. In *UAI (submitted for publication)*, 2001.

[Dean and Boddy, 1988] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Seventh National Conference on Artificial Intelligence*, pages 49–54, 1988.

[Garvey and Lesser, 1993] A. Garvey and V. Lesser. Desing-to-time real-time scheduling. *IEEE Transaction on systems, Man, and Cybernetics*, 23:1491–1502, 1993.

[Garvey *et al.*, 1993] A. Garvey, M. Humphrey, and V. Lesser. Task interdependencies in desing-to-time real-time scheduling. In *AAAI*, pages 580–585, 1993.

[Horvitz, 1987] E. J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Workshop UAI-87*, pages 429–444, 1987.

[Hull *et al.*, 1996] D. Hull, W. C. Feng, and J. W. S. Liu. Operating system support for imprecise computation. In *AAAI Fall Symposium on Flexible Computation*, pages 96–99, 1996.

[Liu *et al.*, 1991] J. Liu, K. Lin, W. Shih, J. Chung A. Yu, and W. Zhao. Algorithms sor scheduling imprecise computations. *IEEE Computer*, 24:58–68, 1991.

[Mouaddib and Zilberstein, 1997] A-I. Mouaddib and S. Zilberstein. Handling duration uncertainty in meta-level control of progressive processing. In *IJCAI*, pages 1201–1206, 1997.

[Mouaddib and Zilberstein, 1998] A-I. Mouaddib and S. Zilberstein. Optimal scheduling of dynamic progressive processing. In *ECAI*, pages 449–503, 1998.

[Mouaddib, 2000] A. I. Mouaddib. Optimizing multi-criteria decision quality in a progressive processing system. In *AAAI Symposium on realtime autonomous agent*, pages 56–61, 2000.

[Washington *et al.*, 1999] R. Washington, K. Golden, J. Bresina, D. E. Smith, C. Anderson, and T. Smith. Autonomous rovers for mars exploration. In *Proceedings of The 1999 IEEE Aerospace Conference*, 1999.

[Zilberstein and Mouaddib, 1999] S. Zilberstein and A-I. Mouaddib. Reactive control of dynamic progressive processing. In *IJCAI*, pages 1268–1273, 1999.

[Zilberstein, 1996] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17:73–83, 1996.

An Improved Grid-Based Approximation Algorithm for POMDPs

Rong Zhou and Eric A. Hansen
Computer Science Department
Mississippi State University
Mississippi State, MS 39762
{rzhou,hansen}@cs.msstate.edu

Abstract

Although a partially observable Markov decision process (POMDP) provides an appealing model for problems of planning under uncertainty, exact algorithms for POMDPs are intractable. This motivates work on approximation algorithms, and grid-based approximation is a widely-used approach. We describe a novel approach to grid-based approximation that uses a variable-resolution regular grid, and show that it outperforms previous grid-based approaches to approximation.

1 Introduction

A partially observable Markov decision process (POMDP) models planning problems for which actions have stochastic effects and sensors provide imperfect and incomplete state information. Originally developed in the operations research community, this model has been adopted by the AI community as a framework for research in planning under uncertainty and related problems of reinforcement learning.

A standard approach to solving a POMDP is to transform it into an equivalent, fully observable Markov decision process with a state space that consists of all probability distributions over the core states of the POMDP, and to solve the POMDP in this form. For a POMDP with n core states, the transformed state space is the n-dimensional simplex, or *belief simplex*. Although a continuous state space such as this presents a computational challenge, discrete approximations of continuous state spaces are a natural and often-used approximation technique. Grid-based approximation was the first approach adopted for solving POMDPs, and it is still widely-used [Drake, 1962; Kakalik, 1965; Eckles, 1966; Lovejoy, 1991; Brafman, 1997; Hauskrecht, 1997; 2000]. A finite grid is placed over the belief simplex, values are computed for points in the grid, and interpolation is used to evaluate all other points in the simplex. This is closely related to grid-approximation techniques for other continuous MDPs [Munos & Moore, 1999].

Different approaches to constructing a grid have been explored. Lovejoy (1991) describes a *fixed-resolution regular grid*, in which the points of the grid are spaced in a regular pattern and divide the belief simplex into equal-sized subsimplices. This allows a very efficent interpolation algorithm based on the concept of triangulation. However, the size of the grid grows exponentially with any increase in resolution. Hauskrecht (1997, 2000) and Brafman (1997) propose *non-regular grids* that allow the points of the grid to be spaced unevenly in the belief simplex, in order to approximate the contours of the value function as economically as possible. However, interpolation algorithms for non-regular grids are much less efficient. In this paper, we develop a *variable-resolution regular grid* that combines the strengths of both approaches. It allows the resolution of the grid to be adjusted in different regions of the belief simplex, but in a regular pattern that makes it possible to generalize the efficient interpolation algorithm for regular grids. The result is a higher quality approximation at less computational cost.

The paper is organized as follows. We begin with a review of the POMDP model, the grid approximation approach, and Lovejoy's interpolation algorithm for regular grids. We generalize the interpolation algorithm for use with a variable-resolution regular grid, and discuss methods for adjusting the resolution of the grid to achieve the best quality-time tradeoff for the approximation. We evaluate this approach analytically and experimentally, and show that it outperforms previous grid-approximation methods.

2 Background

We consider a discrete-time POMDP with a finite set of states S, a finite set of actions A, and a finite set of observations Z. Each time period, the environment is in some state $s \in S$, the agent takes an action $a \in A$ for which it receives an immediate reward with expected value $r(s,a)$, the environment makes a transition to state $s' \in S$ with probability $P(s'|s,a)$, and the agent observes $z \in Z$ with probability $P(z|s',a)$. Let b denote a vector of state probabilities, called a *belief (or information) state*, where $b(s)$ denotes the probability that the environment is in state s. If action a is taken and observation z follows, the successor belief state, denoted $\tau(b,a,z)$, is determined by revising each state probability as follows,

$$\tau(b,a,z)(s') = \frac{\sum_{s\in S} P(z,s'|s,a)b(s)}{P(z|b,a)},$$

where $P(z,s'|s,a) = P(z|s',a)P(s'|s,a)$ and the denominator is a normalizing factor $P(z|b,a) = \sum_{s'\in S} P(z|s',a) \sum_{s\in S} P(s'|s,a)b(s)$. A POMDP is

solved by finding a rule for selecting actions, called a policy, that optimizes a performance objective. We assume the objective is to maximize expected total discounted reward over an infinite horizon (where $\beta \in [0,1)$ is the discount factor). An optimal policy has a value function that satisfies the Bellman optimality equation,

$$V^*(b) = \max_{a \in A} \left[r(b,a) + \beta \sum_{z \in Z} P(z|b,a) V^*(\tau(b,a,z)) \right],$$

where $r(b,a) = \sum_{s \in S} b(s) r(s,a)$. Exact algorithms for solving this optimality equation are intractable for all but trivial problems. This motivates work on approximation methods.

2.1 Grid-based approximation

Let G denote a grid that contains a finite set of belief states, $b_1^G ... b_{|G|}^G$. A belief state b is a *convex combination* of the belief states in grid G if

$$b(s) = \sum_{i=1}^{|G|} \lambda(i) b_i^G(s), \text{ for all } s \in S,$$

where λ is a vector of size $|G|$, $\lambda(i) \geq 0$ for all $\lambda(i)$, and $\sum_{i=1}^{|G|} \lambda(i) = 1$. For each belief state b_i^G in the grid, we store a value denoted $\hat{v}(b_i^G)$. The *convex interpolation function*,

$$\hat{V}(b) = \sum_{i=1}^{|G|} \lambda(i) \hat{v}(b_i^G),$$

defines a continuous and piecewise-linear value function over all belief states, given values for the belief states in the grid. To compute values for the belief states in the grid, we use value iteration with the update rule,

$$\hat{v}(b_i^G) = \max_{a \in A} \left\{ r(b_i^G, a) + \beta \sum_{z \in Z} P(z|b_i^G, a) \hat{V}(\tau(b_i^G, a, z)) \right\},$$

which converges to a unique fixed-point solution that is guaranteed to be an upper bound on the optimal value function.

A belief state can be expressed as many different convex combinations of grid points. The convex combination that provides the best upper bound can be found by solving the following linear program.

Variables: $\lambda(i)$ for $1 \leq i \leq |G|$
Minimize: $\sum_{i=1}^{|G|} \lambda(i) \hat{V}(b_i^G)$
Constraints: $\sum_{i=1}^{|G|} \lambda(i) b_i^G(s) = b(s)$, for all $s \in S$
$\sum_{i=1}^{|G|} \lambda(i) = 1$
$\lambda(i) \geq 0$ for $1 \leq i \leq |G|$

Using linear programming to find the best interpolation can be very expensive. However, any convex combination provides an upper bound, and fast methods for finding a non-optimal convex combination can produce good upper bounds. Design of a grid-approximation strategy requires addressing two questions. What interpolation method provides a good tradeoff between time and quality? What belief states should be included in the grid? Every grid must contain the corner points of the belief simplex to ensure that a convex combination can always be found. Different strategies for adding other points to a grid have been explored. We begin with a review of a fixed-resolution regular grid and the efficient interpolation algorithm it supports. In the rest of the paper, we develop a variable-resolution generalization of this.

2.2 A fixed-resolution regular grid

Lovejoy (1991) constructs a regular grid as follows. Let M be a positive integer that represents the *resolution* of the grid. The set of belief states in the grid is defined as,

$$G = \left\{ b = \left(\frac{1}{M}\right) m | m \in I_+^{|S|}, \sum_{i=1}^{|S|} m(i) = M \right\},$$

where $I_+^{|S|}$ denotes the set of $|S|$-vectors of non-negative integers. The grid divides the belief simplex into a set of equal-size sub-simplices. Note that when $M = 1$, the grid contains only the corner points of the simplex. The number of points in the grid is given by the formula:

$$|G| = \frac{(M + |S| - 1)!}{M!(|S| - 1)!}$$

The advantage of a regular grid is that it allows an elegant and efficient method of interpolation. To evaluate a belief state b requires finding the vertices of the smallest sub-simplex that contains b, and then finding the coefficients of interpolation $\lambda(i)$. Lovejoy defines a second grid of integer vectors,

$$G' = \left\{ q \in I_+^{|S|} | M = q_1 \geq q_2 \geq q_3 \geq ... \geq q_n \geq 0 \right\},$$

and converts a belief state $b \in G$ into an integer vector $x \in G'$ in order to perform interpolation. Because of a one-to-one correspondence between grid points in G and G', the sub-simplex that contains $x \in G'$ can be used to determine the sub-simplex that contains $b \in G$. Here, we summarize the steps of the interpolation algorithm. We refer to Lovejoy (1991) for a detailed explanation.

1. Given a belief state b, create an $|S|$-vector x such that $x(s) = M \sum_{i=s}^{|S|} b(i)$ for $1 \leq s \leq |S|$. (This transforms probability density function b into cumulative distribution function x.)
2. Let v be the largest integer $|S|$-vector such that $v(s) \leq x(s)$ for all $s \in S$.
3. Let d be an $|S|$-vector such that $d(s) = x(s) - v(s)$ for all $s \in S$.
4. Let p be an $|S|$-vector that contains a permutation of the integers $1, 2, ..., |S|$ that orders the components of d in descending fashion, so that $d(p(1)) \geq d(p(2)) \geq ... \geq d(p(|S|))$.
5. Find the vertices $\{v^i : 1 \leq i \leq |S|\}$ of the sub-simplex in G' that contains x, as follows:

$$v^1(s) = v(s), \text{ for } 1 \leq s \leq |S|$$
$$v^{i+1}(s) = \begin{cases} v^i(s) + 1 & \text{if } s = p(i) \\ v^i(s) & \text{otherwise} \end{cases}$$

By the isomorphism between G and G', this identifies the corresponding vertices $\{b_i^G : 1 \leq i \leq |S|\}$ of the sub-simplex in G that contains b.

6. Find the barycentric coordinates $\{\lambda(i) : 1 \leq i \leq |S|\}$ for the interpolation, as follows:

$$\begin{aligned} \lambda(i) &= d(p(i-1)) - d((p(i)), \text{ for } 2 \leq i \leq |S| \\ \lambda_1 &= 1 - \sum_{i=2}^{|S|} \lambda(i). \end{aligned}$$

7. Let $\hat{V}(b) = \sum_{i=1}^{|S|} \lambda(i) \hat{V}(b_i^G)$.

The complexity of each of these steps is $O(|S|)$, with the possible exception of step 4 which requires sorting the elements of an $|S|$-vector. Although sorting has worst-case complexity $O(|S| \lg |S|)$, we can improve this by noting that the difference vector d is uniformly distributed over $[0, 1)$. Bucket sort is ideally suited for input data that is uniformly distributed between 0 and 1, and has only linear average-case complexity. Using bucket sort, the average-case complexity of the interpolation algorithm is $O(|S|)$. We note the important fact that the complexity of interpolation does not depend on the size of the grid.

We store the grid using a hash table. Our hash function uses lexicographical ordering of the integer vectors in G' to map each vector in G' to a unique integer from 0 to $(M + |S| - 1)!/M!(|S| - 1)!) - 1$. The hash function is perfect if the size of the hash table is equal to or greater than $(M + |S| - 1)!/M!(|S| - 1)!)$, which is the total number of possible grid points. (The hash function assumes a maximum resolution, M.) Because the number $(M+|S|-1)!/M!(|S|-1)!) - 1$ explodes quickly as M and $|S|$ increase, it is usually impossible to have a hash table this large. We resolve collisions by associating a linked list with each slot, and use the hash code to uniquely identify the grid point in the list.

3 A variable-resolution regular grid

The disadvantage of a fixed-resolution grid is that increasing the resolution of the value function approximation in one region of the belief simplex requires increasing its resolution everywhere. This causes an exponential explosion in the size of the grid, and makes this approach infeasible for all but small problems. To address this problem, Lovejoy (1991) suggested a variable-resolution generalization of his method that would "maintain the simplicity of the Freudenthal triangulation, but allow the mesh M to vary on different portions of $\Pi(S)$." However, neither Lovejoy nor anyone else developed this extension.

In the rest of this paper, we develop this variable-resolution generalization of Lovejoy's regular grid. We discuss how to generalize the Freudenthal interpolation and how to select the appropriate resolution for different regions of the simplex. In allowing the resolution of the grid to vary, we adopt the rule that the resolution M is always a positive integer power of 2. This ensures that vertices of low-resolution sub-simplices are also vertices of higher-resolution sub-simplices, and still useful when resolution is increased. From now on, we let M denote the highest resolution anywhere in the grid.

3.1 Interpolation

Smallest complete sub-simplex To perform interpolation for belief state b, we search for the smallest (*i.e.*, highest-resolution) sub-simplex containing b that is complete (*i.e.*, all its vertices are in the grid). The lowest possible resolution of the grid is 1, and the highest possible resolution, M, is given. (We must know M in order to create the hash function for the hash table used to store the grid.) For any resolution, the algorithm summarized in Section 2.2 identifies the vertices of the sub-simplex containing b, and we can check whether all of these are in the grid. Thus, we can use binary search to find the resolution between 1 and M with the smallest complete sub-simplex containing b. Because the grid always includes the corner points of the belief simplex, the search is guaranteed to find a complete sub-simplex at some resolution. The complexity of this interpolation algorithm is $O(|S| \lg \lg M)$, where the factor $\lg \lg M$ reflects the complexity of binary search and the fact that only resolutions that are powers of two must be considered.

Virtual points Each point in a regular grid is a vertex in many different sub-simplices. In a variable-resolution grid in which not every part of the grid is refined to the same resolution, this means that there will be many sub-simplices for which some but not all of their vertices are grid points. The smallest complete algorithm considers sub-simplices containing b at different resolutions, until it finds the smallest complete one. In its search, it always considers the sub-simplex at the next higher resolution. Even though this sub-simplex is not complete, many of its vertices may be present in the grid. It can be advantageous to use these higher-resolution grid points in interpolation, and it is possible to do so by "filling in" the missing vertices with what we call *virtual vertices*. Any vertex that is missing from a sub-simplex is a belief state for which we can create an upper bound value by performing interpolation, since the missing vertex is contained in a lower-resolution sub-simplex. We call this computed value a *virtual vertex*. By using virtual vertices to fill in the missing vertices of the incomplete sub-simplex at the next higher resolution after the smallest complete one, we can improve the interpolated value. The cost for this improvement is an increase in the worst-case complexity of interpolation by a factor of $|S|$ compared to the smallest complete algorithm (since in the worst case every vertex may be missing from a sub-simplex). But this complexity factor can be improved by not creating virtual vertices for an incomplete sub-simplex with more than *MaxVirtual* vertices missing. This limits the increase in worst-case complexity to a factor of *MaxVirtual* $\leq |S|$. In practice, we found that another technique significantly reduces the average-case complexity of virtual interpolation without adversely affecting the result. For an incomplete sub-simplex, let λ*Sum* denote the sum of the barycentric coordinates for vertices that are present. (Note that $\lambda\text{sum} \in [0, 1)$ for an incomplete sub-simplex.) We don't perform virtual interpolation for a sub-simplex for which λ*Sum* $< \lambda$*Threshold*. Adjusting this threshold lets us adjust a time-quality tradeoff for this method of interpolation. We report experimental results for different threshold values in Section 4.

3.2 Refining the grid

We now discuss the question of how to refine the grid, that is, how to selectively adjust the resolution of the grid in different regions of the belief simplex. A natural strategy is to do so in a way that reduces the error of the approximation. This requires a method for estimating the error. Since our grid-based approximation is an upper bound on the optimal value function, the error can be estimated by measuring its distance from a lower-bound function.

Lovejoy (1991) discusses how to use a grid to compute a lower-bound function, as well as an upper-bound function. The lower-bound function is represented by a set of $|S|$-vectors, $\Gamma = \{\alpha_1, \alpha_2, \ldots, \alpha_{|G|}\}$, with the value of belief state b computed as follows,

$$\bar{V}(b) = \arg\max_{\alpha_i \in \Gamma} \sum_{s \in S} b(s)\alpha_i(s).$$

One vector is associated with each grid belief state, and is computed by value iteration using the following update rule.

1. For each action a and observation z, let

$$\alpha^{i,a,z} = \max_{\alpha_i \in \Gamma} \sum_{s \in S} \tau(b_i^G, a, z)(s)\alpha_i(s).$$

2. For each action a, let

$$\alpha^{i,a}(s) = r(s,a) + \beta \sum_{s',z} P(z, s'|s, a)\alpha^{i,a,z}(s'), \forall s \in S.$$

3. Let $\alpha^i = \arg\max_{\alpha^{i,a}} \text{ for } {}_{a \in A} \sum_{s \in S} b_i^G(s)\alpha^{i,a}(s)$.

The complexity of performing this update for every grid belief state is $O(|G||A||Z||S|^2)$. Although value iteration using this update is guaranteed to produce a value function that is a lower bound on the optimal value function, it may not converge to a fixed point. Hauskrecht (1997, 2000) and Zhang *et al.* (1999) propose refinements of this update that guarantee convergence, but they fail to bound the size of Γ or the complexity of the update. So we do not use their refinements.

Given these upper and lower-bound grid-based value functions, Lovejoy describes an algorithm that computes a *uniform error bound* (which is the maximum difference between the upper and lower-bound functions for any belief state). Unfortunately, it has complexity $O(|S|!|G|^2)$, and our experience indicates that it is intractable for problems with more than nine or ten states. However, Lovejoy notes that it is easy to compute an approximate error bound (defined as the maximum difference between the upper and lower-bound functions for any *grid* belief state). This is guaranteed to be tighter than the uniform bound, and we can use it to refine the grid.

Once we have identified the grid point(s) with the largest error, what belief states should we add to the grid in order to reduce the error at these points? The value of a grid belief state depends on the interpolated value of its successor belief states, so refining the grid at the successor belief states will reduce the error at the grid point. For a successor belief state b, we increase the resolution of the grid by adding the vertices of the $2M$-resolution sub-simplex containing b to the grid, where M is the resolution of the smallest complete sub-simplex containing b in the current grid. This adds at most $|S|$ points to the grid, which is the maximum number of missing vertices in the higher-resolution sub-simplex.

This first approach to refining the grid tries to improve the grid-based approximation everywhere, without considering reachability from a start state. If the starting belief state for a POMDP is given, a policy can be found using forward search [Satia & Lave, 1973; Washington, 1997; Hansen, 1998; Hauskrecht, 2000]. This focuses computation on regions of the belief simplex that are reachable from the starting belief state. Upper and lower-bound functions can be used to prune branches of the search tree, as well as to compute an error bound for the starting belief state (by backing-up upper and lower-bound values from the leaves of the search tree to the root). This suggests a more focused approach to refining grid-based upper and lower-bound functions: attempt to reduce the error bound of the belief state at the root of the search tree. This can be done by refining the grid for belief states on the fringe of the search tree that contribute most to the value of the starting belief state.

Although we choose to focus on grid-refinement strategies that attempt to reduce the error bound of the approximation, other strategies are possible. For example, given upper and lower-bound value functions, it is sometimes possible to identify the optimal action for a belief state using action elimination. It may be desirable to refine the grid at points where the optimal action is uncertain, and by the same reasoning, unnecessary to refine it at points where the optimal action can be determined, regardless of the error bound at these points. Hauskrecht (1997, 2000) and Brafman (1997) suggest other heuristics for choosing points to add to a (non-regular) grid. These heuristics consider such factors as reachability, likely improvement of corner points of belief simplex, and likely effect on the policy, and can be viewed as complementary to our strategy of reducing an error bound.

3.3 Comparison to non-regular grids

A variable-resolution regular grid is not the only way to allow the resolution of a grid to vary in different regions of the belief simplex. Another approach is a non-regular grid that allows the points of the grid to occur anywhere in the belief simplex, and not necessarily in a regular pattern that allows for triangulation. Hauskrecht (1997, 2000) and Brafman (1997) propose non-regular grids, and we now compare this approach to our variable-resolution regular grid.

A drawback of non-regular grids is that interpolation is more difficult. Hauskrecht proposes an interpolation algorithm that creates a convex combination using one point in the grid and $|S| - 1$ corner points of the belief simplex. Because it tests each point in the grid in order to select the one that gives the best interpolated value, its complexity is $O(|G||S|)$. Brafman proposes an interpolation algorithm that relies on the grid belief states being sorted in decreasing order of entropy. Given a belief state b for which interpolation is to be performed, Brafman's algorithm searches the list of grid belief states until a belief state b_i^G is found that assigns positive probability only to states to which b assigns positive probability. The maximum coefficient c for which $(b - c \cdot b_i^G \geq 0)$ is calculated, and the process is repeated with $(b - c \cdot b_i^G)$ instead of b until interpolation is complete. (Searching a list

sorted by entropy has the effect of prefering low-information belief states that are likely to be closer to b.) The algorithm has worst-case complexity $O(|G||S|^2)$, and requires an initial sort with complexity $O(|G| \lg |G||S|)$. Although both interpolation algorithms are simple, their complexity depends on the size of the grid. This is a serious drawback that limits the size of non-regular grids in practice.

3.4 Value iteration

Although the complexity of interpolation in regular grids does not depend on the size of the grid, the complexity of value iteration in both regular and non-regular grids does. Values for grid belief states are computed using value iteration, with values initialized to known upper bounds (such as their completely observable values). In a variable-resolution regular grid, the compexity of the first iteration of value iteration is $O(|G||A||Z||S|^2 \lg \lg M)$, where $|G|$ is the number of grid points, $|A||Z|$ is the number of successor belief states of each grid point, $|S|^2$ is the worst-case complexity of computing a successor belief state by Bayesian conditioning, and $\lg \lg M$ reflects the added complexity of interpolation. Because both successor belief states and barycentric coordinates used for interpolation can be cached for re-use, the complexity of subsequent iterations of value iteration is only $O(|G||A||Z||S|)$, where $|S|$ is the complexity of computing an interpolated value using the cached barycentric coordinates. In practice, caching successor belief states and barycentric coordinates dramatically improves the running time of value iteration.

Because the same successor belief states and barycentric coordinates are used in each iteration of value iteration, this grid-based approximation is a finite-state MDP and convergence to a fixed-point solution in polynomial time is guaranteed [Hauskrecht, 2000]. In non-regular grids, the barycentric coordinates used for interpolation are re-computed each iteration. Because the complexity of interpolation in non-regular grids depends on $|G|$, the complexity of each iteration of value iteration in a non-regular grid is quadratic in $|G|$. This underscores the advantage of using a regular grid.

In addition to limiting the complexity of interpolation, it is helpful to limit the factor $|G||A||Z|$, which is the number of times interpolation (and other calculations involved in computing a backup) are performed in each iteration of value iteration. The obvious way to limit this factor is to limit the size of the grid. This adjusts a tradeoff between quality of approximation and grid size, and a variable-resolution grid lets us space the points of the grid to achieve the best possible approximation for a given grid size. Action eliminations can be used to limit $|A|$. As for $|Z|$, it may be reduced by ignoring some, or all, observations, and evaluating the successor belief states of actions. If grid-based interpolation is used to evaluate the successor belief states of actions, however, the grid-based approximation may no longer be an upper bound. The upper bound guarantee is lost because information provided by the observation is ignored. (The upper bound guarantee can be restored by using the corner points of the belief simplex to perform interpolation, since this is equivalent to assuming perfect information, but this impairs the quality of the approximation. If always done, the grid-based approximation becomes no better than the completely observed heuristic.)

Finally, we note that it is possible to improve a grid-based approximation without adding points to the grid. Instead of performing a conventional backup using one-step lookahead, the value of a grid point can be improved further by using a deeper lookahead search that we call a *multi-step backup*. Using branch-and-bound or AO* search, the upper-bound function can be used to prune the search tree and the depth of the lookahead can adjust a tradeoff between search complexity and improvement of of value. We report some experimental results for this technique in the next section.

4 Performance Results

We experimentally evaluate our grid-based approximation algorithm using several test problems from the literature. These include the machine maintenance problem that is Lovejoy's (1991) only test example; the four most-difficult-to-solve problems from a test set used by Cassandra *et al.* (1997) (see Table 1); the 20-state, 6-action, 8-observation gridworld navigation problem used as a test example by Hauskrecht (1997, 2000); and the 89-state, 5-action, 17-observation gridworld navigation problem introduced by Littman *et al.* (1995) and used as a test example by Brafman (1997).

Interpolation We first compare the time-quality tradeoff for different methods of interpolation in a variable-resolution regular grid. We use the five small test problems in Table 1 for this comparison, so that we can also compute an optimal interpolation using linear programming. Figure 1 shows the performance of interpolation using virtual points relative to performance using the smallest complete sub-simplex method alone. We measure improvement as a percentage of the difference between the interpolated value using the smallest complete method and the optimal interpolated value. Virtual point interpolation is tested with three different λSum threshold values; 0.2, 0.5, and 0.8. As expected, a smaller λSum threshold achieves better quality at the cost of speed. For these problems, virtual point interpolation takes only slightly longer than the smallest complete sub-simplex method. (Interpolation using linear programming runs an average of 16 times slower.) For problems with larger state spaces, virtual point interpolation takes relatively longer be-

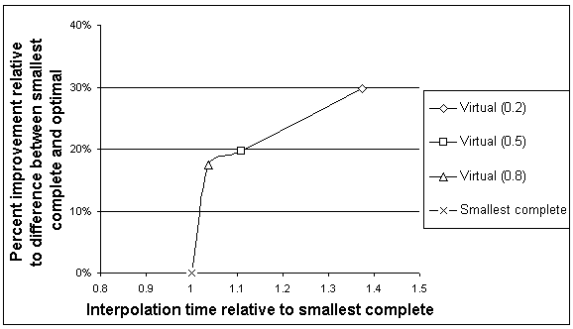


Figure 1: Time-quality tradeoff for interpolation using virtual points with different λSum thresholds. Results are averaged over the five test problems in Table 1.

Problem	$\|S\|$	$\|A\|$	$\|Z\|$	Error	Fixed resolution M	Fixed resolution $\|G\|$	Fixed resolution UB time	Fixed resolution LB time	Variable resolution M	Variable resolution $\|G\|$	Variable resolution UB time	Variable resolution LB time
Machine Maintenance	3	4	2	0.02	20	231	0.5	63.2	64	45	3.04	7.98
Network Monitoring	7	4	2	36.41	4	210	1.44	84.69	8	71	7.07	7.57
Shuttle Docking	8	3	5	0.38	4	330	0.43	128.29	8	47	0.35	4.08
Navigation	12	4	6	0.03	4	1365	3.74	2133.3	32	299	5.34	149.86
Aircraft Identification	12	6	5	0.29	4	1365	7.62	4378.4	8	186	2.57	80.43

Table 1: A comparison of fixed and variable-resolution grids for 5 small test problems, showing the grid size, as well as maximum resolution and time (in CPU seconds) used to compute upper and lower-bound grid approximations, in order to achieve the same error bound.

cause its worst-case complexity is quadratic in the size of the state space instead of linear, but reasonable performance can still be achieved by adjusting the λSum threshold. Optimal interpolation using linear programming is infeasible for larger problems.

Interpolated values are typically close to optimal. There appear to be two reasons for sub-optimality. First, interpolation using a sub-simplex is less flexible than interpolation using a convex combination. (Every sub-simplex is a convex combination, but not every convex combination is a sub-simplex.) As a result, better interpolated values can sometimes be found using convex combinations that do not correspond to sub-simplices. The second reason is more problematic. Although the interpolation function is piecewise linear and continuous, there is no guarantee that it is convex. For most problems, we found that it *is* convex (or nearly so). However, we sometimes found non-convexities. When they exist, the "nearest" grid points do not necessarily give the best interpolation. We found non-convexities in two of our test problems: the network monitoring problem from Cassandra's test set and Hauskrecht 20-state grid navigation problem. The large error bound for the network monitoring problem in Table 1 reflects this. Note that these non-convexities occur in both fixed-resolution and variable-resolution regular grids.

One explanation for non-convexity is that "localized" transitions may prevent values from propagating throughout the grid. That is, non-convexities may occur when improved values in one area of the grid do not improve values in another area of the grid because there is not a chain of successor belief states between them through which they can propagate.

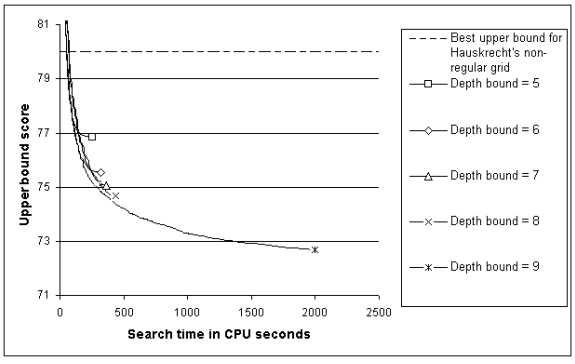


Figure 2: Performance of multi-step backups for Hauskrecht's maze navigation problem. Results are for a fixed-resolution regular grid with $M = 2$, which has 210 grid points.

For both test problems with non-convexities, we found that multi-step backups corrected the problem. (Multi-step backups are described in the last paragraph of Section 3.4.) Instead of refining the grid at points with the largest error bound, we performed multi-step backups at these points. For the network monitoring problem, using multi-step backups with a depth bound of ten reduced the error bound from 36.41 to 0.7 after 150 CPU seconds. Figure 2 shows the positive effect of multi-step backups on Hauskrecht's 20-state maze problem. (For this problem, the upper bound score is defined as the average value of a fixed set of 1000 randomly generated belief states plus the corner points, a metric Hauskrecht uses.) For the other test problems, multistep backups do not improve the value function as much as grid refinement does. As a rule of thumb, when the error bound for a problem remains large despite grid refinement, multi-step backups may correct the problem.

Refining the grid Table 1 compares the performance of a fixed-resolution grid and a variable-resolution grid on five small test problems. Each problem is solved using a fixed-resolution regular grid, and the error bound is measured. Then a variable-resolution grid is used to find a solution with the same error bound. We use virtual interpolation with λSum = 0.5, for the variable-resolution grid. The results show that a variable-resolution approximation can achieve the same accuracy with a much smaller grid.

Figure 3 shows how the error bound decreases as the size of grid increases for the Aircraft Identification problem. Re-

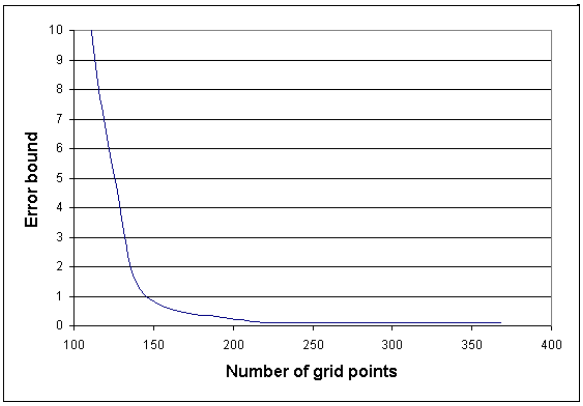


Figure 3: Decrease in error in a variable-resolution regular grid, as function of grid size. Results are for aircraft identification problem.

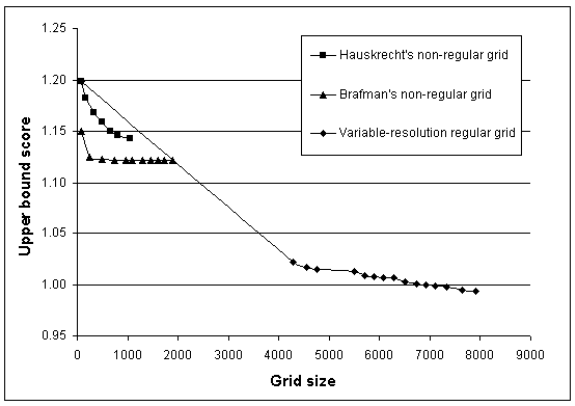


Figure 4: Upper bound value as a function of grid size. Results are for 89-state hallway navigation problem.

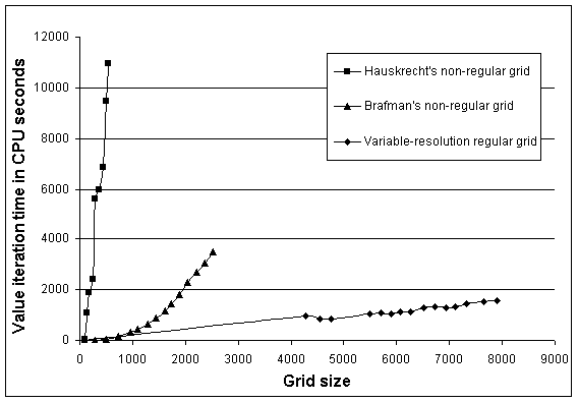


Figure 6: Time for value iteration to converge as a function of grid size. Results are for 89-state hallway navigation problem.

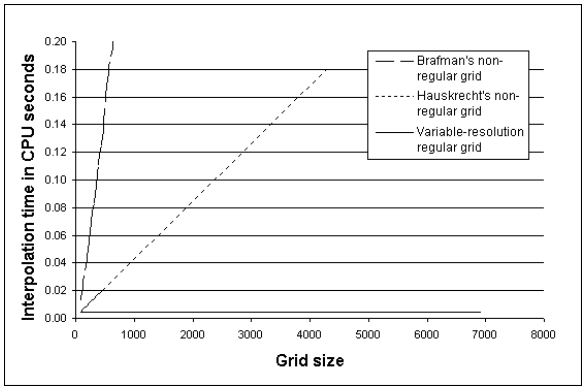


Figure 5: Average interpolation time as a function of grid size. Results are for 89-state hallway navigation problem.

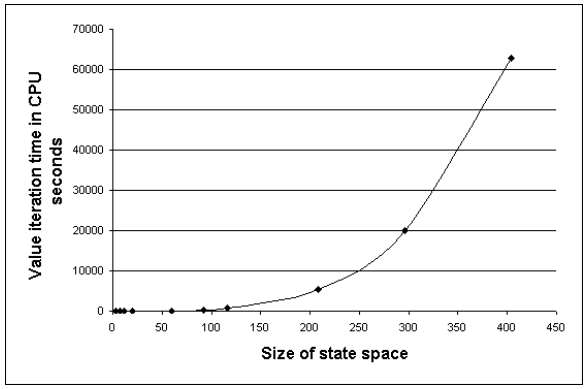


Figure 7: Increase in value iteration time in a fixed-resolution regular grid with M = 2, as a function of state space size.

sults are similar for other problems. Using the error bound to determine where to refine the grid requires a lower bound function, but Lovejoy's grid-based lower bound function can be time-consuming to compute. For the 89-state hallway navigation problem, computing an $|S|$-vector for each and every grid point is prohibitive. But it is not necessary. In our experiments with the hallway problem, we got good results using a lower-bound function consisting of only ten $|S|$-vectors. Each iteration, we computed an $|S|$-vector for the ten grid points with the largest error bound.

Comparison to non-regular grids Figure 4 shows how the quality of the upper bound function improves as the size of the grid increases for the 89-state hallway navigation problem. It also shows that a better upper bound function can be computed using a variable-resolution regular grid than using a non-regular grid. The reason for this is that faster interpolation makes it possible to use a much larger grid. Figure 5 shows average interpolation time as a function of grid size for our regular grid, compared to Brafman's and Hauskrecht's non-regular grids, and dramatically underscores the advantage of regular grids. Figure 6 compares the time it takes value iteration to converge for each of these methods, as a function of grid size. Although the variable-resolution regular grid is much larger, it takes less time to compute. Brafman's value iteration algorithm is faster than Hauskrecht's (even though his interpolation algorithm is slower), because it disregards observations and only considers the successor belief states of actions. Even with this simplification, value iteration in Brafman's non-regular grid is slower than value iteration in a regular grid. Moreover, as noted earlier, ignoring observations means that his grid-based approximation is not guaranteed to be an upper bound (although the values it computes for this problem are, in fact, upper bounds).

Scalability Although a variable-resolution regular grid allows much larger grids (and thus better approximations), the size of the state space remains a problem in scaling-up this approach. Figure 7 compares the time it takes to compute a regular grid for a succession of POMDP problems with increasing state space size. (All problems have 5 actions and 17 observations.) Besides confirming that the size of the state space is the primary factor that limits the scalability of grid-based approximation, this gives some idea of the range of problems for which grid-based approximation is currently feasible. We have found that almost all of the running time of

value iteration is spent performing interpolation or computing successor belief states using Bayesian conditioning, and the complexity of both depends on the size of the state space. This points to where further work on grid-based approximation is needed. In order to improve the efficiency of interpolation and Bayesian conditioning for problems with larger state spaces, we need to integrate grid-based approximation with various techniques for state abstraction that have been explored for MDPs and POMDPs [Boutilier et al., 1999]. (State abstraction would also make it possible to compute Lovejoy's grid-based lower-bound function more efficiently.)

5 Discussion

POMDPs are notoriously difficult to solve, and finding even a bounded-optimal solution is an intractable problem [Lusena et al., 1998]. Grid-based approximation algorithms have polynomial-time complexity in the size of the grid and the number of states, actions, and observations [Hauskrecht, 2000]. Although there is no guarantee on the quality of the approximation that can be achieved in polynomial time, a well-designed grid-based approximation can be adjusted in several ways to optimize a time-quality tradeoff.

This paper introduces a variable-resolution regular grid that allows the resolution of the grid to be adjusted in different regions of the belief simplex, in order to approximate the contours of the value function as efficiently as possible. Although non-regular grids also allow this, interpolation in non-regular grids is more difficult and its complexity is a function of the size of the grid. In a variable-resolution regular grid, the complexity of interpolation is independent of the size of the grid. Thus, it is feasible to use much larger grids with this approach in order to achieve better approximations.

Our experiments indicate that it is easier to scale up this approach to large grids than to large state spaces, and the size of the state space remains the most prohibitive factor in the complexity of grid-based approximation. To use grid-based approximation to solve POMDPs with large state spaces, we need to integrate it with various techniques for state abstraction that have been explored for MDPs and POMDPs. This is a promising direction for future research.

Acknowledgments

We thank Tony Cassandra and Milos Hauskrecht for making available their test problems, and the anonymous reviewers for helpful comments. This research is supported in part by the National Science Foundation under grant IIS-9984952.

References

[Boutilier et al., 1999] Boutilier, C.; T. Dean; and S. Hanks. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11: 1–94.

[Brafman, 1997] Brafman, R. 1997. A heuristic variable grid solution method for POMDPs. In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), 727–733. Providence, RI.

[Cassandra et al., 1997] Cassandra, A.; Littman, M.; and Zhang, N. 1997. Incremental pruning: A simple, fast, exact algorithm for partially observable Markov decision processes. In Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence, 54-61.

[Drake, 1962] Drake, A. 1962. Observation of a Markov Process Through a Noisy Channel. Ph.D. thesis, Electrical Engineering Department, M.I.T., Cambridge, MA.

[Eckles, 1966] Eckles, J. 1966. Optimum replacement of stochastically failing systems. Ph.D. thesis, Department of Engineering-Economic Systems, Stanford University.

[Hansen, 1998] Hansen, E. 1998. Solving POMDPs by searching in policy space. In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98), 211–219. Madison, WI.

[Hauskrecht, 1997] Hauskrecht, M. 1997. Incremental methods for computing bounds in partially observable Markov decision processes. In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), 734–739. Providence, RI.

[Hauskrecht, 2000] Hauskrecht, M. 2000. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research* 13:33–94.

[Kakalik, 1965] Kakalik, J. 1965. Optimum policies for partially observable Markov systems. Technical report 18, Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA.

[Littman et al., 1995] Littman, M.; A. Cassandra; and L. Kaelbling. 1995. Learning policies for partially observable environments: Scaling up. In Proceedings of the Twelth International Conference on Machine Learning, 362–370.

[Lovejoy, 1991] Lovejoy, W. 1991. Computationally feasible bounds for partially observed Markov decision processes. *Operations Research* 39:162–175.

[Lusena et al., 1998] Lusena, C.; J. Goldsmith; and M. Mundhenk. 1998. Nonapproximability results for Markov decision processes. Technical Report, Computer Science Department, University of Kentucky.

[Munos & Moore, 1999] Munos, R. and A. Moore. 1999. Variable resolution discretization for high accuracy solutions of optimal control problems. In Proceedings of 16th International Joint Conference on Artificial Intelligence.

[Satia & Lave, 1973] Satia, J. and Lave, R. 1973. Markovian Decision Processes with Probabilistic Observation of States. *Management Science* 20(1):1–13.

[Washington, 1997] Washington, R. 1997. BI-POMDP: Bounded, incremental partially-observable Markov-model planning. In Proceedings of the Fourth European Conference on Planning (ECP-97).

[Zhang et al., 1999] Zhang, N.; S. Lee; and W. Zhang. 1999. A method for speeding up value iteration in partially observable Markov decision processes. In Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence.

UNCERTAINTY AND PROBABILISTIC REASONING

UNCERTAINTY

Weakening the Commensurability Hypothesis in Possibilistic Qualitative Decision Theory

Adriana Zapico
Departamento de Computación
Universidad Nacional de Río Cuarto
Campus de la U.N.R.C
5800 Río Cuarto
Argentina
zapico@dc.exa.unrc.edu.ar

Abstract

Models for Qualitative Decision under Uncertainty assuming that uncertainty is of possibilistic nature have been proposed in 1990's. In them, as in the classical approach of Expected Utility Theory, two alternatives emerge: à la Von Neumann and Morgenstern (VNM) or à la Savage. In the first one, uncertainty and preferences on consequences are measured on finite scales. Decisions are ranked in terms of the ranking induced, on their associated possibility distributions on consequences, by qualitative utility functions. To define these utilities, an hypothesis of commensurability, i.e. the existence of an onto order-preserving mapping h linking both scales of uncertainty and preference, is assumed. This hypothesis forces us to restrict to decision problems where the cardinality of the uncertainty values set is greater than or equal to the cardinality of the preference set. This point has been attacked in the possibilistic à la Savage approach, but as far as we know, it is an open question in the qualitative VNM approach. As a first step to weaken the commensurability hypothesis in this model, the preference orderings resulting of applying the qualitative criteria without requiring h to be onto are characterized. These orderings may be not continuous, but they satisfy a relaxed continuity axiom.

1 Introduction

The basic references in classical Decision Theory characterizing preference relations under uncertainty and the rationality hypothesis are Von Neumann and Morgenstern's *Expected Utility Theory(EUT)*[1944], and the version of Savage [1972]. Both approaches assume that *uncertainty is represented by probability distributions*. Von Neumann and Morgenstern assume a probability distribution encoding uncertainty on situations. Then, as each decision induces a probability distribution on the set of consequences X, *each decision is identified with its associated probability distribution on* X. For sorting decisions, associated distributions are ranked in terms of their expected value with respect to Decision Maker's (DM's for short) preferences on consequences. On the other hand, Savage [1972] proposes a somewhat different framework for *EUT*, axiomatically characterizing the preference relation *on acts* of Decision Makers that behave as *EUT* agents.

The classical axiomatic frameworks of Utility Theory have actually been questioned rather early (e.g. [Allais, 1953; Ellsberg, 1961]). Another problem with *EUT* is that it needs numerical probabilities for each state and numerical utilities for all possible consequences, but this assumption may be too strong if there is only incomplete or poor available information. As Doyle and Thomason [1999] comment in a recent paper, there are many experiences showing that usually people explain and make their decisions with partial, generic and "uncertain" information. Hence, a qualitative approach may give tools for representing this decision making behaviour. They summarize main proposals on Qualitative Decision Theory. Among them, we find those models that use *Possibility Theory* as uncertainty formalism in which, analogously to classical EUT, two alternatives emerge: *à la Von Neumann and Morgenstern*, initiated by Dubois and Prade in [1995] and latter developed and extended in [Dubois *et al.*, 1999; Zapico, 1999], and *à la Savage*, this proposal is summarized in [Dubois *et al.*, 1997; 1998].

Here, we are interested in the possibilistic qualitative counterpart to VNM's EUT. In this approach, both DM's preferences on consequences and uncertainty are graded on *finite ordinal sets* (i.e. scales equipped with the maximum, minimum and an order-reversing operations) *that are commensurate*. As each decision is linked with its associated possibility distribution on consequences, then for ranking decisions the associated distributions are ranked by qualitative utility functions -expressed in terms of maximum, minimum and reversing operations. To define these utilities an hypothesis of commensurability, i.e. the existence of an onto order-preserving mapping linking both scales of uncertainty and preference, is assumed. This forces us to restrict to decision problems where the cardinality of the uncertainty set is greater than or equal to the cardinality of the preference set, unfortunately in several problems this cardinality relation is not satisfied.

The commensurability hypothesis has been a point of interest for researchers in the possibilistic à la Savage approach (e.g. [Fargier and Perny, 1999]) but as far as we know, it has not been attacked in the qualitative Von Neumann and Morgenstern style. Here, we propose to weaken the commensurability hypothesis for this approach by not requiring the linking mapping scales involved to be onto. That is, we will characterize the preference ordering resulting of applying these qualitative criteria only requiring h to be an order-preserving mapping, in particular we show that not requiring h to be onto results in a relaxation of the continuity property of the preference relations on distributions. On the other way, replacing in the axiomatic framework the usual continuity axiom by the relaxed one is not enough for guaranteeing the relation is representable by the qualitative criteria, an axiom establishing how some special mixtures are handled is required.

This weakening of the commensurability hypothesis will allow us to deal with other types of problems, in particular, those in which the cardinality of the preference values set may be greater than the cardinality of the uncertainty set.

In the next Section we summarize the Possibilistic Qualitative Decision Theory (PQDT). Our proposal on weakening the commensurability hypothesis is introduced in Section 3, including the representation theorem for characterizing the preference relations representable by these "more general" utility functions. The respective extension of the model for involving non-normalized distributions are considered in it as well. Finally, in last Section we include some concluding remarks.

2 Background on PQDT

Let us define the PQDT framework. V denotes a *finite linear* plausibility scale, where $\inf(V) = 0$ and $\sup(V) = 1$, X is a *finite* set of consequences, and $\Pi(X, V)$ denotes *the set of consistent possibility distributions on X over V*, i.e.

$$\Pi(X, V) = \{\pi{:}X \to V \,|\, \max_{x \in X}\ \pi(x) = 1\}.$$

For the sake of simplicity, we shall use A for denoting both a subset $A \subseteq X$ and the normalized possibility distribution on X such that $\pi(x) = 1$ if $x \in A$ and $\pi(x) = 0$ otherwise. Hence, we can consider X as included in $\Pi(X, V)$.

The so-called *Possibilistic mixture* (possibilistic lottery), the qualitative counterpart of the probabilistic lottery, is an operation defined on $\Pi(X, V)$ that combines two possibility distributions π_1 and π_2 into a new one, denoted $(\lambda/\pi_1, \mu/\pi_2)$, with $\lambda,\ \mu \in V$ and $max(\lambda, \mu) = 1$, defined as:

$$(\lambda/\pi_1, \mu/\pi_2)(x) = max(min(\lambda, \pi_1(x)), min(\mu, \pi_2(x))).$$

With this definition, and taking into account the properties of maximum and minimum, the *"reduction of lotteries"* holds, i.e. $(\lambda/\pi_1, \mu/(\alpha/\pi_1, \beta/\pi_2)) \sim (max(\lambda, min(\mu, \alpha))/\pi_1,$ $min(\mu, \beta)/\pi_2)$.

U is a *finite linearly ordered* scale of preference (or utility), with $\sup(U) = 1$ and $\inf(U) = 0$, n_U is the reversing involution in U. As usual, we assume as working hypothesis the existence of a preference function representing DM's preference on consequences, i.e. there exists a function $u{:}X \to U$ that assigns a preference level of U to each consequence of X such that $u(x) \leq u(y)$ if and only if y is at least as preferred as x. Let $h{:}V \to U$ *be an order preserving function* relating V and U such that $h(0) = 0,\ h(1) = 1$.

In such a framework, *assuming also that h is onto*, Dubois et al. [1999] have characterized the preference relations induced by the utility functions

$$QU^-(\pi|u) = \min_{x \in X} \max(n(\pi(x)), u(x)),$$

where $n = n_U \circ h$. This criterion requires π to make, at least to some extent, all plausible consequences to be preferred ones[1]. Since in some problems this criterion may be too conservative, it may be interesting to consider an optimistic behaviour, like requiring π to make at least one of the good consequences highly plausible- at least to some extent. With this goal, Dubois et al. [1999] considered the utility function:

$$QU^+(\pi|u) = max_{x \in X} min(h(\pi(x)), u(x)).$$

which is dual to QU^-. Notice that $QU(.|u)$, restricted to X, coincides with u.

Dubois et al. have shown that[2] QU^- has an interesting property: QU^- preserves the possibilistic mixture, in the sense that it holds that:

$$QU^-(\lambda/\pi_1, \mu/\pi_2) = \quad min\{max(n(\lambda), QU^-(\pi_1), \\ max(n(\mu), QU^-(\pi_2)\}.$$

They also provide the following set of axioms AX for preference relations on $\Pi(X, V)$, with the max-min possibilistic mixture as the internal operation on $\Pi(X, V)$, to characterize the ordering induced by QU^- ($\preccurlyeq_{QU^-}$).

- $A1$: $\sqsubseteq$ is a total pre-order(reflexive, transitive and total).

This axiom guarantees the existence of maximal and of minimal elements on $(X, \sqsubseteq)$. We denote by $\overline{x}$ and $\underline{x}$ a maximal and a minimal ones respectively. For each $\lambda \in V$, let $\pi_\lambda^- = (1/\overline{x}, \lambda/\underline{x})$ and $\pi_\lambda^+ = (\lambda/\overline{x}, 1/\underline{x})$.

- $A2$ $(uncertainty\ aversion)$: if $\pi \leq \pi' \Rightarrow \pi' \sqsubseteq \pi$.
- $A3(independence)$: $\pi_1 \sim \pi_2 \Rightarrow (\lambda/\pi_1, \mu/\pi) \sim (\lambda/\pi_2, \mu/\pi)$.
- $A4(continuity)$: $\forall \pi \in \Pi(X, V), \exists \lambda \in V$ s.t. $\pi \sim \pi_\lambda^-$.

$A4$ establishes that distributions are preferentially equivalent to having a λ-level of uncertainty with respect to $\overline{x}$.

Theorem 2.1 *A preference relation $(\Pi(X, V), \sqsubseteq)$ satisfies axioms AX if, and only if, there exist*

- *a finite linearly ordered utility scale U, with* $\inf(U) = 0$ *and* $\sup(U) = 1$,
- *a preference function $u : X \to U$ s.t. $u^{-1}(1) \neq \emptyset \neq u^{-1}(0)$,*
- *an onto order-preserving function $h : V \to U$,*

[1] The utility of π is measured in terms of a degree of inclusion of the fuzzy set π of possible consequences into the fuzzy set of good results u.

[2] For the sake of a simpler notation, we write $QU^-(\pi)$ instead of $QU^-(\pi|u)$ when the mapping u is not relevant for the context. In fact, these utility functions also depend on h. To simplify, we omit h and will use the notation of QU to refer a utility involving an onto h and QU_W for the case of not requiring h this onto condition.

in such a way that $\pi' \sqsubseteq \pi$ *iff* $\pi' \preccurlyeq_{QU^-} \pi$,

For characterizing the optimistic behaviour representable by QU^+, it is necessary to change the uncertainty aversion axiom $A2$ by an uncertainty-prone postulate:

- $A2^+$: if $\pi \leq \pi'$ then $\pi \sqsubseteq \pi'$,

and to adequate the continuity axiom $A4$ replacing it with:

- $A4^+ :\ \forall \pi \in \Pi(X,V),\ \exists \lambda \in V$ s.t. $\pi \sim \pi^+_\lambda$.

In the next Section, the proposal for weakening the commensurability hypothesis is introduced.

3 A First Approach with a Weaker Commensurability Hypothesis

Let us remark that the difference with the works developed previously in [Dubois *et al.*, 1999; 1997] is that now h is ***not*** required to be onto. That is, given $h : V \to U$, an order-preserving mapping such that $h(0) = 0, h(1) = 1$, for any $\pi \in \Pi(X,V)$, consider the qualitative utility functions

$$\begin{aligned} QU^-_W(\pi|u) &= \min_{x\in X} \max(n(\pi(x)), u(x)), \\ QU^+_W(\pi|u) &= \max_{x\in X} \min(h(\pi(x)), u(x)), \end{aligned}$$

where $n = n_U \circ h$. Observe that $QU^-_W(\cdot|u)$ and $QU^-_W(\cdot|u)$, restricted to X, coincide with the preference function u.

It is interesting to notice the following properties.

Lemma 3.1 *These functions still preserve the mixture:*

$$\begin{aligned} QU^-_W(\lambda/\pi_1, \mu/\pi_2) = &\quad \min\{\max(n(\lambda), QU^-_W(\pi_1)), \\ &\quad \max(n(\mu), QU^-_W(\pi_2))\}, \\ QU^+_W(\lambda/\pi_1, \mu/\pi_2) = &\quad \max\{\min(h(\lambda), QU^+_W(\pi_1)), \\ &\quad \min(h(\mu), QU^+_W(\pi_2))\}. \end{aligned}$$

In particular, we have:

$$QU^-_W(\max(\pi_1, \pi_2)) = \min\{QU^-_W(\pi_1), QU^-_W(\pi_2)\},$$

$$QU^+_W(\max(\pi_1, \pi_2)) = \max\{QU^+_W(\pi_1), QU^+_W(\pi_2)\}.$$

Corollary 3.2 *It remains true that* $QU_W(\lambda/\pi_1, \mu/\pi_2)$ *is the median of three terms including* $QU_W(\pi_1), QU_W(\pi_2)$,*i.e.*

1. *if* $QU^-_W(\pi_1) < QU^-_W(\pi_2)$, *then*

$$QU^-_W(\lambda/\pi_1, \mu/\pi_2) = med\{QU^-_W(\pi_1), QU^-_W(\pi_2), n(\lambda)\}.$$

2. *if* $QU^+_W(\pi_1) < QU^+_W(\pi_2)$, *then*

$$QU^+_W(\lambda/\pi_1, \mu/\pi_2) = med\{QU^+_W(\pi_1), QU^+_W(\pi_2), h(\lambda)\}.$$

3.1 The Roles of the Onto Condition and of the Continuity Axiom

Let us observe that the onto condition of h is a basic requirement involved in the characterization of the relations, since the fact of allowing h to be a *non*-onto mapping results in that the continuity axiom $A4$ may not be true.

Example: For instances, consider consequences are either totally possible or impossible, i.e. $V = \{0,1\}$, while preferences are measured on $U = \{0 < w < 1\}$, being the set of consequences $X = \{\underline{x},\ x_1,\ \overline{x}\}$, with DM's preference on consequences $u(\underline{x}) = 0,\ u(x_1) = w,\ u(\overline{x}) = 1$. Since $h(0) = 0$ and $h(1) = 1$, it is obvious that $QU^-_W(\pi) = \min_{x\in X|\pi(x)=1} u(x)$. That is, the ordering induced by QU^-_W coincides with the *maximin* criterion while the ordering induced by QU^+_W coincides with the *maximax* one. Observe that for $\pi = \{x_1\}$, there is no $\lambda \in V$ such that π is preferentially equivalent to π^-_λ.

Hence, it is clear that the continuity property $A4$ for relations on $\Pi(X,V)$ may be lost if h is non-onto. For facing this point, a relaxed continuity axiom is proposed. The underlying idea is to relax the continuity of the preference in the sense that there exists a subset on X such that either the distributions are preferentially equivalent to individual consequences on this subset, or, the distributions are preferentially equivalent to having a λ-level of uncertainty with respect to $\overline{x}$ as usual.

On the other way, having a relation $(\Pi(X,V), \sqsubseteq)$ satisfying axioms $A1, A2, A3$ and the relaxed continuity one (see $A4RC$ in Section 3.2) is not enough for guaranteeing $\sqsubseteq$ is representable by QU^-_W as it is shown following.

Example: Consider the following framework:

$X = \{\underline{x} \sqsubset x \sqsubset \overline{x}\}$, $X_{NM} = \{x\}$, $V = \{0 < \beta < 1\}$, and consider the (reflexive and transitive) relation $\sqsubseteq$ s.t. $\underline{x} \sqsubset x \sqsubset \pi^-_\beta \sqsubset \overline{x}$, also satisfying

$$x \sim (1/\overline{x}, \beta/x) \sim (\mu/\overline{x}, 1/x) \quad \forall \mu \in V.$$

All other distributions are taken equivalent to $\underline{x}$. This relation satisfies axioms $A1, A2, A3$ and $A4RC$. But it is not representable by QU^-_W, since although $(1/\overline{x}, \beta/x) \sqsubset \pi^-_\beta$, we have $QU^-_W(\pi^-_\beta) \leq QU^-_W(1/\overline{x}, \beta/x)$.

3.2 Axiomatic Setting Proposed

The above discussion has led us to propose this new set of axioms AXM for preference relations on $\Pi(X,V)$, with the max-min mixture as the internal operation:

- $A1(structure)$: $\sqsubseteq$ is a total pre-order .
- $A2(uncertainty\ aversion)$: if $\pi \leq \pi' \Rightarrow \pi' \sqsubseteq \pi$.
- $A3(independence)$:
  $\pi_1 \sim \pi_2 \Rightarrow (\lambda/\pi_1, \mu/\pi) \sim (\lambda/\pi_2, \mu/\pi)$.
- $A4RC(relaxed\ continuity)$: There exists a subset[3] $X_{NM} \subseteq X$ s.t. all maximal and all minimal elements of $(X, \sqsubseteq)$ are in the complement of X_{NM}, and $(\forall \pi \in \Pi(X,V))$ either
  $(\exists \lambda \in V$ s.t. $\pi \sim \pi^-_\lambda)$ or $(\exists x \in X_{NM}$ s.t. $\pi \sim x)$.
- $AxMix$:
  1. if $x, y \in X_{NM},\ \beta \in V$ then

$$(1/x, \beta/y) \sim \begin{cases} x, & \text{if } (x \sqsubseteq y) \text{ or } (x \sqsubset \pi^-_\beta) \\ \pi^-_\beta, & \text{if } y \sqsubset \pi^-_\beta \sqsubset x \\ y, & \text{if } \pi^-_\beta \sqsubset y \sqsubset x, \end{cases}$$

[3] Observe that $X_{NM} = \emptyset$ is possible, being $A4$ recovered.

2. if $x \in X_{NM}$ then

$$(1/\pi_\lambda^-, \beta/x) \sim \begin{cases} \pi_\lambda^-, & \text{if } (\pi_\lambda^- \sqsubset x) \text{ or } (\pi_\lambda^- \sqsubseteq \pi_\beta^-) \\ \pi_\beta^-, & \text{if } x \sqsubset \pi_\beta^- \sqsubset \pi_\lambda^- \\ x, & \text{if } \pi_\beta^- \sqsubset x \sqsubset \pi_\lambda^-. \end{cases}$$

This axiom establishes how the relation handles (reduces) lotteries involving consequences that are not preferentially equivalent to binary lotteries of $\overline{x}$ and $\underline{x}$.

Now, we introduce some results that will be useful latter.

Lemma 3.3 *Axioms A1, A2, A3 , A4RC and AxMix imply $Ax2$: If $A \subseteq X$ then there is $x \in A$ s.t. $x \sim A$.*

Dubois et al. [1999, Lemma 1] have shown:

Lemma 1 *If $\sqsubseteq$ verifies axioms $A1$, $Ax2$ and $A2$, then:*

- $\underline{x} \sim (1/\overline{x}, 1/\underline{x}) \sim X$.
- *$\underline{x}$ and $\overline{x}$ are also the minimal and maximal elements of $(\Pi(X,V), \sqsubseteq)$.*

3.3 Representation of Pessimistic Qualitative Utilities

Next, we show that the preference ordering on $\Pi(X,V)$ induced by the qualitative/ordinal pessimistic utility QU_W^- satisfies the above set of axioms.

Lemma 3.4
Let $\preccurlyeq_{QU_W^-}$ be the preference ordering on $\Pi(X,V)$ induced by QU_W^-, i.e., $\pi \preccurlyeq_{QU_W^-} \pi'$ iff $QU_W^-(\pi) \leq QU_W^-(\pi')$. Then $\preccurlyeq_{QU_W^-}$ satisfies axioms set AXM.

Proof: $A1$ is easily verified, $A2$ is a consequence of maximum and minimum being non-decreasing functions.$A3$ results from the fact that QU_W^- preserves max-min possibilistic mixtures, while $AxMix$ is not difficult to verify taking into account Lemma 3.1. Now, we check axiom $A4RC$. If h is onto, $X_{NM} = \emptyset$ and $A4RC$ reduces to $A4$, hence, we are in the case detailed in Section 2. If h is non-onto, let $X_{NM} = (\{x |\ u(x) \in n(V)\})^c$. As $u^{-1}(1) \neq \emptyset \neq u^{-1}(0)$, and $h(0) = 0$ and $h(1) = 1$, if x is a maximal or a minimal element of $(X, \preccurlyeq_{QU_W^-})$, then $x \notin X_{NM}$. We have to prove that $\forall \pi \in \Pi(X,V)$ either $(\exists \lambda$ s.t. $QU_W^-(\pi) = QU_W^-(\pi_\lambda^-))$ or $(\exists x \in X_{NM}$ s.t. $QU_W^-(\pi) = QU_W^-(x))$. By definition of QU_W^-, $\forall \pi, \exists x_0 \in X$, s.t. $QU_W^-(\pi) = \max(n(\pi(x_0)), u(x_0))$. Hence,

- if $QU_W^-(\pi) = n(\pi(x_0))$, then $QU_W^-(\pi) = QU_W^-(\pi_\lambda^-)$ with $\lambda = \pi(x_0)$.
- Otherwise, $QU_W^-(\pi) = u(x_0)$. Now, either $u(x_0) \in n(V)$ or not. In the first option, $\exists \lambda \in V$ s.t. $QU_W^-(\pi) = n(\lambda) = QU_W^-(\pi_\lambda^-)$. While in the second one, $u(x_0) \in X_{NM}$, and $QU_W^-(\pi) = QU_W^-(x_0)$. □

Now, it is shown in the following Representation Theorem that the preference orderings satisfying AXM can always be represented by a pessimistic qualitative utility QU_W^-.

Theorem 3.5 *A preference relation $\sqsubseteq$ on $\Pi(X,V)$ satisfies axiom set AXM if, and only if, there exist*

(i) a finite linearly ordered utility scale U with $\inf(U) = 0$ and $\sup(U) = 1$,

(ii) a preference function u:$X \to U$ s.t. $u^{-1}(1) \neq \emptyset \neq u^{-1}(0)$,

(iii) an order-preserving[4] function h:$V \to U$ s.t. $h(0) = 0$ and $h(1) = 1$,

in such a way that $\pi' \sqsubseteq \pi$ iff $\pi' \preccurlyeq_{QU_W^-} \pi$.

Proof: The "if" part corresponds to the preceding Lemma. As for the "only if" part, we go on structuring the proof, in the following three steps:

1. The utility scale U and an order-preserving function h from V to U are defined.
2. A function QU_W^-:$\Pi(X,V) \to U$ representing $\sqsubseteq$, i.e. s.t. $QU_W^-(\pi) \leq QU_W^-(\pi')$ iff $\pi \sqsubseteq \pi'$ is introduced.
3. The preference function u:$X \to U$ is the restriction of QU_W^- to X and $n = n_U \circ h$, n_U being the reversing involution on U. It is proved that

$$QU_W^-(\pi) = \min_{x \in X} \max(n(\pi(x)), u(x)).$$

Now, we develop these steps.

1. As usual, $\sqsubseteq$ stratifies $\Pi(X,V)$ in a linearly ordered set of classes of equivalently preferred distributions ($\pi' \in [\pi]$ iff $\pi \sim \pi'$). We define $U = \Pi(X,V)/\sim$ with the natural (linear) order $[\pi] \leq [\pi']$ iff $\pi \sqsubseteq \pi'$. Denote by 1 and 0 the maximum and minimum elements of U, $[\overline{x}] = 1$ and $[\underline{x}] = 0$.
   Define the order reversing function n:$V \to U$ as $n(\lambda) = [\pi_\lambda^-]$. $A2$ guarantees that n reverses the order. Let n_U be the reversing involution in U, the linking-scales mapping h is defined as $h = n_U \circ n$.
2. Now, we define QU_W^- in three steps.
   (a) For all $\lambda \in V$, $QU_W^-(\pi_\lambda^-) = n(\lambda)$.
   (b) Secondly, $\forall x \in X_{NM}$, let $QU_W^-(x) = [x]$.
   It is easy to check that $\pi_\lambda^- \sqsubseteq \pi_{\lambda'}^-$ iff $QU_W^-(\pi_\lambda^-) \leq QU_W^-(\pi_{\lambda'}^-)$, and analogously, that restricted to distributions of type x, QU_W^- represents $\sqsubseteq$.
   (c) Finally, since $A4RC$ guarantees that $\forall \pi$, either $(\exists \lambda$ s.t. $\pi \sim \pi_\lambda^-)$ or $(\exists x \in X_{NM}$ s.t. $\pi \sim x)$, we define

$$QU_W^-(\pi) = \begin{cases} n(\lambda), & \text{if } \exists\ \lambda \text{ s.t. } \pi \sim \pi_\lambda^- \\ [x], & \text{if } \exists x \in X_{NM} \text{ s.t. } \pi \sim x. \end{cases}$$

   Notice that, because of $A4RC$, QU_W^- is well defined.
3. Now, u:$X \to U$ is defined as $u(x) = QU_W^-(x)$. Notice that $u(\overline{x}) = 1$ and $u(\underline{x}) = 0$. It remains to prove that $QU_W^-(\pi) = \min_{x \in X} \max(n(\pi(x)), u(x))$. With this goal, we will prove the following equalities:

[4] Note that h is not required to be onto.

- $QU^-_W(1/\pi_1, \beta/\pi_2) = \min(QU^-_W(\pi_1), \max(n(\beta), QU^-_W(\pi_2)))$.

  By $A4RC$, there are several alternatives for π_1, π_2 :
  (a) $\exists \mu, \lambda$ s.t. $\pi_1 \sim (1/\overline{x}, \lambda/\underline{x})$ and $\pi_2 \sim \pi^-_\mu$.
  (b) $\exists x, y \in X_{NM}$ s.t. $\pi_1 \sim x$ and $\pi_2 \sim y$,
  (c) $\exists \lambda \in V, x \in X_{NM}$ s.t. $\pi_1 \sim x$ and $\pi_2 \sim \pi^-_\lambda$,
  (d) $\exists \lambda \in V, x \in X_{NM}$ s.t. $\pi_1 \sim \pi^-_\lambda$ and $\pi_2 \sim x$.
  It is not difficult to verify that applying $A3$, $AxMix$ and reducing lotteries, in each case we obtain that
  $QU^-_W(1/\pi_1, \beta/\pi_2) = \min(QU^-_W(\pi_1), \max(n(\beta), QU^-_W(\pi_2)))$.
  In particular, we have that
  $QU^-_W(\max(\pi_1, \pi_2)) = \min(QU^-_W(\pi_1), QU^-_W(\pi_2))$.
  This may be easy generalized to
  $$QU^-_W(\max_{i=1,\ldots,p} \pi_i) = \min_{i=1,\ldots,p} QU^-_W(\pi_i).$$

- Now, we verify
  $$QU^-_W(\pi) = \min_{i=1,\ldots,p} \max(n(\pi(x_i)), u(x_i)).$$
  As π is normalized, $\exists x_j \in X$ s.t. $\pi(x_j) = 1$. Without loss of generality we assume $j = 1$. Then, let $\pi_i = (1/x_1, \pi(x_i)/x_i)$. Since $\pi = \max_{i=1,\ldots,p} \pi_i$, we have:
  $$\begin{aligned} QU^-_W(\pi) &= QU^-_W(\max_{i=1,\ldots,p} \pi_i) \\ &= \min_{i=1,\ldots,p} QU^-(\pi_i) \\ &= \min_{i=1,\ldots,p} \{\min(u(x_1), \max(n(\pi(x_i)), u(x_i))\} \\ &=^5 \min_{i=1,\ldots,p} \max(n(\pi(x_i)), u(x_i)). \end{aligned}$$

This ends the proof of the theorem. □

3.4 Representation of Optimistic Qualitative Utilities

For modeling an optimistic DM's behaviour, we consider the axioms set $AXM^+ = \{A1, A2^+, A3, A4RC^+, AxMix^+\}$, as usual being $\pi^+_\lambda = (\lambda/\overline{x}, 1/\underline{x})$, with

- $A2^+$: if $\pi \leq \pi'$ then $\pi \sqsubseteq \pi'$,
- $A4RC^+$: There exists a subset[6] $X_{NM} \subseteq X$, s.t. all maximal and all minimal elements of $(X, \sqsubseteq)$ are in its complement, and $\forall \pi \in \Pi(X, V)$ either $(\exists \lambda \in V$ s.t. $\pi \sim \pi^+_\lambda)$ or $(\exists x \in X_{NM}$ s.t. $\pi \sim x)$.
- $AxMix^+$:
  1. if $x, y \in X_{NM}$, $\beta \in V$ then
  $$(1/x, \beta/y) \sim \begin{cases} x, & \text{if } (x \sqsupseteq y) \text{ or } (x \sqsupset \pi^+_\beta) \\ \pi^+_\beta, & \text{if } y \sqsupset \pi^+_\beta \sqsupset x \\ y, & \text{if } \pi^+_\beta \sqsupset y \sqsupset x. \end{cases}$$
  2. if $x \in X_{NM}$ then
  $$(1/\pi^+_\lambda, \beta/x) \sim \begin{cases} \pi^+_\lambda, & \text{if } (\pi^+_\lambda \sqsupset x) \text{ or } (\pi^+_\lambda \sqsupseteq \pi^+_\beta) \\ \pi^+_\beta, & \text{if } x \sqsupset \pi^+_\beta \sqsupset \pi^+_\lambda \\ x, & \text{if } \pi^+_\beta \sqsupset x \sqsupset \pi^+_\lambda. \end{cases}$$

[5] Note that $\pi(x_1) = 1$, so $u(x_1) = \max(u(x_1), n(\pi(x_1))$.
[6] If $X_{NM} = \emptyset$, then axiom $A4^+$ is recovered.

We have analogous results to Lemmas 3.3 and 3.4, but for spaces reasons they are omitted here. The respective Representation Theorem of Optimistic Utility is:

Theorem 3.6 *A preference relation $\sqsubseteq$ on $\Pi(X, V)$ satisfies axioms set AXM^+ if, and only if, there exist*

(i) a finite linearly ordered utility scale U with $\inf(U) = 0$ and $\sup(U) = 1$,

(ii) a preference function $u{:}X \to U$ s.t. $u^{-1}(1) \neq \emptyset \neq u^{-1}(0)$,

(iii) an order preserving function $h{:}V \to U$ such that $h(0) = 0$ and $h(1) = 1$,

in such a way that $\pi' \sqsubseteq \pi$ iff $\pi' \preccurlyeq_{QU^+_W} \pi$.

3.5 A Particular Case: Maximin and Maximax Decision Criteria

Let us consider again the simplest scale of uncertainty, $V = \{0, 1\}$, that is, consequences can be either fully possible or fully impossible. This is a very particular case since for any preference scale U, the only requirement to be fulfilled by a mapping $h{:}V \to U$ is that $h(0) = 0$ and $h(1) = 1$. In this framework $\Pi(X, V)$ is just the power set 2^X and the resulting utility functionals are

$$QU^-_W(A|u) = \min_{x \in A} u(x) \text{ and } QU^+_W(A|u) = \max_{x \in A} u(x),$$

leading to the well-known *maximin* and *maximax* decision models. Now, it is very easy to check that, in order to fully characterize a preference relation on 2^X induced by these QU^-_W and QU^+_W, the above axioms simplify to these ones:

- $A1$: $\sqsubseteq$ is a total pre-order,
- $A2$: if $A \subseteq B$ then $B \sqsubseteq A$,
- $A3$: if $A \sim B$ then $A \cup C \sim B \cup C$,
- $A4RC$: $\forall A \subseteq X, \exists x \in X$ s.t. $A \sim x$,
- $AxMix$: if $x \sqsubseteq y$ then $\{x, y\} \sim x$.

Actually, in this setting, axiom $A2$ is redundant since it is a logical consequence of the remaining axioms. Moreover, as we are working with a finite set X, $A4RC$ is a consequence of $AxMix$. The axiomatic frameworks à la Savage of these maximax and maximin criteria are provided in [Brafman and Tennenholtz, 1996; 1997].

Dubois et al. [1999] shown the necessity of extending the model to non-normalized possibility distributions, because of case-based decision-making problems or to be able of facing problems where different, partially inconsistent, information sources about the situation are available,etc. To include these problems, they extended the model summarized in Section 2. Analogously, we extend our above proposal and provide corresponding characterizations of the orderings induced by suitably modified utility functions.

3.6 Utilities for Non-Normalized Distributions

Now, we consider the set of *non-necessarily normalized distributions* on X, $\Pi^{ex}(X, V)$, with values on the finite uncertainty scale V (keeping the usual definition of possibilistic mixture) as the working set of possibilistic

lotteries. The utility functionals are extended to evaluate non-normalized distributions as well. Indeed, given a mapping $F{:}V \to V$ s.t. $F(1) = 0$, $\forall \pi \in \Pi^{ex}(X,V)$:

$$\underline{QU}^{-}_{W}(\pi|u) = \min\{QU^{-}_{W}(\mathcal{N}(\pi)|u), n \circ F(\mathcal{H}(\pi))\},$$

$$\underline{QU}^{+}_{W}(\pi|u) = \max\{QU^{+}_{W}(\mathcal{N}(\pi)|u), h \circ F(\mathcal{H}(\pi))\},$$

where $\mathcal{H}(\pi)$ is the *height of the distribution* π, $\mathcal{H}(\pi) = \max_{x \in X} \pi(x)$, and $\mathcal{N}(\pi)$ is a normalized version of π defined as 1, if $\pi(x) = \mathcal{H}(\pi)$, and as $\pi(x)$, otherwise. Notice that when $\mathcal{H}(\pi) = 1$, the original expression is retrieved.

To characterize the preference orderings induced by $\underline{QU}_W$ in $\Pi^{ex}(X,V)$ we extend AXM and AXM^{+} respectively, with the usual [Dubois *et al.*, 1999; Godo and Zapico, 2001] additional axiom:

- $A7F$: $\forall \pi \in \Pi^{ex}(X,V), \pi \sim (1/\mathcal{N}(\pi),\ F(\mathcal{H}(\pi))/X)$.

Obtaining the following representation theorems.

Theorem 3.7 *A preference relation $\sqsubseteq$ on $\Pi^{ex}(X,V)$ satisfies $AXM^{ex} = AXM + A7F$(resp. $AXM^{+ex} = AXM^{+} + A7F$) if, and only if, there exist*

(i) a linearly ordered and finite preference scale U with $\inf(U) = 0$ and $\sup(U) = 1$,

(ii) a preference function $u{:}X \to U$ s.t. $u^{-1}(1) \neq \emptyset \neq u^{-1}(0)$,

(iii) an order-preserving mapping $h{:}V \to U$, $h(0) = 0$ and $h(1) = 1$,

in such a way that $\pi' \sqsubseteq \pi$ iff $\underline{QU}^{-}_{W}(\pi'|u) \sqsubseteq \underline{QU}^{-}_{W}(\pi|u)$, ($\pi' \sqsubseteq \pi$ iff $\underline{QU}^{+}_{W}(\pi'|u) \sqsubseteq \underline{QU}^{+}_{W}(\pi|u)$ respectively).

4 Conclusions and Future Works

As it has been mentioned, to weaken the onto condition of the linking-scales mapping allows us to work with decision-making problems where the cardinality of the uncertainty values set may be not greater than or equal to the cardinality of the preference set.

It has been shown that this onto condition in the commensurability hypothesis is directly related with the continuity property of the preference relation on $\Pi(X,V)$. For characterizing those preference relations representable by the qualitative utility functions where h is not required to be onto, we have proposed to include a relaxed continuity axiom and an axiom for handling some special mixtures.

It is true, that given U and h non-onto, we may extend V to some V' and define $h' : V' \to U$, with h' onto, in a such a way that $\forall \pi \in \Pi(X,V), QU_W(\pi|h) = QU(\pi|h')$. But the point, for us, is that while the ordering induced by $QU(\cdot|h')$ in $\Pi(X,V')$ satisfies continuity, the ordering induced by $QU_W(\cdot|h)$ in the *original* distributions set $\Pi(X,V)$ may not satisfy continuity.

This is a first approach to the goal of weakening commensurability. A next step is to extend the analysis to the general utility functions proposed by Dubois et al. [1999]. However, this extension seems non-trivial since the onto conditions is required to guarantee the well-definedness of these generalized utility functions. Another point is to consider the utilities when the sets values of uncertainty and preferences are non-linear lattices [Zapico, 1999].

Acknowledgments

The author wishes to thank the valuable comments of Lluís Godo and of the anonymous referees .

References

[Allais, 1953] M. Allais. Le comportement de l'homme rationnel devant le risque: critique des postulats et axiomes de l' école américaine. *Econometrica*,(21):503–546,1953.

[Brafman and Tennenholtz, 1996] R.I. Brafman and M. Tennenholtz. On the foundations of qualitative decision criteria. In *13th Nat. Conf. on A.I.(AAAI'96)*, 1996.

[Brafman and Tennenholtz, 1997] R.I. Brafman and M. Tennenholtz. On the axiomatization of qualitative decision criteria. In *14th Nat. Conf. on A.I.(AAAI'97)*, 76–81, 1997.

[Doyle and Thomason, 1999] J. Doyle and R. Thomason. Background to qualitative decision theory. *AI Magazine*, 20(2):55–68, 1999.

[Dubois and Prade, 1995] D. Dubois and H. Prade. Possibility theory as a basis for QDT. In *14th Int. Joint Conf. on A.I. (IJCAI'95)*, 1924–1930, 1995.

[Dubois *et al.*, 1997] D. Dubois, H. Prade, and R. Sabbadin. Decision under qualitative uncertainty with Sugeno integrals: an axiomatic approach. In *7th Int. Fuzzy Syst. Assoc. Cong. (IFSA'97)*, vol. I, 441–446, 1997.

[Dubois *et al.*, 1998] D. Dubois, H. Prade, and R. Sabbadin. QDT with Sugeno integrals. In *14th Conf. on Uncertainty in A.I. (UAI'98)*, 121–128, 1998.

[Dubois *et al.*, 1999] D. Dubois, L.Godo, H. Prade, and A. Zapico. On the possibilistic-based decision model:From decision under uncertainty to case-based decision. *Int. J. of Uncert., Fuzz. and Know-Based Syst.*, 7(6): 631–670, 1999. *Short version in*:Making decision in a qualitative setting:from decision under uncertainty to case-based decision. In *6th Int. Conf. on Principles of Know. Repres and Reasoning (KR'98)*,594 – 605, 1998.

[Godo and Zapico, 2001] L.Godo and A.Zapico. On the Possibilistic-Based Decision Model: Characterisation of Preferences Relations under Partial Inconsistency. In *Applied Intelligence*,14(3):319–333, 2001.

[Ellsberg, 1961] D. Ellsberg. Risk, ambiguity and the Savage axioms. *Quarterly J. of Economics*, 75:643–669, 1961.

[Fargier and Perny, 1999] H. Fargier and P. Perny. Qualitative models for decision under uncertainty without the commensurability assumption. In *15th Conf. on Uncertainty in A.I. (UAI'99)*, 188–195, 1999.

[Savage, 1972] L. J. Savage. *The Foundations of Statistics*. Dover, New York, 1972.

[von Neumann and Morgenstern, 1944] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton Univ. Press, 1944.

[Zapico, 1999] A. Zapico. Axiomatic foundations for qualitative/ordinal decisions with partial preferences. In *16th Int. Joint Conf. on A.I. (IJCAI'99)*, 1999.

A Fuzzy Modal Logic for Belief Functions

Lluís Godo
AI Research Institute
IIIA - CSIC
08193 Bellaterra, Spain
godo@iiia.csic.es

Petr Hájek
Institute of Computer Science
Academy of Sciences,
182 07 Prague, Czech Republic
hajek@cs.cas.cz

Francesc Esteva
AI Research Institute
IIIA - CSIC
08193 Bellaterra, Spain
esteva@iiia.csic.es

Abstract

In this paper we introduce a new logical approach to reason explicitly about Dempster-Shafer belief functions. We adopt the following view: one just starts with Boolean formulas φ and a belief function on them; the belief of φ is taken to be the truth degree of the (fuzzy) proposition $B\varphi$ standing for "φ is believed". For our complete axiomatization (Hylbert-style) we use one of the possible definitions of belief, namely as probability of (modal) necessity. This enables us to define a logical system combining the modal logic S5 with an already proposed fuzzy logic approach to reason about probabilities. In particular, our fuzzy logic is the logic Ł$\Pi\frac{1}{2}$ which puts Lukasiewicz and Product logics together.

1 Introduction

Reasoning under uncertainty is a key issue in many areas of Artificial Intelligence. From a logical view point, uncertainty basically concerns formulas, describing a situation, that can be either true or false, but their truth value is unknown due to incompleteness of the available information. When this uncertainty can be measured, then one may assign to propositions the degree of belief of being true. Among the different models of numerical uncertainty, in the sense of belief, that have been proposed, probability theory is undoubtedly the most relevant, however other interesting models have been also proposed trying to remedy different shortcomings of the pure Bayesian approaches. In particular, Demspter-Shafer belief functions [Shafer, 1975] provide a more general framework to assess uncertainty to events or propositions, generalizing not only the probabilistic model but also for instance the possibilistic model. In this framework propositions can be attached two uncertainty degrees, a lower and an upper estimate, and the lack of evidence on the truth of a proposition does not necessarily induce evidence for the falsity of that proposition.

One may find in the literature a number of uncertainty logics. As for probability, let us mention [Keisler, 1985] for a deep investigation of probabilistic quantifiers (probability of φ is $\geq \alpha$, α a fixed parameter) and [Fagin *et al.*, 1990; Halpern, 1989] for logics to reason explicitly about probabilities; see also [Nilsson, 1982; Bacchus, 1990] for other probabilistic logical formalisms. As for other types of belief, see e.g. [Dubois *et al.*, 1994] for a full overview on possibilistic logic, and [Fagin and Halpern, 1991; Saffiotti, 1992; Sossai *et al.*, 1999] for different proposals of belief function logics. Another type of approaches, that may be called comparative, deal with a binary modality $\triangleright$, the formula $\varphi \triangleright \psi$ meaning "belief of φ is $\leq$ belief of ψ"; this was adopted e.g. in [Boutilier, 1992; Bendova and Hájek, 1993] (possilibity theory) and [Harmanec and Hájek, 1994] (Dempster-Shafer belief). Let us remark that, as it is obvious, the above list of references of relevant works is far from being exhaustive, it is only for exemplifying purposes.

It is of worth noticing that all the above uncertainty logics (and in many others) are built on the basis that the representation of situations is done by means of two-valued (Boolean) propositions: they can only be either true or false. However, when we need to deal with propositions representing gradual notions (e.g. high temperature, low income) it is natural to move from two-valued classical logic to a many-valued logical framework where propositions may be attached intermediate truth degrees. This is the realm of fuzzy logic[1], which is completely different from the one of uncertainty logics, although in both propositions are usually attached with numbers between 0 and 1. It has been repeatedly stressed that truth degrees in fuzzy logic must be carefully and clearly distinguished from belief degrees (probabilities, or more generally belief degrees of Dempster–Shafer theory). Fuzzy logics are logics of *comparative truth* and are mainly understood as *truth-functional:* the truth degree of a compound formula (conjunction, disjunction, implication,...) is determined by the truth degrees of its components via truth functions. On the other hand, uncertainty is basically concerned with degrees of belief that two-valued propositions are true. Belief degrees are obviously not truth-functional. This does not prevent to find proposals to extend the various notions of uncertainty to fuzzy propositions and hence to find in the literature also *fuzzy belief logics* of various kinds, for instance let us mention the paper [Hájek *et al.*, 1994] fuzzifying the comparative modality $\triangleleft$ for possibilistic measures over a finitely

[1] Fuzzy logic has been developed as a strictly formal system in both propositional and predicate calculi, see e.g. [Hájek, 1998].

valued Łukasiewicz logic).

In this paper we introduce a new logical approach to reason explicitly about Dempster-Shafer belief functions which combines classical uncertainty measures (probability and belief functions) with elements of fuzzy logic: the basic observation is that "uncertainty" or belief is itself a gradual notion, e.g. a proposition may be totally, quite, more or less, or slightly certain (in the sense of probable, possible, believable, plausible, etc.). Then, one just starts with Boolean formulas φ and a probability on them; the probability of φ is taken to be the truth degree of the fuzzy proposition $P\varphi$ = "φ is probable". This was already used in [Hájek *et al.*, 1995] and continued in [Hájek, 1998; Godo *et al.*, 2000] to define a fuzzy logic theory for probabilistic reasoning. Here we generalize this approach to deal with belief functions, i.e. we take the following identity

$$\text{belief degree of } \varphi = \text{truth degree of } B\varphi,$$

as our working assumption, where $B\varphi$ stands for the fuzzy proposition "φ is believed". For getting a complete axiomatization of "being (highly) believed" we use one of possible definitions of Dempster-Shafer belief, namely as probability of necessity. This enables us to combine the approach of [Godo *et al.*, 2000] with the modal logic S5. In particular, our fuzzy logic is the logic Ł$\Pi\frac{1}{2}$ "putting Łukasiewicz and Product logic together", which provides a nice and powerful logical setting where arithmetical operations are implicitly available.

We are forced to have extensive preliminaries: Section 2 contains preliminaries on belief functions and modal logic S5, Section 3 preliminaries on the logic Ł$\Pi\frac{1}{2}$ and a variant of it suitable to handle infinite theories. Section 4 is devoted to the definition of our fuzzy belief logic FB(Ł$\Pi\frac{1}{2}$) for regular belief functions and main (completeness) results, while Section 5 contains an approach to handle Dempster combination rule in our fuzzy logic setting. Finally, Section 6 discusses a generalization to deal with general belief functions, i.e. belief functions bel with possibly $bel(\emptyset)$ positive. Due to space limitation proofs are not included here but can be found in [Godo *et al.*, 2001].

2 Preliminaries I

We survey here some basic facts on belief functions. (The basic monograph is [Shafer, 1975].) We are interested in belief functions on propositional formulas in finitely or countably infinitely many propositional variables. Starting from the classical definition of Dempster (using what we call Dempster spaces) we explain a presentation using Kripke models of modal logic as well as a definition using a superadditivity condition. All results of this section, except the last one to the best of our knowledge, are already known modulo possibly a different notation.

Definition 1 *A* Dempster space *(or d-space) is a structure* $\mathbf{D} = (E, W, \Gamma, \mu)$ *where* $E \neq \emptyset \neq W$, Γ *is a mapping of* E *into the power set* $\mathbf{P}(W)$ *of* W *and* μ *is a finitely additive probability on an algebra of subsets of* E. *For* $X \subseteq W$, *put* $bel_{\mathbf{D}}(X) = \mu\{w \mid \Gamma(w) \subseteq X\}$ *(if this set is measurable).* $bel_{\mathbf{D}}$ *is the belief function given by* **D**. **D** *is* regular *if* $\Gamma(w) \neq \emptyset$ *for each* w; *then* $bel_{\mathbf{D}}$ *is a regular belief function* $(bel_{\mathbf{D}}(\emptyset) = 0)$.

Needless to say, a function bel mapping (some) subsets of W into $[0, 1]$ is a belief function iff it is a belief function given by some d-space. Note that it may be given by various d-spaces; only properties independent of the choice of the underlying d-space are of interest.

Let *Var* be a set of propositional variables $p_0, p_1, \ldots$ and let $e : (W \times Var) \to \{0, 1\}$ (value of a propositional variable in a world). The pair $(\mathbf{D}, e)$ (or just (E, W, Γ, μ, e) is an *evaluated* d-space. For each formula φ built from our propositional variables $e(w, \varphi)$ is the truth value of φ in w defined in the obvious way. The belief of φ is defined as

$$bel_{\mathbf{D},e}(\varphi) = bel_{\mathbf{D}}\{w \mid e(w, \varphi) = 1\}.$$

A function bel is a *belief function on formulas* if for some evaluated d-space $(\mathbf{D}, e)$, $bel = bel_{(\mathbf{D},e)}$. Actually we shall deal only with total belief functions on formulas, i.e. with those belief functions for which the sets of worlds $\{w \in W \mid \Gamma(w) \subseteq \{w' \mid e(w', \varphi) = 1\}\}$ are μ-measurable for any formula φ. Evidently, if φ and ψ are logically equivalent then $bel_{(D,e)}(\varphi) = bel_{(D,e)}(\psi)$, i.e. bel respects logical equivalence.

Now let us introduce some basic concepts of modal logic. Formulas of modal propositional logic are built form propositional variables using connectives and the modality $\Box$ (necessarily); if φ is a formula then $\Box\varphi$ is a formula. A Kripke model for the logic $S5$ (briefly a $\Box$-Kripke model) is a structure $\mathbf{K} = (W, R, e)$ where R is an equivalence relation on W (reflexive, symmetric and transitive) and e is an evaluation of propositional variables as above. The truth value of $\Box\varphi$ is defined as follows: $e(w, \Box\varphi) = 1$ if for each w', wRw' implies $e(w', \varphi) = 1$. A formula φ is an $S5$-tautology if $e(w, \varphi) = 1$ for each $\Box$-Kripke model $\mathbf{K} = (W, R, e)$ and each $w \in W$ (φ is true in each world of each $\Box$-Kripke model). The following are axioms of $S5$: Axioms of classical propositional logic plus

(K) $\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$
(T) $\Box\varphi \to \varphi$
(4) $\Box\varphi \to \Box\Box\varphi$
(B) $\varphi \to \Box\Diamond\varphi$

where $\Diamond\varphi$ is read "possibly φ" and stands for $\neg\Box\neg\varphi$. Deduction rules are modus ponens and necessitation (from φ deduce $\Box\varphi$). $S5$ is *complete* with respect to the given semantics; a formula is provable in $S5$ iff it is a $S5$-tautology.

A formula is *without nested modalities* if no box occurs in the scope of other box (box is applied only to box-free formulas). In $S5$, each formula is logically equivalent to a formula without nested modalities. This, together with the known normal form theorem of propositional logic, gives the following important fact.

Fact. If the set *Var* of propositional variables is finite then there is a finite set Θ of formulas such that each formula is, over $S5$, logically equivalent to a formula from Θ.

A $\Box$-*probabilistic Kripke model* is a pair $(\mathbf{K}, \mu)$, or just

(W, R, e, μ), where $\mathbf{K}$ is a $\Box$-Kripke model and μ is a probabilistic (finitely additive) measure on an algebra of subsets of W. $(\mathbf{K}, \mu)$ can be seen as an evaluated d-space $(\mathbf{D}, e) = (W, W, \Gamma, \mu, e)$ where $\Gamma(w) = \{w' \in W \mid wRw'\}$ (the R-class of w). Observe that then $bel_{(\mathbf{D},e)}(\varphi) = \mu(\{w \mid e(w, \Box\varphi) = 1\})$, or briefly $bel(\varphi) = \mu(\Box\varphi)$, which tells us that the belief of φ is the probability of the necessity of φ.

Note that the belief function given by a $\Box$-probabilistic Kripke model is regular, thanks to reflexivity of R. General (regular and non-regular) belief functions may be obtained e.g. from probabilistic Kripke models of *provability logic* GL, see [Hájek, 1995]. We shall come back to general belief functions later in Section 6.

Now assume W be finite non-empty. A *basic belief assignment* on W is a mapping $m : \mathbf{P}(W) \to [0, 1]$ such that $\Sigma_{X \subseteq W} m(X) = 1$; m is *regular* if $m(\emptyset) = 0$. The *belief function* given by m is defined by

$$bel_m(X) = \sum_{Y \subseteq X} m(Y).$$

This is indeed a belief function; moreover, given an evaluation $e : W \times Var \to \{0, 1\}$, the corresponding belief function on formulas, $bel_{m,e}(\varphi) = bel_m(\{w \mid e(w, \varphi) = 1\})$, is regular iff it is given by a $\Box$-probabilistic Kripke model; namely by a model $\mathbf{K} = (W', e', R, \mu)$ where

(1) $W' = \{(T, w) \mid T \subseteq W, w \in W\}$,
(2) $e'((T, w), p_i) = e(w, p_i)$,
(3) $(T, w)R(S, w)$ iff $T = S$ and
(4) μ is any probability on W' such that $\mu(\{(T, w) \mid w \in T\}) = m(T)$.

This construction is from [Harmanec *et al.*,]. Conversely, each belief function bel on a finite set W is given by a basic belief assignment defined as follows [Shafer, 1975]:

$$m(A) = \Sigma_{I \subseteq A} (-1)^{|A-I|} bel(I).$$

Note that if *Var* (the set of propositional variables) is finite and bel is a belief function on the formulas built from *Var*, then we may assume that bel is given by a d-space with $W = 2^n$ (the set of all n-tuples of zeros and ones). Then if regular, bel is given by a finite $\Box$-probabilistic Kripke model.

Let $(\mathbf{D}, e)$ be an evaluated d-space. For each n and each tuple of formulas $\varphi_1, \ldots, \varphi_n$, the belief function bel given by $(\mathbf{D}, e)$ satisfies the inequality (see[Shafer, 1975]):

$$bel(\varphi_1 \vee \ldots \vee \varphi_n) \geq \sum_{\emptyset \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} bel(\bigwedge_{i \in I} \varphi_i),$$

We shall call it the *generalized super-additivity* condition, GSA for short, since it reduces to the usual super-additivity condition for $n = 2$ and when φ_1 and φ_2 are mutually exclusive. Moreover, if Var is finite and bel is a function assigning numbers from $[0, 1]$ to formulas such that

(1) $bel(true) = 1$,
(2) $bel(false) = 0$,
(3) bel respects logical equivalence, and
(4) bel satisfies the GSA condition,

then bel is a regular belief function and the formula above defines a regular basic belief assignment defining bel. Consequently, bel is given by a finite $\Box$-probabilistic Kripke model.

Our last aim in this section is to prove that this result can be extended to the case of *Var* being countable.

Theorem 1 *If Var is countable and bel is a function satisfying $bel(true) = 1$, $bel(false) = 0$, the superadditivity condition (SA) and bel respects logical equivalence then there is a $\Box$-probabilistic Kripke model* $\mathbf{K}$ *defining bel.*

An sketch of the proof goes as follows. Let $Var = \{p_1, p_2, \ldots, p_n, \ldots\}$, and for each n, let F_n be the set of formulas in the variables $p_1, \ldots, p_n$. Let bel_n the restriction of bel to F_n and let $K_n = (W_n, e_n, R_n, \mu_n)$ be the corresponding $\Box$-probabilistic Kripke model. Assume the sets W_n to be pairwise disjoint and define the structure $\mathbf{K} = (W, e, R, \mu)$ as follows:

1. $W = \bigcup_n W_n$.
2. $e(w, p_i) = e_n(w, p_i)$, if $w \in W_n$ and $i \leq n$; $e(w, p_i) = 0$, otherwise.
3. $R = \bigcup_n R_n$.
4. for each $X \subseteq W$, put $\mu(X) = c$ if there exists n_0 such that $\mu_n(X \cap W_n) = c$ for all $n \geq n_0$, otherwise we let $\mu(X)$ undefined. Observe that trivially $\mu(\emptyset) = 0$ and $\mu(W) = 1$.

Then one can prove that $\mathbf{K} = (W, e, R, \mu)$ is a $\Box$-probabilistic Kripke model such that $\mu(\{w \mid w \models_K \Box\varphi\}) = bel(\varphi)$ for each modality-free formula φ.

This result is in the same spirit as the one by Fagin and Halpern [Fagin and Halpern, 1991, Theorem 3.3] connecting belief functions and inner measures induced by probability measures when the set *Var* of propositional variables is finite.

3 Preliminaries II

Here we survey the underlying system of fuzzy logic we shall use for our belief function logic.

Ł$\Pi\frac{1}{2}$ is a logic "putting Łukasiewicz and Product logics together", introduced in [Esteva *et al.*, 2000]. Formulas are built from propositional variables and the truth constants $\bar{0}$ and $\overline{\frac{1}{2}}$ using connectives $\to_{Ł}, \to_{\Pi}, \odot$ (Łukasiewicz and Goguen implication and product conjunction) with the following truth functions[2]:

$$\begin{array}{rcl} e(\overline{0}) = 0 \quad , \quad e(\overline{\frac{1}{2}}) = \frac{1}{2} & & \\ e(\varphi \to_{Ł} \psi) & = & \min(1 - e(\varphi) + e(\psi), 1) \\ e(\varphi \odot \psi) & = & e(\varphi) \cdot e(\psi) \\ e(\varphi \to_{\Pi} \psi) & = & \begin{cases} 1, & \text{if } e(\varphi) \leq e(\psi) \\ e(\psi)/e(\varphi), & \text{otherwise} \end{cases} \end{array}$$

Evaluations of propositional variables e into [0, 1] which are extended to arbitrary formulas using these rules will be called Ł$\Pi\frac{1}{2}$-evaluations. Other definable connectives are:

[2]Note that here we work only with the standard semantics, i.e. our set of truth degrees is the unit real interval $[0, 1]$.

$$
\begin{array}{lll}
\neg_{\text{Ł}}\varphi & \text{is} & \varphi \rightarrow_{\text{Ł}} \bar{0},\\
\varphi \oplus \psi & \text{is} & \neg_{\text{Ł}}\varphi \rightarrow_{\text{Ł}} \psi,\\
\varphi\&\psi & \text{is} & \neg_{\text{Ł}}(\neg_{\text{Ł}}\varphi \oplus \neg_{\text{Ł}}\psi),\\
\varphi \equiv_{\text{Ł}} \psi & \text{is} & (\varphi \rightarrow_{\text{Ł}} \psi)\&(\psi \rightarrow_{\text{Ł}} \varphi),\\
\varphi \wedge \psi & \text{is} & \varphi\&(\varphi \rightarrow_{\text{Ł}} \psi),\\
\varphi \vee \psi & \text{is} & \neg_{\text{Ł}}(\neg_{\text{Ł}}\varphi \wedge \neg_{\text{Ł}}\psi),\\
\neg_{\Pi}\varphi & \text{is} & \varphi \rightarrow_{\Pi} 0,\\
\Delta\varphi & \text{is} & \neg_{\Pi}\neg_{\text{Ł}}\varphi,
\end{array}
$$

with the following truth functions associated (as the reader easily verifies):

$$
\begin{array}{lcl}
e(\neg_{\text{Ł}}\varphi) & = & 1 - e(\varphi),\\
e(\varphi \oplus \psi) & = & \min(1, e(\varphi) + e(\psi))\\
e(\varphi\&\psi) & = & \max(0, e(\varphi) + e(\psi) - 1)\\
e(\varphi \equiv_{\text{Ł}} \psi) & = & 1 - |e(\varphi) - e(\psi)|\\
e(\varphi \wedge \psi) & = & \min(e(\varphi), e(\psi))\\
e(\varphi \vee \psi) & = & \max(e(\varphi), e(\psi))\\
e(\neg_{\Pi}\varphi) & = & \begin{cases} 1, & \text{if } e(\varphi) = 0\\ 0, & \text{otherwise}\end{cases}\\
e(\Delta\varphi) & = & \begin{cases} 1, & \text{if } e(\varphi) = 1\\ 0, & \text{otherwise}\end{cases}
\end{array}
$$

Definition 2
(i) ŁΠ is the logical system whose axioms and rules are as follows[3]*:*

(Ł)	*Axioms of Łukasiewicz logic (for* $\rightarrow_{\text{Ł}}, \&, \bar{0}$*);*
(Π)	*Axioms for product logic (for* $\rightarrow_{\Pi}, \odot, \bar{0}$*);*
(¬)	$\neg_{\Pi}\varphi \rightarrow_{\text{Ł}} \neg_{\text{Ł}}\varphi$
(Δ)	$\Delta(\varphi \rightarrow_{\text{Ł}} \psi) \equiv_{\text{Ł}} \Delta(\varphi \rightarrow_{\Pi} \psi)$
(ŁΠ5)	$\varphi \odot (\psi \ominus \chi) \equiv_{\text{Ł}} (\varphi \odot \psi) \ominus (\varphi \odot \chi)$

The deduction rules are modus ponens and necessitation for Δ*: from* φ *infer* $\Delta\varphi$*.*
(ii) $\text{ŁΠ}\frac{1}{2}$ *is the logic obtained from ŁΠ by adding a truth constant* $\overline{\frac{1}{2}}$ *together with the axiom:* $\overline{\frac{1}{2}} \equiv_{\text{Ł}} \neg_{\text{Ł}}\overline{\frac{1}{2}}$*.*

In this setting, a theory T is just a set of formulas. A $\text{ŁΠ}\frac{1}{2}$-evaluation e is a model of T if $e(\varphi) = 1$ for each $\varphi \in T$. The notion of proof in $\text{ŁΠ}\frac{1}{2}$ is as usual.

Theorem 2 $\text{ŁΠ}\frac{1}{2}$ *is strongly complete for finite theories with respect to the given semantics. That is, if* T *is finite then* $T \vdash \varphi$ *iff* $e(\varphi) = 1$ *for any evaluation* e *model of* T*.*

It is interesting to remark that, as shown in [Esteva *et al.*, 2000], rational truth constants $\bar{r}$ (for each rational $r \in [0, 1]$) are definable in $\text{ŁΠ}\frac{1}{2}$ and the logic proves their basic properties. Thus the following definition is sound.

Definition 3 *Rational* ŁΠ *logic,* RŁΠ *for short, is the logic obtained from* $\text{ŁΠ}\frac{1}{2}$ *by adding the following* infinitary *rule:*

from $\varphi \rightarrow_{\Pi} \overline{r}$*, for each* $r > 0$*, derive* $\varphi \rightarrow_{\Pi} \overline{0}$*.* *(IR)*

The notion of proof in RŁΠ is as follows. If T is a theory over RŁΠ, then the set $C_{RL\Pi}(T)$ of all provable formulas from T is the smallest theory T' containing T as a subset, containing all axioms of RŁΠ and closed under all deduction rules. For simplicity we shall denote $\varphi \in C_{RL\Pi}(T)$ by $T \vdash^{\infty} \varphi$. We shall keep the notation $T \vdash \varphi$ to denote finitary deduction, i.e. deduction under $\text{ŁΠ}\frac{1}{2}$.

[3]These axioms are actually a simplified formulation of [Esteva *et al.*, 2000] proposed by P. Cintula in forthcoming paper.

Theorem 3 (Strong completeness) *For any theory* T *over* RŁΠ *and any formula* φ*,* $T \vdash^{\infty} \varphi$ *iff* φ *is 1-true in all [0, 1]-valued models of* T*.*

To close this section let us mention that in [Godo *et al.*, 2000] the logics $\text{ŁΠ}\frac{1}{2}$ and RŁΠ have been used to build the corresponding fuzzy probability logics FP($\text{ŁΠ}\frac{1}{2}$), FP(RŁΠ). In the next section we shall elaborate the corresponding belief logics FB($\text{ŁΠ}\frac{1}{2}$), FB(RŁΠ).

4 A logic for belief functions

After all these preliminaries we are prepared to define our logic. We distinguish two kinds of *formulas*:

(1) *Boolean formulas* (or S5-formulas) are built from propositional variables $p_0, p_1, \ldots$ using connectives $\rightarrow, \vee, \wedge, \neg$ (say) and the modality $\Box$ as described above. *Modality-free* formulas do not contain $\Box$; a formula is *closed* if each propositional variable is in the scope of a modality. Closed formulas without nested modalities are called *normal;* they are just built form formulas of the form $\Box\varphi$, φ modality free using the connectives above.
(2) *P-formulas* (or many-valued formulas). Atomic P-formulas have the form $P(\alpha)$, read "probably α", where α is a normal S5-formula. P-formulas are built from atomic P-formulas using the connectives of $\text{ŁΠ}\frac{1}{2}$, i.e. $\rightarrow_{\text{Ł}}, \rightarrow_{\Pi}, \odot$, and truth constants $\overline{0}$ and $\overline{\frac{1}{2}}$. We shall use $B\varphi$ for $P(\Box\varphi)$ where φ is a modality-free formula; $B\varphi$ reads "φ is believed". Formulas built from these using $\text{ŁΠ}\frac{1}{2}$-connectives and truth-constants are called *belief formulas* or B-formulas; they are of course particular P-formulas.

Examples: $(p \wedge q) \rightarrow r$ is a Boolean, modality free formula, $\Box(p \wedge q) \rightarrow \Box r$ is a normal S5-formula, $P(\Box(p \wedge q) \rightarrow \Box r)$ is a P-formula, and $P(\Box(p \wedge q)) \rightarrow_{\text{Ł}} P(\Box r)$ is a B-formula, equivalently expressed as $B(p \wedge q) \rightarrow_{\text{Ł}} B(r)$.

From a knowledge representation point of view, we want to point out the rich expressivity B-formulas provide with. As a matter of fact, we can express for instance simple numerical assignments like $bel(\varphi) \geq 0.4$ and $bel(\psi) = 0.6$ by formulas $\overline{0.4} \rightarrow_L B\varphi$ and $\overline{0.4} \equiv_L B\varphi$; a positive belief $bel(\varphi) > 0$ by $\neg_{\Pi}\neg_{\Pi}B\varphi$; comparisons of beliefs like $bel(\varphi) \leq bel(\psi)$ by the formula $B\varphi \rightarrow_L B\psi$; or even more complex conditions like $bel(\eta) \geq 0.4 \cdot bel(\varphi) + bel(\chi) \cdot bel(\psi)$ by the formula $((\overline{0.4} \odot B\varphi) \oplus (B\chi \odot B\psi)) \rightarrow_L B\eta$.

Semantics. Our language is interpreted in the obvious way in $\Box$-probabilistic Kripke models $\mathbf{K} = (W, R, e, \mu)$. For each Boolean φ and each $w \in W$, $e(w, \varphi)$ is defined (and is 0 or 1). Moreover, for each normal S5-formula α, the probability $\mu(\alpha) = \mu(\{w | e(w, \alpha) = 1\})$ is well-defined; then the truth value of $P(\alpha)$ in $\mathbf{K}$ is defined as

$$||P(\alpha)||_{\mathbf{K}} = \mu\{w | e(w, \alpha) = 1\}.$$

This extends to other P-formulas using truth connectives of $\text{ŁΠ}\frac{1}{2}$; thus $||\Phi||_{\mathbf{K}}$ is well defined for each P-formula Φ. In particular, $||B\varphi||_{\mathbf{K}}$ is defined for each modality free formula φ.

An obvious *alternative semantics* for B-formulas is given by an arbitrary regular belief function *bel* on modality-free formulas; take $||B\varphi||_{bel} = bel(\varphi)$ and extend using truth functions of connectives. Since each belief function on formulas is given by a $\Box$-probabilistic Kripke model this reduces to the former case.

A P-formula Φ is a *1-tautology* (or just tautology) of FB(Ł$\Pi\frac{1}{2}$) if $||\Phi||_{\mathbf{K}} = 1$ for each $\Box$-probabilistic Kripke model $\mathbf{K}$. It follows using [Hájek, 1998] that the following formulas are tautologies, for any normal S5-formulas α, β :

(FP1) $(P(\alpha)\&P(\alpha \to \beta)) \to_{\text{Ł}} P(\beta)$
(FP2) $P(\neg\alpha) \equiv_{\text{Ł}} \neg_{\text{Ł}} P(\alpha)$
(FP3) $P(\alpha \vee \beta) \equiv_{\text{Ł}} (P(\alpha) \to_{\text{Ł}} P(\alpha \wedge \beta)) \to_{\text{Ł}} P(\beta)$.

These three axioms axiomatize the (fuzzy) notion of being *probable*. Axiom (FP1) says that if both α and $\alpha \to \beta$ are probable, then β is probable as well. Axiom (FP2) says that α is probable iff $\neg\alpha$ is not probable. Finally, it is easy to check, by means of the truth function of $\to_L$, that Axiom (FP3) expresses the condition that the probability of $\alpha \vee \beta$ is the sum of the probabilities of α and β minus the probability of $\alpha \wedge \beta$.

Definition 4
Axioms *of FB(Ł$\Pi\frac{1}{2}$) and FB(RŁΠ) are the following:*

(i) α*, for each S5-formula* α *provable in S5*

(ii) $P(\alpha)$*, for each normal S5-formula* α *provable in S5 (call this axiom (FB0))*

(iii) The schemes (FP1)–(FP3) above for P-formulas

(iv) Axioms of Ł$\Pi\frac{1}{2}$ for P-formulas.

Deduction rules of FB(Ł$\Pi\frac{1}{2}$) are those of Ł$\Pi\frac{1}{2}$ (modus ponens and $\triangle$-necessitation) and deduction rules of FB(RŁΠ) are those of RŁΠ (the above plus the infinitary rule (IR)).

This is a recursive axiom system since provability in S5 is decidable. Alternatively one may take axioms and rules of S5 (including classical propositional logic) for S5-formulas and add the rule "from α infer $P(\alpha)$" for a normal S5-formula, in this way one can obtain a system with finitely-many axiom schemes and rules.

Interestingly, one can easily check that FB(Ł$\Pi\frac{1}{2}$) proves the following formulas (φ_i being modality free Boolean):

(1) $P(\Box\varphi_1 \vee \ldots \vee \Box\varphi_n) \to_{\text{Ł}} B(\varphi_1 \vee \ldots \vee \varphi_n)$
(2) $B(\overline{1})$
(3) $\neg B(\overline{0})$

Moreover, if φ, ψ are Boolean modality free, then

$$\frac{\varphi \to \psi}{B\varphi \to_{\text{Ł}} B\psi}$$

is a derived rule in FB(Ł$\Pi\frac{1}{2}$). The reader may notice that these theorems and rule correspond to the four properties characterizing belief functions on formulas (see Section 2), in particular (i) above corresponds to the GSA condition. Indeed, in semantic terms (i) expresses the inequality $bel(\varphi_1 \vee \ldots \vee \varphi_n) \geq \mu(\Box\varphi_1 \vee \ldots \vee \Box\varphi_n)$, but by the inclusion-exclusion rule for probabilities the last term is $\geq \sum_{\emptyset \neq I \subseteq \{1\ldots n\}} (-1)^{|I|+1} \mu(\bigwedge_{i\in I} \Box\varphi_i)$, and since $\bigwedge_{i\in I} \Box\varphi_i$ is logically equivalent to $\Box(\bigwedge_{i\in i} \varphi_i)$, one has $\mu(\bigwedge_{i\in I} \Box\varphi_i) = bel(\bigwedge_{i\in i} \varphi_i)$.

After this preparation we can immediately state and prove two completeness theorems for *belief theories*, i.e. sets of B-formulas. Recall that $\Box$-probabilistic Kripke models that are models of a belief theory T are in the obvious correspondence with regular belief functions *bel* that are models of T, i.e. such that $||\Phi||_{bel} = 1$ for each axiom $\Phi \in T$.

Theorem 4 (Completeness)

(1) Strong completeness of FB(RŁΠ). Let T be a belief theory and Φ a belief formula. $T \vdash^{\infty} \Phi$ (i.e. T proves Φ over $RL\Pi$) iff $||\Phi||_{bel} = 1$ for each belief function *bel* which is a model of T (iff $||\Phi||_{\mathbf{K}} = 1$ for each $\Box$-probabilistic Kripke model of T).

(2) *Strong finite completeness of* FB(Ł$\Pi\frac{1}{2}$). Let T be a finite belief theory and Φ a belief formula. $T \vdash \Phi$ (T proves Φ over Ł$\Pi\frac{1}{2}$) iff $||\Phi||_{bel} = 1$ for each belief function *bel* which is a model of T.

5 About Dempster's rule

In this section we shall give some hints about how our logical framework is powerful enough to enable the modelling of *regular* belief functions combination by means of the well-known Dempster's rule, although in a bit cumbersome way and provided we work with a finite set $Var = \{p_1, \ldots, p_n\}$ of propositional variables.

Let us briefly recall that if m_1 and m_2 are two regular basic belief assignments on a finite W, then the (regular) basic belief assignment $m_1 \otimes m_2$ corresponding to the Dempster rule combination of m_1 and m_2 is defined for $\emptyset \neq X \subseteq W$ as

$$(m_1 \otimes m_2)(X) = k^{-1} \cdot \sum_{Y,Z:Y\cap Z=X} m_1(Y) \cdot m_2(Z),$$

where $k = \sum_{Y,Z:Y\cap Z=\emptyset} m_1(Y) \cdot m_2(Z)$, and $(m_1 \otimes m_2)(\emptyset) = 0$.

The key point is that, in a such a finite framework, given a belief function on formulas, we can access to its corresponding mass distribution. Similarly to the fact that the belief of a Boolean modality-free formula φ can be obtained as the probability of the modal formula $\Box\varphi$, the corresponding mass assignment for φ can be obtained as the probability of another related modal formula $\widehat{\Box}\varphi$. Indeed, let $\mathbf{K} = (W, e, R, \mu)$ be a $\Box$-probabilistic Kripke model and let bel_K be its induced belief function, i.e. $bel_K(\varphi) = ||P\Box\varphi||_K$. Call m_K its corresponding mass distribution. Assume φ to be decomposed in a disjunction of maximally elementary conjunctions (mec's for short), that is, $\varphi \equiv \vee_{\eta\in\lceil\varphi\rceil}\eta$, where each η is of the form

$$(\bigwedge_{i\in I} p_i) \wedge (\bigwedge_{j\in\{1,\ldots,n\}\setminus I} \neg p_j)$$

for some $I \subseteq \{1, \ldots, n\}$. Notice that since we have n propositional variables, we have 2^n mec's, call them $\eta_1, \ldots, \eta_{2^n}$. Then one can prove the following lemma.

Lemma 1 *For each modality free Boolean formula φ, let $\widehat{\Box}\varphi$ stand for*

$$(\Box\varphi) \wedge (\bigwedge_{\eta \in \lceil\varphi\rceil} \neg\Box(\varphi \wedge \neg\eta)).$$

Then $\|P\widehat{\Box}\varphi\|_K = m_K(\varphi)$.

Now, since we are able to identify mass distributions, and since Dempster's rule defines a new mass m_3 from two masses m_1 and m_2 with products and sums (available in the ŁΠ and RŁΠ logics), the rest is easy, although a bit cumbersome as already mentioned. One has to introduce three pairs of modalities $P_i, \Box_i$, for $i = 1, 2, 3$, and construct formulas in the obvious way. Belief formulas now will be built from atomic belief formulas of the form $P_i\Box_i\varphi$, called $B_i\varphi$, using connectives and truth-constants from ŁΠ$\frac{1}{2}$. Analogously to the simple case, we can restrict ourselves to belief Kripke models of the form $K = (W, e, bel_1, bel_2, bel_3)$, where the bel_i's are belief functions on 2^W. For each non-modal formula φ, let us denote by $(\widehat{B_1} \otimes \widehat{B_2})\varphi$, the following (cumbersome) formula:

$$\bigoplus_{I,J} P_1(\widehat{\Box}_1 \bigvee_{i\in I} \eta_i) \odot P_2(\widehat{\Box}_2 \bigvee_{j\in J} \eta_j),$$

where $I, J \subseteq \{1, \ldots, 2^n\}$ are such that $(\bigvee_{i\in I} \eta_i) \wedge (\bigvee_{j\in J} \eta_j) \equiv \varphi$. Notice that, due to Lemma 1, the truth value of $(\widehat{B_1} \otimes \widehat{B_2})\varphi$ in the model $\mathbf{K} = (W, e, bel_1, bel_2, bel_3)$ is

$$\|(\widehat{B_1} \otimes \widehat{B_2})\varphi\|_K = \sum_{X,Y\subseteq W : X\cap Y = [\varphi]_K} m_1(X) \cdot m_2(Y).$$

Then let FB$_3$(ŁΠ$\frac{1}{2}$) be like FB(ŁΠ$\frac{1}{2}$) but repeating FB(ŁΠ$\frac{1}{2}$) axioms for each pair of modalities and having

$$P_3\widehat{\Box}_3(\bigvee_{i\in I} \eta_i) \equiv_{Ł} \neg_{Ł}(\otimes_{1,2}(\overline{0})) \to_{\Pi} \otimes_{1,2}(\vee_{i\in I}\eta_i) \quad \text{(DR1)}$$

for each non-empty $I \subseteq \{1, \ldots, 2^n\}$ and

$$P_3\widehat{\Box}_3(\overline{0}) \equiv_{Ł} \overline{0} \quad \text{(DR2)}$$

as additional axioms; analogously with FB$_3$(RŁΠ) and FB(RŁΠ).

Corollary 1 *Let T be a belief theory over* FB$_3$(RŁΠ) *and let Φ be belief formula. Then, $T \vdash^{\infty} \Phi$ iff $\|\Phi\|_{(bel_1, bel_2, bel_3)} = 1$ for each triple of belief functions (bel_1, bel_2, bel_3) model of T and with $bel_3 = bel_1 \otimes bel_2$. In particular,*

$$T \vdash^{\infty} \overline{r} \to_{Ł} B_3\varphi \text{ iff } (bel_1 \otimes bel_2)(\varphi) \geq r,$$

for each triple $(bel_1, bel_2, bel_1 \otimes bel_2)$ model of T.

6 Final Remarks

In this final section we like to briefly comment about the extension of the approach developed in the previius sections to deal with "general" belief functions. By general we mean regular as well as singular, i.e. belief functions bel such that $bel(\emptyset)$ can be positive. A d-space as defined in Definition 1 yields general belief functions; if bel is singular (non-regular), its *regularization* is defined as

$$bel'(X) = (bel(X) - bel(\emptyset))/(1 - bel(\emptyset)).$$

There is no room for full details so we shall only sketch how an extension of our approach covering this can be done.

The main point is that when dealing with modal logic we have to allow possible worlds satisfying $\Box\bar{0}$ (where $\bar{0}$ is the truth constant for falsity), i.e. to allow for worlds w such that wRw' for no w'. There are several logics admitting this but to be able to generalize directly our proofs we have to keep the logic as near to S5 as possible. Our proposal is motivated semantically: call $\mathbf{K} = (W, R, e, \mu)$ a $\Box'$-probability Kripke model where W, e, μ are as above and R is a symmetric transitive relation, not necessarily reflexive. The modal logic whose models are structures of the form (W, R, e), where R is symmetric and transitive, is known as KB4, a weaker logic than S5 (where axiom T is dropped).

So swichting our base modal logic from S5 to KB4, the reader may go through Sections 2 and 4 generalizing from S5 to KB4 and from regular belief functions to general belief functions. Then define the logics FB$'$(ŁΠ$\frac{1}{2}$) and FB$'$(RŁΠ) modifying the definitions of FB-logics as follows: axioms sub (i) are all KB4-tautologies and sub (ii) all formulas $P(\alpha)$ where α is any normal KB4-tautology. Finally, the completeness Theorem 4 generalizes in the obvious way by replacing FB by FB$'$ and deleting all occurrences of the word "regular"). It is important to observe that our generalized fuzzy logic of belief functions can express the regularization of a singular belief function, namely one can check that $B'\varphi = P(\Diamond\bar{1}) \to_{\Pi} (P(\Diamond\bar{1})\&P(\Box\varphi))$ expresses the regularized belief of φ.

Finally let us comment how this general approach also covers general belief functions combination by means of Demspter's rule, actually in a simplified way. In such a setting, the (non-normalized) Dempster's combination of two masses m_1 and m_2 is defined for any $X \subseteq W$ as

$$(m_1 \otimes' m_2)(X) = \sum_{Y,Z : Y\cap Z = X} m_1(Y) \cdot m_2(Z),$$

where obviously now $(m_1 \otimes' m_2)(\emptyset)$ may be positive, even if m_1 and m_2 were regular. Then one can just follow the same approach as in Section 5 and define the logics FB$'_3$(ŁΠ$\frac{1}{2}$) and FB$'_3$(RŁΠ) as FB$_3$(ŁΠ$\frac{1}{2}$) and FB$_3$(RŁΠ) respectively but replacing axioms (DR1) and (DR2) just by this one

$$P_3\widehat{\Box}_3(\bigvee_{i\in I} \eta_i) \equiv_{Ł} (\widehat{B_1} \otimes \widehat{B_2})(\bigvee_{i\in I} \eta_i), \quad \text{(DR1')}$$

for each $I \subseteq \{1, \ldots, 2^n\}$, with the convention that if $I = \emptyset$, then $\bigvee_{i\in I} \eta_i$ is taken as the truth constant $\bar{0}$. Finally one can show that Corollary 2 still remains valid if we replace there normalized Demspter combination $\otimes$ by the non-normalized version $\otimes'$.

Acknowledgements.

Petr Hájek acknowledges partial support by the grant No. IAA 1030004 of the Grant Agency of the Academy of Sciences of the Czech Republic. Lluis Godo and Francesc Esteva acknowledges partial support of the bilateral project CSIC-CNR "Logics for Evidential Theory". The authors also thank the anonymous referees for their useful comments and criticisms.

References

[Bacchus, 1990] Bacchus F. *Representing and reasoning with probabilistic knowledge*. MIT-Press, Cambridge Massachusetts, 1990.

[Bendova and Hájek, 1993] Bendová, K., Hájek, P. Possibilistic logic as tense logic. In: (Piera et al., ed.) Proc. of QUARDET'93, Barcelona 1993.

[Boutilier, 1992] Boutillier C. Modal logics for qualitative possibility and beliefs, in: (Dubois et al., ed.) *Uncertainty in Artifical Intelligence VIII,* Morgan-KaufmannPubl. 1992, 17-24.

[Dubois *et al.*, 1994] Dubois D., Lang, J., Prade H. Possibilistic logic. In: (Gabbay et al., ed.) *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol. 3, 439–514. Oxford UP, 1994.

[Esteva *et al.*, 2000] Esteva F., Godo L. and Montagna F. The ŁΠ and ŁΠ$\frac{1}{2}$ logics: two complete fuzzy logics joining Łukasiewicz and product logic. *Archive for Mathematical Logic* 40, 39-67, 2000.

[Fagin and Halpern, 1991] Fagin R., Halpern J.Y. Uncertainty, Belief and Probability. *Computational Intelligence* 7, 160–173, 1991.

[Fagin *et al.*, 1990] Fagin R., Halpern J.Y., and Megiddo N. A Logic for Reasoning about Probabilities *Information and Computation* 87, 78–128, 1990.

[Godo *et al.*, 2000] Godo L., Esteva F., Hájek P. Reasoning about probabilities using fuzzy logic. *Neural Network World* 10 (2000) 811-824.

[Godo *et al.*, 2001] Godo L., Hájek P. and Esteva F. A fuzzy modal logic for belief functionsm (extended version). IIIA Research Report 2001-05, 2001. Available at http://www.iiia.csic.es/Publications/Reports/2001/.

[Hájek, 1998] Hájek P. *Metamathematics of fuzzy logic*, Kluwer 1998.

[Hájek, 1995] Hájek P. Getting Belief Functions from Kripke Models. *Int. J. General Systems*, 23, No. 3 (1995), 00–00.

[Hájek *et al.*, 1995] Hájek P., Godo L. and Esteva F. Fuzzy Logic and Probability. In Proc. of UAI'95, pp. 237-244, 1995.

[Hájek *et al.*, 1994] Hájek P., Harmancová D., Esteva F., Garcia P., Godo L. On modal logics of qualitative possibility in fuzzy setting. In Proc. UAI'94, Morgan-Kaufmann pp. 278-285, 1994.

[Halpern, 1989] Halpern J. Y. An analysis of First-Order Logics of Probability In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'89),* pp. 1375-1381, 1989.

[Harmanec and Hájek, 1994] Harmanec, D., Hájek, P. A qualitative belief logic. *Int. Journ. Uncertainty, fuzziness and Knowledge-based Systems* 2 (1994), 227-236.

[Harmanec *et al.*,] Harmanec D., Klir G.J. and Resconi G. On modal logic interpretation of Demspter-Shafer theory of evidence. *Int. J. of Intelligent Systems* 9, 941-951.

[Keisler, 1985] Keisler J. Probability quantifiers. In (J. Barwise and S. Feferman, ed.) *Model-theoretic logics* Springer-Verlag New York 1985, 539-556

[Nilsson, 1982] Nilsson N.J. Probabilistic Logic. *Artificial Intelligence* 28, 71–87, 1982.

[Shafer, 1975] Shafer G. *A mathematical theory of evidence*. Princeton Univ. Press 1975.

[Saffiotti, 1992] Saffiotti, A. A Belief Function Logic. Proc. of the 10th AAAI Conf. San Jose, CA, pp. 642-647, 1992

[Sossai *et al.*, 1999] Sossai C., P. Bison P., Chemello G. and Trainito N. Merging probability and possibility for robot localization. In IJCAI-99 Workshop on Reasoning with Uncertainty in Robot Navigation Stockholm (Sweden) August 2, 1999.

UNCERTAINTY AND PROBABILISTIC REASONING

PROBABILISTIC REASONING

IBAL: A Probabilistic Rational Programming Language

Avi Pfeffer
Division of Engineering and Applied Sciences
Harvard University
avi@eecs.harvard.edu

Abstract

In a rational programming language, a program specifies a situation faced by an agent; evaluating the program amounts to computing what a rational agent would believe or do in the situation. This paper presents IBAL, a rational programming language for probabilistic and decision-theoretic agents. IBAL provides a rich declarative language for describing probabilistic models. The expression language allows the description of arbitrarily complex generative models. In addition, IBAL's observation language makes it possible to express and compose rejective models that result from conditioning on the observations. IBAL also integrates Bayesian parameter estimation and decision-theoretic utility maximization thoroughly into the framework. All these are packaged together into a programming language that has a rich type system and built-in extensibility. This paper presents a detailed account of the syntax and semantics of IBAL, as well as an overview of the implementation.

1 Introduction

In a rational programming language, a program specifes a situation encountered by an agent; evaluating the program amounts to computing what a rational agent would believe or do in the situation. Rational programming combines the advantages of declarative representations with features of programming languages such as modularity, compositionality, and type systems. A system designer need not reinvent the algorithms for deciding what the system should do in each possible situation it encounters. It is sufficient to declaratively describe the situation, and leave the sophisticated inference algorithms to the implementors of the language.

One can think of Prolog as a rational programming language, focused on computing the beliefs of an agent that uses logical deduction. In the past few years there has been a shift in AI towards specifications of rational behavior in terms of probability and decision theory. This paper presents IBAL, a probabilistic rational programming language. IBAL, pronounced "eyeball", stands for Integrated Bayesian Agent Language. As its name suggests, it integrates various expects of probability-based rational behavior, including probabilistic reasoning, Bayesian parameter estimation and decision-theoretic utility maximization.

IBAL makes four main contributions. The first is a highly expressive language for representing probability models, that is significantly more expressive than previous languages. Second, IBAL integrates a language for probabilistic modeling, Bayesian learning and decision theory under a single coherent semantic framework. Third, it provides a unified inference engine for solving reasoning, learning and utility maximization problems, that generalizes algorithms for many standard kinds of models. Finally, IBAL is packaged together into a usable programming language with features such as a strong type system and built-in extensibility.

IBAL is designed with three kinds of users in mind. The first is the system modeler, who may not be an expert in probabilistic reasoning. For this type of user the basics of the language should be reasonably easy to learn, and it should be fairly easy to come up with decent models for many domains. This kind of user will benefit greatly from a good selection of libraries implementing standard kinds of models. Also, a good default inference algorithm is needed that can be expected to do reasonably well on a large number of models.

The second kind of user is a modeling expert, who understands well the inference algorithms for probabilistic reasoning. For the expert user, the language should provide the power to carefully tweak the model being used, and to control what inference algorithm is used to evaluate different parts of the model.

The third kind of user is the AI researcher, who may want to introduce new kinds of probabilistic models and new inference algorithms. IBAL makes it easy to introduce new models via libraries, and the implementation framework provides support for building new algorithms and extending the inference capabilities of the system.

Space limitations prevent a full description of both the language and the implementation in this paper. The next section provides a fairly detailed account of the language and its semantics. Section 4 presents an extended example, showing how the various components of IBAL can be used together to provide a declarative implementation of a fairly complex decision-theoretic agent. Section 5 presents an overview of the main features of the IBAL implementation.

2 Genealogy and Related Work

The most direct presucrsor to IBAL is the language of [Koller *et al.*, 1997], hereinafter referred to as KMP97. KMP97 is a Lisp-like language extended with a `flip` construct to describe random events. IBAL extends KMP97 in a number of powerful ways. IBAL's basic definition and expression language for describing generative probabilistic models is similar to KMP97, but significantly richer, particularly in its use of higher-order functions. In addition, IBAL uses observations to condition distributions, which allows it to easily describe a much richer class of models. IBAL also integrates decisions and learning into the framework, which are not provided by KMP97.

IBAL is also indebted to recent work integrating probabilistic and first-order representation languages. Two recent strands in that direction are relational probability models [Pfeffer *et al.*, 1999], and stochastic logic programs [Muggleton, 2000].

Other projects have tried to integrate an expressive modeling language with at least some aspect of learning or decision theory. Another project similar in spirit to IBAL is DTGolog [Boutilier *et al.*, 2000], which integrates decision theory into the Golog action caculus. It is based on logic-programming rather than the functional programming framework IBAL uses. Also, its roots can be traced back to Markov decision processes, while IBAL's roots are in Bayesian networks. As a result, the two systems have quite a different approach to inference; for example, there is no notion of variable elimination in DTGolog. Also, DTGolog does not integrate learning into the framework.

[McAllester, 2000] presents a language based on KMP97 for describing decision problems and the policies used by agents, and for calculating the expected utilities of the agents. Unlike IBAL, there is no attempt to solve the decision problem and compute the optimal policies. [Cumby and Roth, 2000] integrates learning and reasoning in an expressive, compositional language, but one that is not probabilistic.

3 The IBAL Language

IBAL provides a declarative language for describing probability distributions, parameter estimation problems and utility maximization problems. The top level language component in IBAL is a *block*. A block consists of a sequence of *declarations*. There are a number of kinds of declarations, including *definitions* stating how values of things are stochastically generated; *observations* stating that some property holds of generated values; *priors* describing the prior probability distributions over learnable parameters; *decisions* describing the decisions that an agent makes and the information it has; *rewards* describing the rewards an agent receives; and *pragmatics*, containing control knowledge on how to perform inference in a block. I will describe each of these in turn. For the sake of presentation I will present the semantics of the language incrementally, as I discuss each kind of declaration. However, the discussions of semantics are somewhat technical, and can be skipped on first reading.

3.1 Definitions

A *definition* states how the value associated with a name is generated. A *name* is a symbol such as x; a *chain* is a sequence of names such as $x.y.z$. A definition has the form $x = e$, where e is an *expression* with one of the forms:

(Constant)	$'s$ where s is a symbol
(Chain)	σ
(Function)	$\lambda(x_1, \ldots, x_n) \rightarrow e$
(Conditional)	`if` $e_1 \models \pi$ `then` e_2 `else` e_3 where π stands for a pattern, discussed below
(Dist)	`dist` $[p_1{:}e_1, \ldots, p_n{:}e_n]$ where the p_i are probabilities summing to 1
(Block)	$\{b\}$ where b is a block
(Application)	$e_0(e_1, \ldots, e_n)$

A *pattern* defines a condition that may be satisfied by a value. The notation $e \models \pi$ stands for the predicate that is true iff the value of expression e satisfies the pattern π. A pattern may specify equality to a constant, a chain, it may be the negation of a pattern, or the conjunction or disjunction of two patterns.

In addition to the above expression syntax, the language provides plenty of syntactic sugar, such as case statements and easier syntax for defining functions. There is also a type system, discussion of which will be omitted for lack of space.

The intuitive meaning of a definition $x = e$ is that it defines a stochastic experiment generating the value of x. Consider a simple example:

```
fair()   = dist [0.5 : 'h, 0.5 : 't]
biased() = dist [0.9 : 'h, 0.1 : 't]
pick()   = dist [0.5 : fair, 0.5 : biased]
coin     = pick()
x        = { y = coin() ; z = coin() }
```

Intuitively, `fair` and `biased` are functions that return either `'h` or `'t` with appropriate probabilities. The function `pick` is a higher-order function that returns either the fair or biased function. The value of `coin` is generated by applying pick; it is either `fair` or `biased`. The expression defining `x` is a block expression; `x` is a data structure with components `y` and `z`, both generated by applying the value of `coin`. Chains `x.y` and `x.z` are mutually dependent on whether `coin` is `fair` or `biased`, but they are conditionally independent given `coin`.

IBAL inherits from KMP97 the idea of using *laziness* to allow infinitely long experiments to be defined, only some of whose results may be needed. For example:

```
real() = { first = dist [0.5 : 'zero, 0.5 :'one]
           rest = real() }
less_than_half = real.first |= 'zero
```

Here, `real` defines a uniform distribution over real numbers between 0 and 1, represented by their binary expansion. Executing `real` involves an infinite recursion, but only the first bit is needed to determine the value of `less_than_half`, which is the value of a Boolean predicate that looks only at `real.first`. We can think of executing `real` lazily, to get the single bit needed.

I will now make these intuitions precise. There are four kinds of values in IBAL: (1) Symbols such as heads and true; (2) A special *undefined* value, denoted by $\perp$; (3) *Complex values* that consist of an *environment*, which maps names

to values; (4) *Closures*, denoted $(x_1, \ldots, x_n) \stackrel{\epsilon}{\rightarrow} e$ consisting of formal paramaters $x_1, \ldots, x_n$, body e and environment ϵ.

A chain can be viewed as representing a function on values. Formally, if σ is a chain and v a value, we define $\sigma(v)$ as follows: If σ is empty, $\sigma(v) = v$. Otherwise, if v is not complex, $\sigma(v) = \bot$. Otherwise, let σ be $x.\sigma'$, and v' be the value associated with x in v; then $\sigma(v) = \sigma'(v')$. To determine whether a value satisfies a pattern, we need an environment assigning values to the variables appearing in the pattern. Formally, we define $v \models^\epsilon \pi$, meaning that v satisfies π in ϵ, by $v \models^\epsilon {}'s$ if v is the symbol s; $v \models^\epsilon \sigma$, if $\sigma(\epsilon) = v$ and $v \neq \bot$; and $v \models^\epsilon \neg\pi$ if $v \not\models^\epsilon \pi$ and $v \neq \bot$. $v \models^\epsilon \pi$ for conjunctive or disjunctive π is defined in the obvious way. Note that $\bot$ cannot satisfy any pattern.

In order to generate the value of an expression e, we need an environment ϵ binding the free variables of e, and a source of randomness, which is provided by an infinite sequence ρ of i.i.d. real numbers $\in [0, 1)$. Formally, we will define a function $\langle e \rangle^{\epsilon,\rho}$, meaning the value generated for e in environment ϵ, given random choices as in ρ. The notation ρ_i^n is the subsequence of ρ of elements with index congruent to i modulo n. This device allows us to split ρ into multiple independent subsequences. Hd[ρ] and Tl[ρ] indicate the head and tail of ρ. For constant, chain, function, conditional and dist expressions, $\langle e \rangle^{\epsilon,\rho}$ is defined by:

$$
\begin{array}{lcl}
\langle {}'s \rangle^{\epsilon,\rho} & = & s \\
\langle \lambda(x_1, \ldots, x_n) \rightarrow e \rangle^{\epsilon,\rho} & = & (x_1, \ldots, x_n) \stackrel{\epsilon}{\rightarrow} e \\
\langle \sigma \rangle^{\epsilon,\rho} & = & \sigma(\epsilon) \\
\left\langle \begin{array}{l} \texttt{if } e_1 \models \pi \\ \texttt{ then } e_2 \\ \texttt{ else } e_3 \end{array} \right\rangle^{\epsilon,\rho} & = & \left\{ \begin{array}{ll} \langle e_2 \rangle^{\epsilon,\rho_2^2} & \text{if } v \models^\epsilon \pi \\ \langle e_3 \rangle^{\epsilon,\rho_2^2} & \text{if } v \models^\epsilon \neg\pi \\ \bot & \text{if } v = \bot \end{array} \right. \\
 & & \text{where } v = \langle e_1 \rangle^{\epsilon,\rho_1^2} \\
\langle \texttt{dist}[p_1{:}e_1, \ldots, p_n{:}e_n] \rangle^{\epsilon,\rho} & = & \langle e_i \rangle^{\epsilon,\mathrm{Tl}[\rho]} \\
 & & \text{where } \sum_{j=1}^{i-1} p_i \leq \mathrm{Hd}[\rho] \leq \sum_{j=1}^{i} p_i
\end{array}
$$

The value of a block expression is complex, and a definition $x_i = e_i$ within the block results in the x_i component mapping to the value of e_i. Each definition is evaluated in an environment including bindings for names appearing previously in the block. Furthermore, if e_i is a lambda expression, the binding for x_i is also added to the environment in which e_i is evaluated, so that x_i is bound in the resulting closure. This allows recursive functions to be defined. Formally:

$$
\begin{array}{l}
\langle \{x_1 = e_1; \ldots; x_n = e_n\} \rangle^{\epsilon,\rho} = \{x_1{:}v_1; \ldots; x_n : v_n\} \\
\quad \text{where} \quad v_i = \langle e_i \rangle^{\epsilon_i,\rho_i^n} \\
\quad\quad \text{and} \quad \epsilon_i = \left\{ \begin{array}{ll} \epsilon[x_1 \ldots x_i / v_1 \ldots v_i] & \text{if } e_i \text{ is a lambda} \\ \epsilon[x_1 \ldots x_{i-1} / v_1 \ldots v_{i-1}] & \text{otherwise} \end{array} \right.
\end{array}
$$

The final case to define is function application. To determine the value of $e_0(e_1, \ldots, e_n)$, we first compute the value of e_0. If it is not a closure, the result of the application is undefined. Otherwise we evaluate the body in the environment formed by extending the closure environment by binding each formal parameter x_i to the value of e_i. Formally:

$$
\langle e_0(e_1, \ldots, e_n) \rangle^{\epsilon,\rho} = \left\{ \begin{array}{l} \langle e' \rangle^{\epsilon'[x_1 \ldots x_n / v_1 \ldots v_n], \rho_{n+2}^{n+2}} \\ \quad \text{if } \langle e_0 \rangle^{\epsilon,\rho_{n+1}^{n+2}} = (x_1, \ldots, x_n) \stackrel{\epsilon'}{\rightarrow} e' \\ \bot \text{ otherwise} \end{array} \right.
$$
$$
\text{where } v_i = \langle e_i \rangle^{\epsilon,\rho_i^{n+2}}
$$

Note that the preceding definitions elegantly take care of the issue of infinite experiments with finite observations, such as the `less_than_half` example above, without needing to make explicit use of laziness in the semantics. The rule for generating the value of an expression only uses the chains that appear in it. Furthermore, a block expression always returns a complex value.

The above semantics is an *operational semantics*, showing how a program consisting of definitions defines a random experiment for generating values. We also provide a *denotational semantics* in terms of a probability measure over values. The underlying probability space consists of countable sequences of i.i.d. real numbers generated uniformly from [0,1). A program $\mathcal{I}$ defines a function from sequences to values, by $\mathcal{I}(\rho) = \langle \{b\} \rangle^{\epsilon_0,\rho}$, where ϵ_0 is the empty environment. If R is a measurable set of sequences, and S is the image of R under $\mathcal{I}$, then S is measurable and $\Pr(S) = \Pr(R)$.

Natural properties of values that we would like to talk about are in fact measurable. For example, any property of the form $\sigma(v) = s$ is measurable. We can see this by defining a *depth-bounded* evaluation function $\langle e \rangle_n^{\epsilon,\rho}$. Its definition is the same as above, except that $\langle \rangle_0$ is always $\bot$, and $\langle \rangle_n$ uses $\langle \rangle_{n-1}$ for evaluating the body of function applications. In other words, values that require a recursion to a depth greater than n will be undefined. Now, a program defines a sequence of functions $\mathcal{I}_i(\rho) = \langle \{b\} \rangle_i^{\epsilon_0,\rho}$. It is clear that the set $R_i = \{\rho : \sigma(\mathcal{I}_i(\rho)) = s\}$ is measurable since it only requires looking at a finite subsequence of ρ to determine if it is in R_i. Therefore the set $R = \{\rho : \sigma(\mathcal{I}(\rho)) = s\} = \cup_i R_i$ is measurable. Furthermore, the above argument also suggests an anytime approximation algorithm for computing $\Pr(S)$. Since the R_i are non-decreasing, and their union is R, the probabilities of the R_i are a non-decreasing sequence whose limit is $\Pr(R) = \Pr(S)$.

3.2 Observations

The language described so far is similar to that of KMP97, albeit with a richer syntax and type system, and a more refined semantics. It can express many common models, such as Bayesian networks, relational probability models, stochastic logic programs, hidden Markov models, dynamic Bayesian networks and stochastic context free grammars. All these models are *generative* in nature, defining an experiment that stochastically generates values for variables. The richness of the model is encoded in the way the values are generated.

Another flavor of probability model is a *rejective* model. In a rejective model, the data is generated by a very simple process, e.g. uniformly, but data that fails to satisfy certain constraints may be rejected. The richness of the model is encoded in the rejection process. A good example of a rejective model is a *product of experts (POE)* [Hinton, 2000]. In a POE, a datum x is generated uniformly, and then passed to

a set of probabilistic experts. Each expert i accepts x with some probability $p_i(x)$ that depends on a property of x. The data is accepted only if all experts accept it. The probability of any datum x is proportional to $\prod_i p_i(x)$.

IBAL is able to express rejective models by making observations an integral part of the language. An *observation* is a declaration of the form $e \models \pi$. Recall that this is the syntax used for Boolean predicates. An observation is simply a statement that a certain predicate is true.

Thus, for example, the general schematic form of a POE model is expressed as follows.

```
generate()      =    ...  (* produce a uniform datum *)
x               =    generate()
(* for each expert i, the following code *)
expert_i(x)     =    ...  (* return 'accept or 'reject *)
expert_i(x)     |=   'accept
```

Another kind of model that can be expressed using observation declarations is a *Markov random field (MRF)*. An MRF is an undirected analogue of a Bayesian network, but it can also be viewed as a type of POE. An interesting effect is at work here. IBAL's expression language defines directed, generative models, but the observation language implements the undirected notion of a constraint. As a result, IBAL is able to express both directed and undirected models, and combinations of the two. It is important to stress that observations are an integral part of the language, and not something pasted onto a model after the fact in order to condition it. They can occur within blocks and functions, and therefore they can be composed together, just like generative definitions. All the power of a modular, functional language is thereby extended to rejective models. Of course, IBAL also allows rejective models to be combined with generative models. For example, one natural way to build a language model is to use a stochastic context free grammar as the initial generator of sentences, and then use probabilistic constraints to express global properties like agreement and sentence length.

In defining the semantics of observations in IBAL, one subtle point must be stressed. An observation in a block can only condition variables defined within the block, not free variables. As far as a containing block is concerned, the definition of a contained block is considered to be a black box. It simply defines a distribution over the value of the block, given values for the free variables. The containing block need not concern itself with whether this distribution is defined generatively or rejectively. Failure to enforce this rule would be a serious violation of modularity.

We get the right effect simply by modifying the definition of $\langle\{b\}\rangle^{\epsilon,\rho}$ for the case of block expressions. Now, in addition to defining values for each of the components of the block, it will also make sure that all the observations in the block are satisfied. If they are not, the generated values for the block are rejected, and the process is repeated. The resulting distribution defined by the block is conditioned on the observations being satisfied. It is, of course, possible to define a set of observations that fail with probability 1, in which case the attempt to generate a value for the block will go on for ever. In that case, the value of the block is $\perp$. Note that because the rejection/repetition process is contained within the block itself, only the value of the block itself is conditioned by the observations, not the free variables.

3.3 Learning

Observations provide the basis for integrating learning, in the form of Bayesian parameter estimation, into the IBAL framework. Unknown probability parameters are specified using *prior* declarations, which have the form `learn` $x =$ `dirichlet` $[\alpha_1 : e_1, \ldots, \alpha_n : e_n]$.

A prior declaration of this form achieves two things. First, it defines a probabilistic parameter $\theta^x = \theta^x_1, \ldots, \theta^x_n$, and specifies a Dirichlet prior over the parameter. The α_i are positive real numbers, specifying the the hyperparameters of the Dirichlet. Second, a prior declaration also creates a definition for x, equivalent to $x =$ `dist` $[\theta^x_1 : e_1, \ldots, \theta^x_n : e_n]$.

We can view an IBAL program with prior declarations as specifying a joint model, that defines a joint probability distribution over the model parameters and the value returned by the program. Observations condition the joint model in the standard Bayesian way. Let us refine the coins example from earlier by adding priors and observations.

```
fair()      =    { result = dist [0.5 : 'h, 0.5 : 't] }
biased()    =    { learn result = dirichlet [90 : 'h, 10 : 't] }
pick()      =    { learn result = dirichlet [1 : fair, 1 : biased] }
coin        =    pick().result
x           =    { y = coin().result ; z = coin().result }
x.y         |=   'h
```

A fair coin is known to produce `'h` with probability 0.5. The probability of `'h` for a biased coin is unknown, but its prior is peaked around 0.9, while the prior over which coin gets picked is uniform. As before, `coin` is the result of picking a coin, and `x.y` and `x.z` are two tosses of `coin`. We also have an observation that `x.y` came out `'h`. This observation has multiple effects. First, because `'h` is more likely for a biased coin, the probability that `coin` is biased is increased, which in turn increases the probability that `x.z` is `'h`. The observation also conditions the probability parameters. Because `coin`, a result of applying `pick`, is likely to have turned out biased, we will get a posterior over the `pick` parameter that is more weighted towards a biased result. Furthermore, because `coin` may have been biased, and because a toss of `coin` came out `'h`, the posterior for the `biased` parameter is also weighed slightly more strongly towards heads.

With its learning component, IBAL is able to do parameter estimation for many common models, including hidden Markov models, stochastic context free grammars and probabilistic relational models. Furthermore, learning in IBAL is not just "added on" to the probabilistic representation language, but is thoroughly integrated into the language. As a result, the benefits of compositionality and modularity are obtained for representing learning tasks. In particular, IBAL is good at representing a cumulative learning framework, in which smaller models are learned and then used as components of larger learning problems. Just as observations only condition values within their scope, they are only used to learn about model parameters within their scope. Thus a compositional learning process can be specified by providing a nested scope containing all the data and parameters for a learning subproblem, and a containing scope that uses the results of the subproblem.

In defining the semantics of learning in IBAL, we need to get a subtle point correct. If a prior declaration appears in the body of a function, the same parameter values should be used every time the function is applied. This is fundamental to learning — different observations of the same function are all observations about the same parameter! This means that in terms of the generative semantics, the parameter value is not returned by each application of the function, but rather it is generated when the function is defined, and stored as part of the closure representing the value of the function object.

To achieve this effect, we make the following definition. The parameters *directly inside* a block are the parameters of prior declarations defined in the block, that are not nested in the body of a lambda expression. In the generative process, the values of the parameters directly inside the body of a lambda are generated at the time the closure is created, and these values are used for all future applications of the closure.

The remainder of the semantics stays basically the same as before. When a definition resulting from a prior declaration is encountered, the relevant parameter values are looked up in the environment and used to choose which branch is taken, in the same way as for a `dist` expression.

3.4 Decisions and Utilities

The representation of decision problems in IBAL is geared towards two popular models: influence diagrams (IDs) and Markov decision problems (MDPs). IBAL can easily represent these and other models, including various kinds of structured MDPs. A decision declaration in IBAL has the form `choose` x `from` $s_1, \ldots, s_n$ `given` $\sigma_1, \ldots, \sigma_m$. This specifies the name x of the decision variable, its range $s_1, \ldots, s_n$, and the information available to the decision maker, the chains $\sigma_1, \ldots, \sigma_m$ (called the *informational parents* of x). A block may contain multiple decision declarations, in which case we enforce the no-forgetting rule of IDs, that the informational parents of later decisions always include those of earlier decisions, as well as the decisions themselves. A reward declaration is either `receive case` e `of` $[\pi_1\text{: } r_1, \ldots, \pi_n\text{: } r_n]$ or `receive` $\alpha\sigma$. The first form states that the reward depends on the value of e, and it is the real number r_i associated with the first pattern π_i that the value satisfies. The second states that the reward is α times the reward of σ, where α is a positive real number. A block may have multiple reward declarations, in which case the total reward is the sum of the individual rewards. For example, a typical MDP is schematically represented as follows:

```
MDP(s) = {   (* takes current state as argument *)
  transition(s,a) = ...   (* returns next state *)
  reward(s,a) = { reward case (s,a) of [('s1,'a1) : 3, ... ] }
  choose a from a1,a2 given s
  next_state = transition(s,a)
  current_reward = reward(s,a)
  future_reward = MDP(next_state)
  reward 1 current_reward
  reward 0.9 future_reward }
```

Three points must be made about the representation of decision problems in IBAL. The first is that each block constitutes a distinct decision problem. A block with decisions is viewed as identifying an implicit agent who makes the decisions and receives the rewards mentioned in the block itself. Decisions and rewards in nested blocks implement the notion of delegation; decisions in the nested block do not consider their effects on the containing block. The reason for enforcing this interpretation is similar to the reason observations don't condition values outside their scope; the alternative would result in a serious loss of modularity. If an agent in a nested block had to be concerned about rewards in the calling block, the decisions for the nested block would no longer be determined solely by its free variables. Therefore a program calling the nested block could no longer treat it as a black box.

The second point to clarify is that it is assumed that the values of free variables are always known to an agent making the decisions in a block. Once again, failure to enforce this restriction would result in a modularity problem. If the agent does not know the values of the free variables, it must know the distribution over those values in order to make a rational decision. But then, the distribution defined by a block depends not only on the values of the free variables, but on their distribution. This is an unfortunate restriction, since it prevents IBAL from being capable of representing POMDPs. The situation can perhaps be salvaged, by borrowing the idea of belief state from POMDPs, and adding it as an extra implicit input to the block.

The third point is the relationship between observations and decisions in a block. We assume that all observations are known to the decision maker, even if they appear lexically subsequent to the decision. The reason is that the observations are viewed simply as part of the definition of the probability distribution over the block's values, and we make the assumption that the decision maker knows the correct distribution. A result of this assumption is that we disallow observations statement to mention variables that depend (directly or indirectly) on the decisions in a block. This restriction prevents the semantics of learning and decisions from interfering with each other.

In defining the semantics, we begin with utilities. With every value v we associate a utility $U(v)$. For non-complex values $U(v) = 0$. For complex values $U(v)$ is determined by the reward declarations in the block in which v was created. Recall that a complex value is created by $\langle\{b\}\rangle^{\epsilon,\rho}$. We extend the definition of $\langle\{b\}\rangle^{\epsilon,\rho}$ so that it also produces $U(v)$ after producing v. We set $U(v)$ to be the sum over reward declarations R in b of $U_R(v)$, defined as follows. If R is `reward case` σ `of` $\pi_1 : r_1, \ldots, \pi_n : r_n$,

$$U_R(v) = \begin{cases} r_i & \text{if } \sigma(v) \models^{\epsilon'} \pi_i \text{ and } \sigma(v) \not\models^{\epsilon'} \pi_j \text{ for } j < i \\ 0 & \text{if } \sigma(v) \not\models^{\epsilon'} \pi_i \text{ for } i = 1, \ldots, n \end{cases}$$

where ϵ' is the environment resulting from extending ϵ with the bindings in v. If R is `reward` $\alpha\sigma$, $U_R(v) = \alpha \cdot U(\sigma(v))$.

Now for decisions. A *strategy* d for block b is a decision $d_{D,w}$ for each decision statement D and each value w of the informational parents of D. Given an environment ϵ in which to evaluate b, and a particular strategy d, we define a function $U(b)_d^{\epsilon,\rho}$, whose meaning is the utility for b, given strategy d, bindings of free variables in ϵ, and random choices specified by ρ. Holding ϵ, b and d fixed, this function is a random variable on the space of sequences. Taking the expectation of this function over sequences produces the expected utility; the optimal strategy for b is then the one that maximizes this

expected utility. The definition of $\langle\{b\}\rangle^{\epsilon,\rho}$ now proceeds by first choosing $d^* = \arg\max_d E_\rho[U(b)^{\epsilon,\rho}_d]$,and then processing the block as before, using d^* to choose the values of each decision variable given its informational parents.

3.5 Pragmatics

The IBAL language allows a wide variety of different kinds of models to be expressed. There is no single best inference algorithm to use for all models. While IBAL implements a good default algorithm that should work well in many situations, it also allows the possibility of using other methods. The programmer can specify control knowledge on how to solve a program in the form of *pragmatic declarations*. These could either specify what algorithm to use, or details of how to use a particular algorithm, such as specific elimination orderings for variable elimination.

Because pragmatic declarations are part of a block, they only specify how to do inference in that block and its nested blocks. Furthermore, pragmatic declarations in a nested block can override those in a containing block. These features allow fine, modular, control over how inference is done, allowing complex models to be built in which different inference methods are used for different components.

It is anticipated that pragmatic declarations will often be used by library designers. The user of a library is shielded from thinking about how the library models will be solved. This modularity of inference methods may turn out to be as important as modularity of representation in building complex probabilistic agents.

4 Example

In this section, I illustrate how a declarative high-level representation language that unifies probabilistic reasoning, decision theory and parameter estimation can simplify and integrate the implementation of sophisticated rational agents. Consider the task of implementing an automated receptionist agent. The job of the agent is to receive spoken requests over the phone and to respond to them appropriately. Requests include asking for directions and talking to a particular person; responses include giving directions, connecting to an extension, and asking the caller to repeat the request.

In a decision-theoretic design for this agent, the agent receives a utility based on how well its response matches the actual request of the user. Of course, the agent does not observe the actual user request, but only a sequence of signals. In the decision model, there will be a prior distribution over requests and a conditional distribution of signals given requests.

With existing tools, it is difficult to implement this decision-theoretic design using a single, coherent model. Instead, a typical implementation might consist of several components, each of which is probabilistic, but which are stitched together in an ad-hoc manner. There will typically be a speech-recognition component that determines the likely words that generated the received signal; a language model that can determine the probability that a particular request generated a particular sentence; and a high-level influence diagram for deciding what to do. The results of the speech-recognizer and request generator are fed into the language model. The results of the language model are fed into the influence diagram. The overall result may not be a coherent probabilistic model. Furthermore, it may be hard to tailor the components for their use in this application. For example, getting the speech recognizer to recognize unusual names of individuals in the company might require an extra engineering effort.

With IBAL, the whole application can be described using a single declarative model. At a high-level, the generative model consists of three steps: a request generation function, a function that generates sentences based on requests, and a function that generates phoneme sequences from sentences. For the decision-making component, the agent is given the phoneme sequence, and chooses a response. The agent's utility is determined from the request and the response. The high-level code looks like this:

```
request = make_request()
sentence = make_sentence(request)
phonemes = make_phonemes(sentence)
choose response.type from GiveDirections, Connect, Repeat
choose response.who from fred, wilma
receive match(request, response)
```

I will now describe, in turn, the three generative steps and the utility computation. The function `make_request` produces a request. A request has a `type` field, whose value is either `GetDirections` or `Talk`. A `Talk` request also has a `who` field, whose value is a person. A person is a complex value with three fields: `name`, `title` and `extension`. In a typical programming language, the type of `name` would be string, because what we typically want to do with names is compare them, read them and print them. In this application, however, the most important thing about a name is how it tends to be pronounced. Therefore the type of `name` is a function that stochastically generates a phoneme sequence. The same is true for `title`. The `make_request` function is as follows:

```
make_request() = {
  learn who = [1 : fred, 1 : wilma]
  learn type = [20 : GetDirections; 80 : Talk]
}
```

Generating the request involves choosing two things: the type of the request, and the person, if the request is to talk to a person. Both choices are made learnable. The agent has lots of past experience about what kinds of requests tend to be generated. These are expressed by observation statements as follows:

```
r1 = make_request()
r1.type |= GetDirections
r2 = make_request()
r2.type |= Talk
r2.who |= wilma
...
```

The `make_sentence` function produces a sentence from a request. A sentence is a list of words, each of which is a function that stochastically produces a list of phonemes when activated. Here I illustrate with an extremely simple sentence generation model; in a real application, a more complex grammar model might be used. Even this simple example illustrates the power of passing around data structures that contain functions in fields. The sentence generator simply slots the given person's name or title in the appropriate place. ($[x_1; \ldots; x_n]$ is the list containing $x_1, \ldots, x_n$, and @ is the list concatenation operator.)

```
make_sentence(request) =
  if request.type ⊨ GetDirections
    then [can;you;give;me;directions]
    else [connect;me;to] @
      [dist [0.5:request.who.name, 0.5:request.who.title]]
```

The `make_phonemes` function takes a sentence and produces a list of phonemes. Since each word in the sentence is a function that generates phoneme sequences, `make_phonemes` just executes each of the words in the sentence, and concatenates the results. The individual word models would probably be hidden Markov Models, and may be part of a speech recognition library. Unusual names of people may not be in the generic hMM library, but the library may provide a function that creates a new, trainable hMM that can be learned for a particular name.

```
make_phonemes(sentence) =
  if sentence.is_empty
    then []
    else head(sentence)() @ make_phonemes(tail(sentence))
```

Finally, the utility model takes the request and response, and produces a utility of 1 if the request matches the response. If the response is RepeatPlease, the utility is 0. An incorrect response has utility -5.

```
match(request,response) = {
  correct =
    (request.type ⊨ GetDirections ∧
      response.type ⊨ GiveDirections) ∨
    (request.type ⊨ Talk ∧
      response.type ⊨ Connect ∧
      request.who ⊨ response.who)
  receive
    if correct
      then 1
      else if response.type ⊨ Repeat
        then 0
        else -5
}
```

5 Implementation Overview

The inference tasks in IBAL are to estimate the learnable parameters, to compute the utility maximizing decisions, and to solve for the conditional distribution over various chains, given values for other chains. The tasks are accomplished in that order. Since decisions cannot influence observations, they are irrelevant to the learning task. Once the parameters have been learned, the expected utility for any strategy can be computed and the decision task can be solved. Once the decisions have been fixed, the distribution over all values is known and the probabilistic reasoning task can be solved.

IBAL is implemented in Objective CAML, a variant of ML. The design of the implementation is divided into four parts. The first component is a *frontend* consisting of a parser, type checker, and translator, whose job is to produce code in *shallow form*. In shallow form, nested subexpressions are replaced by chains, and only simple patterns are allowed. The second component is a set of *support modules*. These include a ubiquitous polymorphic container type implementing efficient maps from chains to values of any kind, as well as a general implementation of *events*, which are measurable sets of values, and *factors*, which are measurable functions from values to elements of a ring. Events and factors support a very generalized version of the variable elimination algorithm.

The third component implements the *main line* of inference. The main line begins with code in shallow form, and proceeds through a sequence of steps. The first step is *domain generation* in which the support $D(\sigma)$ of each chain used in the program is computed. Domain generation uses rules such as the following: for a definition $w = \texttt{dist}\ [p_1 : \sigma_1, \ldots, p_n : \sigma_n]$, $D(w)$ is the union of the $D(\sigma_i)$. Domain generation also uses observations to restrict the domains. Domain generation may be followed by an optional *constraint propagation* step, which further restricts the domains, by propagating observations back through the definitions in the program. Constraint propagation may be worthwhile since it is cheaper than the full-scale variable elimination process performed later.

The second step is to compute the *needed chains* that are actually relevant to answering a query. These include all chains that influence query variables or observed variables, directly or indirectly. This computation step is particularly important in recursive models such as stochastic grammars. It deduces, for example, that in order to determine whether the first word of a sentence is "the", only the first non-terminal need be expanded at any stage (assuming there are no ϵ productions).

The key step in the main line, *factor production*, converts each definition in the program into a set of factors. The goal is to produce factors that are as simple as possible; dummy variables may be introduced to achieve this. For example, for a definition $x = y$, suppose x and y are complex, with $x.a$ and $x.b$ needed. Two separate factors will be produced, one enforcing that $x.a$ and $y.a$ are equal, and the other for $x.b$ and $y.b$. For a definition $x = \texttt{dist}\ [p_1 : \sigma_1, \ldots, p_n : \sigma_n]$, a dummy variable d is introduced, to represent which branch is actually taken. The dummy variable serves to separate the σ_i from each other; without the dummy variable, they would all have to appear in the same factor. A set of factors is produced for each branch i, saying that if d takes the value i, then the components of x and σ_i must have the same value, but if d takes any other value this branch is irrelevant and there is no constraint on x and σ_i. An additional factor saying that d takes value 1 with probability p_1, ..., n with probability p_n is also produced.

The final solution step is *variable elimination*, as is commonly used for BN inference. The set of factors computed by factor production represents a sum of products expression. All variables except for query variables and free variables in a block are eliminated from the factors, and the result is the conditional probability of the query variables given the free variables. All solution steps except for the final variable elimination are recursive; a recursive call is used to process nested blocks and function applications.

The final component of the implementation is the *glue* that holds everything together. Each of the solution steps is turned into a dynamic programming algorithm, where all recursive calls are looked up in a cache before solving them explicitly. Also, an iterative deepening strategy is used to provide an anytime approximation algorithm for queries that may not terminate. These recursion strategies are implemented in a

generic, modular way. For example, a `dynamic` function is provided that takes a recursive algorithm and produces a dynamic programming version of the same algorithm. The glue is also responsible for handling pragmatics.

The glue is implemented using a programming technique in which a recursive function f takes a special argument g. When f recurses, rather than calling itself directly, it calls g. Thus f is a function with a "hole", which needs to be filled in. All the recursion strategies can be implemented as ways of filling in the hole. The simplest way to fill in the hole is to plug f into itself, which gives the basic recursion pattern. But another possibility is to define a function `cached`(f) which looks for a result in a cache before calling f, calls f if the result is not found, and stores the result of calling f in the cache. If `cached`(f) is plugged into itself, the result is a dynamic programming algorithm based on f. Similarly, `depth-bounded`(n, d, f) produces a version of f that stops recursing after a recursion depth of n, returning d instead. Recursion strategies can be composed using the method of functions with holes. This provides a nice, modular way to deal with multiple inference methods. Each inference method is invoked by a pragmatics handler that recognizes a particular type of pragmatic declaration. The pragmatic handlers can be installed on top of each other, so that every recursive call results in the all the handlers being checked. The inference methods themselves do not need to know what handlers are used for the recursive calls.

All three inference problems use dynamic programming with the main line of inference described above. Decision making is based on ID algorithms, using backward induction to solve the decisions in a block from the bottom up. For MDPs, the algorithm reduces to value iteration (because of the dynamic programming component) with reachability analysis (because of the domain generation). For the probabilistic reasoning task, all reasoning is directed towards a particular query; pruning of the problem to those variables required for the query is accomplished by computing the needed chains. For probabilistic reasoning tasks, the algorithm reduces to regular variable elimination for standard BNs, while the dynamic programming makes it equivalent to forward-backward for HMMs and the inside algorithm for SCFGs. Finally, parameter estimation uses a general version of the EM algorithm to compute maximum a-posteriori parameter values. This is only an approximation to the true Bayesian posterior, of course. The E step consists of using the main inference line to compute how many times each branch was taken for each prior declaration. These are the expected sufficient statistics. The M step combines the expected sufficient statistics with the Dirichlet hyperparameters to produce new parameter values that have maximum posterior probability given the expected sufficient statistics. Again, the combination of variable elimination with dynamic programming results in the algorithm reducing to the typical EM used for BNs, Baum-Welch for HMMs, and the outside algorithm for SCFGs.

I do not claim that IBAL will be an efficient inference algorithm for all applications. Some problems are inherently hard, or require special methods, and a variety of algorithms is needed. I do believe that IBAL integrates a number of widely used methods, and thereby provides a good default algorithm. Furthermore, the support modules and glue make it easier to extend the system with alternative methods.

6 Conclusion and Future Work

IBAL is a rich declarative programming language for describing probabilistic models, decision theoretic situations, and Bayesian parameter estimation problems. This paper has presented a syntax and semantics for the language, and an overview of an implementation with probabilistic reasoning, utility maximization and parameter learning.

Future work on IBAL is divided into implementation enhancements that are practically useful but not too theoretically challenging, and more significant extensions to the expressive power of the language. Plans in the first category include implementing a variety of approximate solution methods; providing libraries to implement common kinds of models, and useful structures such as sets of objects; and extending the type system to include algebraic data types and polymorphic types, modeled on ML.

One of the main future tasks in the second category, is to support the learning of model structure as well as parameters. In keeping with the philosophy that everything that can be done in IBAL should be thoroughly integrated, this will require a much richer language with which to express priors, including priors over structure. Another important extension is to provide a way to describe how an IBAL agent interacts with its environment; in other words, to provide a declarative description of the I/O capaboilities of the agent. A final task is to allow multiple agents into the IBAL framework. Not only would this allow IBAL to be used for modeling game-theoretic situations, it would also provide a way to describe agents' models of other agents. While these tasks present a range of interesting and challenging issues, IBAL provides a good foundation with which to begin tackling them.

References

[Boutilier *et al.*, 2000] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *AAAI*, 2000.

[Cumby and Roth, 2000] C. Cumby and D. Roth. Relational representations that facilitate learning. In *KR*, 2000.

[Hinton, 2000] G. Hinton. Training products of experts by minimizing contrastive divergence. Technical report, Gatsby Computational Neuroscience Unit, 2000.

[Koller *et al.*, 1997] D. Koller, D. McAllester, and A. Pfeffer. Effective Bayesian inference for stochastic programs. In *AAAI*, 1997.

[McAllester, 2000] D. McAllester. Bellman equations for stochastic programs. Revision of talk at LPNMR-99, 2000.

[Muggleton, 2000] S. Muggleton. Stochastic logic programs. *Journal of Logic Programming*, 2000. Accepted subject to revision.

[Pfeffer *et al.*, 1999] A. Pfeffer, D. Koller, B. Milch, and K.T. Takusagawa. SPOOK: A system for probabilistic object-oriented knowledge representation. In *UAI*, 1999.

Approximate inference for first-order probabilistic languages

Hanna Pasula and Stuart Russell
Computer Science Division, University of California
387 Soda Hall, Berkeley, CA 94720-1776
{pasula,russell}@cs.berkeley.edu

Abstract

A new, general approach is described for approximate inference in first-order probabilistic languages, using Markov chain Monte Carlo (MCMC) techniques in the space of concrete possible worlds underlying any given knowledge base. The simplicity of the approach and its lazy construction of possible worlds make it possible to consider quite expressive languages. In particular, we consider two extensions to the basic relational probability models (RPMs) defined by Koller and Pfeffer, both of which have caused difficulties for exact algorithms. The first extension deals with uncertainty about relations among objects, where MCMC samples over relational structures. The second extension deals with uncertainty about the identity of individuals, where MCMC samples over sets of equivalence classes of objects. In both cases, we identify types of probability distributions that allow local decomposition of inference while encoding possible domains in a plausible way. We apply our algorithms to simple examples and show that the MCMC approach scales well.

1 Introduction

Recent work in AI has made clear the advantages to be derived from combining probability theory with (at least some of) the expressive power of first-order logic [Wellman *et al.*, 1992]. We will call languages that exhibit such a combination *first-order probabilistic languages* (FOPLs). The ability to handle objects, relations, and quantification gives such languages a huge advantage in representational efficiency in certain situations. To take a purely logical example: the rules of chess can be written in about one page of Prolog but require perhaps millions of pages in a propositional language. A recent thesis on first-order probabilistic languages [Pfeffer, 2000] describes a battlespace management system involving potentially thousands of objects whose relationships are unknown and changing. [Pasula *et al.*, 1999] describe a freeway traffic surveillance involving probabilistic inference about the identities and properties of thousands of vehicles. These applications would be infeasible without some ability to specify and reason with FOPL knowledge bases.

The semantics of FOPLs are based on the idea that each model of a FOPL knowledge base should be viewed as a probability measure over the possible worlds (logical models) defined by the constant, function, and predicate symbols of the knowledge base [Halpern, 1990]. Although "wildly undecidable" in full generality, highly restricted FOPLs appear to be practical, especially with finite models. Two threads have arisen, based on semantic networks (e.g., [Koller and Pfeffer, 1998]) and logic programming (e.g., [Sato and Kameya, 1997]).

In this paper, we focus on the family of *relational probability models* (RPMs) [Pfeffer, 2000], although our ideas apply equally to other languages. RPMs, like semantic networks, are based on *classes* containing *instances*, with each instance possessing *attributes*. (See Section 2 for details.) RPMs allow one to specify probability distributions over the attribute values of an instance, either directly or via inheritance from classes. These distributions may depend on other attribute values of the instance or of other instances. For example, a PhD student's success may depend on the fame of his or her advisor.

[Pfeffer *et al.*, 1999] describe an exact inference algorithm for RPM knowledge bases called *structured variable elimination* (SVE). Roughly speaking, SVE applies the variable elimination algorithm [Zhang and Poole, 1996] to a dynamically constructed Bayesian network whose nodes are all those ground propositional variables defined by the knowledge base and relevant to the current query. SVE derives an efficient variable ordering from the structure of the knowledge base and reuses computation results where possible. It is often able to answer queries involving hundreds of variables in a few seconds.

Despite SVE's excellent performance, its runtime is at least exponential in the size of the largest clique in the optimal triangulation of the network. The expressive power of RPMs makes it very easy to construct knowledge bases whose corresponding Bayesian networks have very large cliques. For example, [Pfeffer, 2000] describes a model for matches in a sports league. The knowledge base consists of a single, generic conditional distribution for the outcome of a match given the quality of the two teams, a single prior distribution for the quality of the teams, and the results of some matches. If every team plays every other, then the team qualities form a clique in the corresponding Bayesian network and the infer-

ence cost is exponential in the number of teams. Similar problems will arise in any application in which there are complex relationships among large numbers of objects—i.e., precisely those domains for which FOPLs are really necessary.

The situation is exacerbated when the RPM language is extended to allow for *structural uncertainty*—i.e., uncertainty about which probabilistic dependencies actually exist. We consider two forms of structural uncertainty:

reference uncertainty: the value of a relational attribute may be uncertain—e.g., we may not know which of two professors is the advisor of a certain student;

identity uncertainty: we may not know whether two objects in the knowledge base are the same—e.g., when we see a red bus at two different camera locations on the freeway.

Each of these extensions may lead to Bayesian networks whose size alone causes difficulties and whose high connectivity makes exact inference completely impractical. For example, the inference problem in the freeway surveillance application of [Pasula *et al.*, 1999] is known to be #P-hard, i.e., almost certainly exponential in the number of vehicles.

The solution proposed by [Pasula *et al.*, 1999] is to use a Markov chain Monte Carlo (MCMC) algorithm (see Section 3 for details). The algorithm samples from possible matchings among vehicles, converging (in some cases polynomially) to approximately correct probabilities. Often a few hundred samples suffice for a state space of 2^{1000} states. The states being sampled are essentially the possible worlds defined by the constant symbols (observed vehicles) and predicates (equality) of the knowledge base. This approach—sampling possible worlds with MCMC—can be turned into a general inference algorithm for first-order probabilistic languages, as suggested by [Russell, 1999; Pfeffer, 2000].

In this paper, we investigate MCMC on possible worlds as an inference method to handle reference uncertainty (Section 4) and identity uncertainty (Section 5). We show how the possible worlds may be constructed dynamically and how the transition probabilities may be computed efficiently. For the case of transitions involving a referentially uncertain relational attribute value, we identify a large family of conditional distributions that render the calculation independent of all but the particular values involved in the transition. We illustrate the algorithms using simple examples and give experimental results suggesting that the algorithms scale well.

2 Relational probability models

The following definitions are adapted from [Koller and Pfeffer, 2000]. A relational probability model, in its most basic form, consists of

- A set $\mathcal{C}$ of *classes* denoting sets of objects, related by subclass/superclass relations.
- A set $\mathcal{I}$ of *named instances* denoting objects, each an instance of one class.
- A set $\mathcal{A}$ of *complex attributes* denoting functional relations. Each complex attribute A has a domain type $Dom[A] \in \mathcal{C}$ and a range type $Range[A] \in \mathcal{C}$.

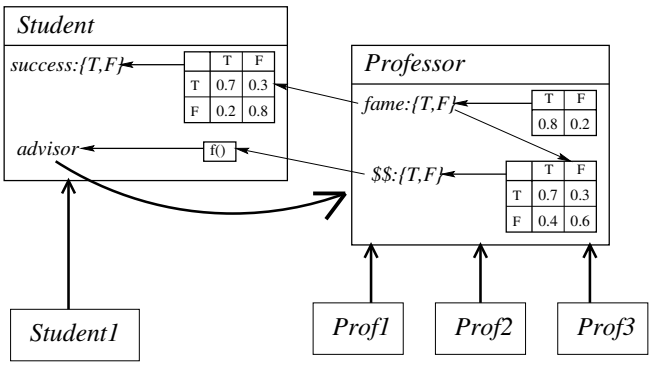


Figure 1: The simple RPM defined in the text, with its associated conditional probability models.

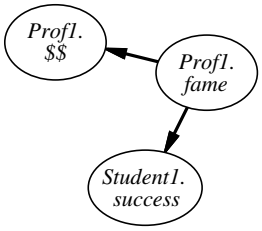


Figure 2: Bayesian network structure generated from the RPM in Figure 1. The *advisor* relationship, being certain, doesn't appear.

- A set $\mathcal{B}$ of *simple attributes* denoting functions. Each simple attribute B has a domain type $Dom[B] \in \mathcal{C}$ and a range that is a finite, enumerated set of values $Val[B]$.
- A set of conditional probability models $P(B|Pa[B])$ for the simple attributes. $Pa[B]$ is the set of B's *parents*, each of which is a nonempty chain of (appropriately typed) attributes $\sigma = A_1.\cdots.A_n.B'$, where B' is a simple attribute. Probability models may be attached to instances or inherited from classes.

For now, we will assume that the values of all the complex attributes are known (no reference uncertainty), and that every instance is distinct (unique names assumption, hence no identity uncertainty). Thus, a possible world is defined by the values of the *instance variables*—the simple attributes for all named instances.

Consider the following very simple RPM. There are two classes, *Student* and *Professor*, and two instances, $Student_1$ and $Prof_1$. There is one complex attribute, *advisor* (mapping *Student* to *Professor*) and the *advisor* of $Student_1$ is $Prof_1$. There are three simple attributes, all Boolean: *success* of a $Student$ and *fame* and *$$* (funding level) of a $Professor$. For any *Student* s, s.*success* has one parent, s.*advisor.fame*, with an appropriate conditional distribution. For any *Professor* p, p.*fame* has no parents and a simple prior distribution, while p.*$$* depends on p.*fame*. Figure 1 shows the knowledge base with probability distributions and Figure 2 shows the Bayesian network structure for the attribute variables definable from the knowledge base.

3 Markov chain Monte Carlo algorithms

MCMC [Gilks *et al.*, 1996] generates samples from a posterior distribution $\pi(x)$ over possible worlds x by defining a Markov chain whose states are the worlds x and whose *stationary distribution* is $\pi(x)$. In the *Metropolis–Hastings* method (henceforth M-H), transitions in the Markov chain are constructed in two steps:

- Given the current state x, a candidate next state is generated from the *proposal distribution* $q(x'|x)$, which may be (more or less) arbitrary.
- The transition to x' is not automatic, but occurs with an *acceptance probability* defined by

$$\alpha(x'|x) = min\left(1, \frac{\pi(x')q(x|x')}{\pi(x)q(x'|x)}\right)$$

It is not necessary that all the variables of state x be updated simultaneously, in a single transition function. *Single-component* M-H alters each variable in turn. It is also possible to factor q into separate transition functions for various subsets of variables. Provided that q is defined in such a way that the chain is ergodic, this transition mechanism defines a Markov chain whose stationary distribution is $\pi(x)$.

The Gibbs sampling algorithm for Bayesian networks [Pearl, 1988] is a special case of Metropolis–Hastings in which the proposal distribution samples a single variable X_i using the distribution $P(X_i|\mathbf{mb}(X_i))$, where $\mathbf{mb}(X_i)$ denotes the current values of the variables in the Markov blanket of X_i (its parents $Pa[X_i]$, children Y_j, and children's other parents). In this case, the acceptance probability is always 1. One can show easily that

$$P(X_i|\mathbf{mb}(X_i)) = \alpha P(X_i|Pa[X_i])) \prod_j P(Y_j|Pa[Y_j]) \quad (1)$$

Gibbs sampling is very simple and also *local*: transitions are generated referring only to parts of the model directly connected to the variable in question. Hence, *the cost per transition is typically independent of model size*. M-H sampling is also typically local because all the parts of the model that are not changed by the transition cancel in the ratio $\pi(x')/\pi(x)$. In particular, if the proposal concerns a single variable X_i, this ratio reduces to $P(x'_i|\mathbf{mb}(X_i))/P(x_i|\mathbf{mb}(X_i))$, where x'_i is the proposed value of X_i and x_i is its current value. The M-H algorithm, unlike Gibbs, has the added advantage that the transition may often be computed without referring to the other values of X_i at all, as we will see.

4 Reference uncertainty

Reference uncertainty arises whenever relations among objects, as described by complex attribute values, are not known with certainty. For example, we may be unsure as to which of three professors is *Student*$_1$'s advisor. We need to be able to describe this uncertainty and to specify the dependencies that influence it. The following definitions are adapted from [Pfeffer, 2000]:

- With each complex attribute A, we associate a simple *reference attribute* $ref[A]$, such that $Val[ref[A]]$ is a finite, enumerated set of named instances. (Where no confusion arises, we may drop the $ref[\cdot]$ and use the attribute name itself.) Dependencies are expressed as before by a conditional distribution $P(ref[A]|Pa[ref[A]])$. The parents of $ref[A]$ are those attributes or attribute chains that influence the choice of an instance as the value of attribute A.
- Reference uncertainty modifies the definition of attribute chains. Suppose that an attribute B depends on the parent chain $\sigma = A_1.\cdots.A_n.B'$. Any of the complex attributes in the chain may be uncertain. Then the parent variables for B are all the instance variables reached by the chain for all possible combinations of values of all the uncertain complex attributes, as well as all the reference variables for those attributes.

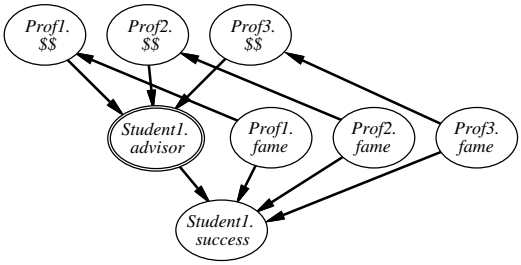


Figure 3: Bayesian network structure with reference uncertainty about *Student*$_1$*.advisor*. The reference attribute is shown in a double oval.

The simple example of Figure 1 can be extended to include reference uncertainty, if *Student*$_1$*.advisor* is unknown. We define a reference attribute ref[*Student*$_1$*.advisor*] with range (say) {*Prof*$_1$, *Prof*$_2$, *Prof*$_3$}. The choice of advisor depends (generically) on the funding (*Prof.$$*) of each candidate, which depends (generically) on *Prof.fame*. This gives the instance-variable network structure shown in Figure 3. Obviously, when reference attributes have many possible values, very large implicit network structures can result.

4.1 Exact inference with reference uncertainty

As mentioned above, the runtime of any variable elimination algorithm is exponential in the size of the largest clique in the optimally triangulated graph. Looking at Figure 3, it is apparent that a straightforward application leads to two "large" cliques: one containing *Student*$_1$*.success* and its parents, and one containing ref[*Student*$_1$*.advisor*] and its parents. In general, these will have $n + 2$ and $n + 1$ members respectively, where n is the number of possible values for the reference attribute. Thus, inference cost grows exponentially with this number.

[Pfeffer, 2000] observes that at least one of these cliques—the one associated with *Student*$_1$*.success*—can be decomposed. *Given a known value for* ref[*Student*$_1$*.advisor*], *Student*$_1$*.success* does *not* depend on the fame of other professors (Figure 4). This is an extreme form of context-specific independence, and allows the $n + 2$-variable factor to be replaced by a product of n 3-variable factors in the variable elimination process. This decomposition applies in general

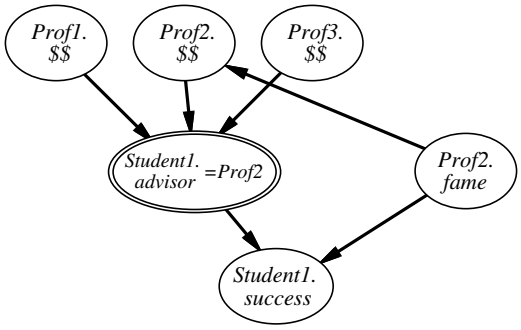


Figure 4: The network structure conditioned on $Student_1.advisor = Prof_2$.

to variables that have referentially uncertain parents.

Unfortunately, the clique associated with the reference attribute itself—$ref[Student_1.advisor]$ in this case—cannot be decomposed. We are left with a network structure that seems (in general) to be intractable for exact inference. The situation becomes worse still as more complex interactions occur among several reference attributes.

4.2 MCMC inference with reference uncertainty

In our approach, we extend the MCMC algorithm commonly used on Bayesian networks by augmenting the Markov chain state space to include reference variables denoting relational uncertainty, and defining appropriate transition functions. Simple attributes, such as $Student_1.success$, use Gibbs sampling as usual—and context-specific independence can sometimes be used to simplify those steps further. Reference attributes, such as $ref[Student_1.advisor]$, depend on a slightly more involved M-H step.

The Gibbs steps

In what follows, we abbreviate $Student_1$ to S_1, *advisor* to a, and so on. Let us consider an extended version of Figure 3 with n potential advisors. Let the current value of $S_1.a$ be P_i. Suppose we want to apply Gibbs sampling to $P_j.f$ ($Prof_j.fame$), where $j \neq i$. By Equation (1), the sampling distribution is

$$\begin{aligned}
&\alpha P(P_j.f)P(P_j.\$\$|P_j.f)P(S_1.s|P_1.f,\ldots,P_n.f,S_1.a=P_i)\\
&\quad = \alpha P(P_j.f)P(P_j.\$\$|P_j.f)P(S_1.s|P_i.f,S_1.a=P_i)\\
&\quad = \alpha' P(P_j.f)P(P_j.\$\$|P_j.f)
\end{aligned}$$

The first step of this derivation uses the fact that $S_1.s$ depends only on the fame of S_1's advisor, $P_i.f$, and not on other professors; the second step simply observes that $P(S_1.s|P_i.f,S_1.a=P_i)$ is constant w.r.t. $P_j.f$. Thus, when the reference attribute is instantiated, sampling operates exactly as if the links from non-selected parents were nonexistent. The same holds when sampling a child of the reference variable (e.g., $S_1.s$). Thus, while reference variables remain constant, the network has a simplified form, such as that shown in Figure 4. MCMC sampling on this simpler network should converge quickly. When the value of the reference variables changes, however, so does the effective structure of the network. We now address this issue.

The Metropolis–Hastings steps

Gibbs sampling for a reference variable with n values involves considering n possible network structures, so we apply M-H sampling instead. M-H proposes a single new value for the reference variable, and then decides whether to accept it. (We can ignore the proposal distribution, which we will assume for now is uniform and hence cancels.) For the transition from $S_1.a=P_i$ to $S_1.a=P_j$, then, we need (from Equation (1), and simplifying based on known values of the reference variable) the ratio

$$\frac{P(S_1.a=P_j|P_1.\$\$,\ldots,P_n.\$\$)P(S_1.s|P_j.f,S_1.a=P_j)}{P(S_1.a=P_i|P_1.\$\$,\ldots,P_n.\$\$)P(S_1.s|P_i.f,S_1.a=P_i)}$$

At first sight, it seems that calculating this ratio requires accessing the current values of $P_1.\$\$,\ldots,P_n.\$\$$, i.e., the funding levels of all possible candidate professors, even though the transition involves just two of them. (This is because the probability of picking any one advisor *does* depend on the funding levels of all candidates.) In turn, this requires that all those nodes be constructed and instantiated, which we prefer to avoid if possible.

It so happens, fortunately, that conditional distributions for "selecting" a value for a relational attribute, given properties of a set of candidates, may have some structural properties that simplify the task. Suppose, for example, that for all i,

$$P(S_1.a=P_i|P_1.\$\$,\ldots,P_n.\$\$)=\frac{f(P_i.\$\$)}{\sum_k f(P_k.\$\$)}$$

for some arbitrary function f. Then, in the transition probability ratio given above, the summations $\sum_k f(P_k.\$\$)$ cancel, leaving

$$\frac{f(P_j.\$\$)P(S_1.s|P_j.f,S_1.a=P_j)}{f(P_i.\$\$)P(S_1.s|P_i.f,S_1.a=P_i)}$$

which does not mention any values for the reference variable besides P_i and P_j. The property of conditional distributions that we require is satisfied by some well-known conditional distributions, including the *softmax* distribution:

$$P(X=i|Y_1=y_1,\ldots,Y_n=y_n)=\frac{e^{y_i}}{\sum_k e^{y_k}}$$

Softmax is indeed a reasonable model for selecting an advisor based on funding.

The complete algorithm

Once we put the two types of steps together, the complete algorithm alternates between ordinary Gibbs steps on a simplified network and M-H steps altering the network structure. It should be noted that different network structures may result in different sets of variables being, at any given time, relevant or irrelevant to the query. Our approach restricts computation to variables that are strictly relevant [Zhang and Poole, 1996] according to the current structure.

A further possible enhancement is *lazy construction* of the Bayesian network. The network can, e.g, be grown "from the query outwards", with nodes added as they are needed to sample a node currently in the network. (Structural variables are, like others, instantiated on creation, yielding a network

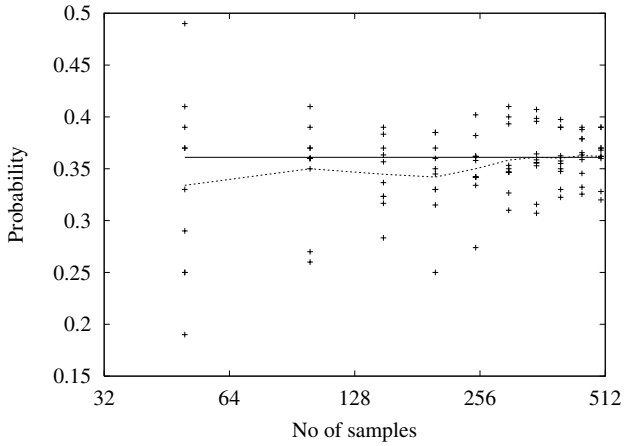


Figure 5: MCMC convergence on a simple example with three professors and one student. The exact probability is 0.361 (horizontal line). Each point represents the estimate from one of 10 Markov chains after a given number of samples (log scale). The dotted line shows the average of the 10 individual estimates.

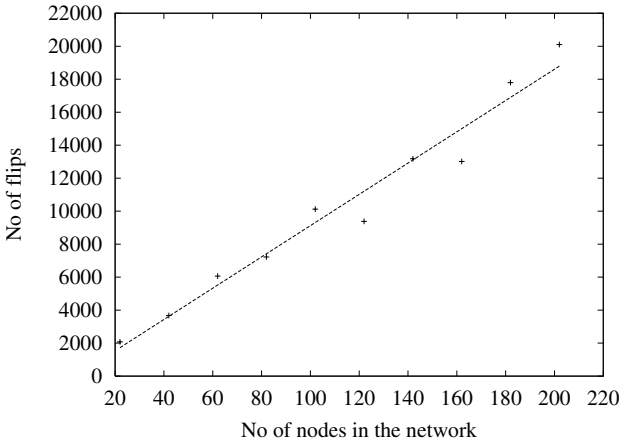


Figure 6: MCMC convergence as a function of network size. The y-axis measures the number of state transitions required to reach a preset convergence threshold for a standard diagnostic. The straight line is a regression fit to the data, which is averaged over 20 trials.

in the usual simplified format.) If necessary, nodes may then be discarded when they become irrelevant to the query under the current network structure. By using an "intelligent" M-H proposal, as discussed in Section 6, we can also focus computation on variables that are "important" to the query—perhaps visiting only an infinitesimal fraction of the potentially relevant variables—without sacrificing the guarantee of convergence to correct probabilities in the limit. This should permit us to apply our approach to very large, or even infinite, knowledge bases.

4.3 Experimental results

To check our algorithm, we applied it first to the knowledge base with reference uncertainty, no identity uncertainty, and three professors. In this case, the algorithm generates and evaluates the network shown in Figure 3. We set values for fame or funding for each professor, and queried the posterior probability of $Student_1.success$. In this tiny model, we can compute the exact value for comparison. The MCMC estimates in Figure 5 clearly converge to the correct value.

We also tested the scalability of the algorithm by trying it out on various types of networks of increasing size. As an example, let us consider networks with n professors and $4n + 1$ students, each of whom could have any one of the professors as advisor. We specified the success (or otherwise) of $4n$ students and queried the success of the last. Figure 6 shows the inference cost as a function of the total network size. Because we cannot compute the exact values for all n, we use a standard convergence diagnostic due to [Gelman, 1996], checking against the exact values for small n. The resulting graph appears to be linear in the size of the network. It would appear that, for this type of network, the sampling algorithm scales well. Note also that cost is measured in terms of *single-variable* state transitions performed by the algorithm, hence the number of transitions *per variable* to reach convergence is approximately constant, regardless of network size. This held in all our experiments, including experiments (not reported here) on genetic inheritance with uncertain parentage, where the number of cascaded reference variables increases with n.

5 Identity uncertainty

Standard relational probability models [Pfeffer, 2000] incorporate the unique names assumption, i.e., the assumption that each instance present in the knowledge base corresponds to a different object in any given possible world. When this assumption is removed, we must consider the possibility that several instances may denote the same object. Identity uncertainty is especially prevalent in settings where an agent perceives multiple objects over time [Pasula *et al.*, 1999], but also occurs in almost all real databases where "duplicate" records abound.

With identity uncertainty, a "possible world" must specify not only the attribute values for all objects, but also the mapping from instances in the knowledge base to objects in the possible world. (Without identity uncertainty, this mapping is one-to-one and hence no distinction between instances and objects is needed.) We represent this mapping using an equivalence relation—a set of equivalence classes, each of which contains all the instances that co-refer in that particular possible world.[1]

Consider the following example, a simplified version of the data association problem studied by [Pasula *et al.*, 1999]. There are two classes, $Vehicle$ and $Observation$. There is one complex attribute, $generated\_by$, which maps an $Observation$ to the $Vehicle$ that generated it. Each $Observation$ of a vehicle reports a $colour$, which depends (through a probabilistic model of the sensing process) on the $colour$ of the corresponding $Vehicle$. Instances are added to the KB as follows: whenever a vehicle is detected at some

[1] Note that it doesn't matter *which* object the instances denote—objects are essentially placeholders in a relational structure.

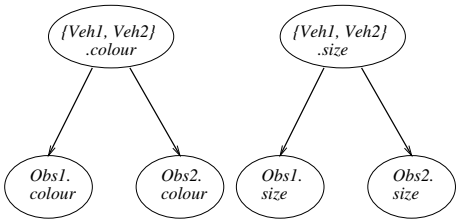


Figure 7: Bayesian network structure for the two-vehicle case, where the vehicles are currently assumed to be one and the same object.

sensor, an $Observation$ is instantiated with its colour set to the measured value, and with a new $Vehicle$ instance attached. In this domain, inference involves reasoning about which $Vehicle$s are, in fact, identical.[2]

Figure 7 shows the Bayesian network structure generated when the knowledge base contains observations Obs_1 and Obs_2 of vehicles Veh_1 and Veh_2, and when the equivalence relation on instances is set to $\{\{Veh_1, Veh_2\}\}$ (i.e., Veh_1 and Veh_2 are the same object).

5.1 Defining a probability distribution

Each possible world now contains an equivalence relation $\omega \in \Omega$ as well as the attribute values, and the knowledge base must now specify a unique probability distribution over this enlarged space of worlds. We can write

$$P(\omega, \langle\text{attribute values}\rangle) = P(\langle\text{attribute values}\rangle|\omega)P(\omega) \quad (2)$$

Given each fixed ω, identity uncertainty is eliminated, and the probability distribution over the attributes can be defined as before. All that remains is to specify the prior $P(\omega)$. When doing so, we can use the fact that instances in disjoint classes cannot co-refer—for example, vehicles cannot be observations. $P(\omega)$ can thus be factored into terms dealing with each of the classes. Even then, however, the state space of each factor is exponential in the number of objects in its class. Fortunately, there are more compact ways of expressing the ω-distribution for each class. The simplest is to give an explicit distribution over the *number* of objects in each class, which can be done by adding a number attribute to classes in the knowledge base. In more complex situations, such as full data association, the ω-distributions are specified implicitly in conjunction with object arrival and detection models [Pasula *et al.*, 1999]. The general topic of modular specification of ω-distributions is likely to require a good deal of further research.

5.2 Inference with identity uncertainty

In the presence of identity uncertainty, exact inference calls for a summation over all possible values of ω. This is clearly infeasible when $|\Omega|$ is large—and $|\Omega|$ is exponential in the number of instances in identity-uncertain classes. MCMC permits us to replace the summation with a sample. The algorithm now works as follows. In addition to "normal transitions," which run as before given a fixed ω, the algorithm performs "identity transitions" in which ω changes. Because of the large state space of ω, the latter type of transition uses M-H steps. Moreover, these steps propose changes not only to ω, but also to some of the conventional attributes.

The reason for this is that an identity transition can propose (among other things) that two currently separate instances be grouped together in the same equivalence class. Clearly, if these two instances currently have different values for the same attribute, the probability of the destination state will be zero. Hence, identity transitions will have a chance of occurring only when the instances "line up"—that is, a sequence of normal transitions causes all of their attribute values to match exactly. When instances have many attributes, line-up may be very rare, resulting in a low acceptance ratio and a slow algorithm. Moreover, there are cases in which waiting for line-up makes the Markov chain non-ergodic—the algorithm can be trapped in a subset of possible ωs with no escape.

Identity transition proposals

Consider a current assignment ω, and a proposed assignment ω'. Let u be the set of unobserved attributes of all the objects in ω affected by the transition, and let u' be the unobserved attributes of the corresponding objects in ω'. Let o be the set of all other attributes within the Markov blankets of the attributes in u or u': this may include observed attributes, as well as the unobserved attributes of other objects not affected by the transition. Note that all attributes in o remain fixed throughout the transition.

Each identity transition proceeds as follows:

- ω' is chosen using a proposal $q_I(\omega'|\omega)$. This proposal can take many forms: one simple possibility is to suggest that a randomly selected instance move from one equivalence class to another. Another method might suggest merging two equivalence classes, or splitting an equivalence class in two.
- Values for all of u' are chosen using a proposal $q_U(u'|\omega', \omega, u, o)$. Some possible choices for q_U will be explained below.
- The proposed change is accepted with probability

$$min\left(1, \frac{\pi(\omega')q_I(\omega|\omega')\pi(o, u'|\omega')q_U(u|\omega, \omega', u', o)}{\pi(\omega)q_I(\omega'|\omega)\pi(o, u|\omega)q_U(u'|\omega', \omega, u, o)}\right)$$

  which is derived from Equation (3) using Equation (2) and the two-step nature of our proposal.

As an example, let us consider the world state portrayed in Figure 7. In this situation,

- $\omega = \{\{Veh_1, Veh_2\}\}$
- $u = \{\{Veh_1, Veh_2\}.colour, \{Veh_1, Veh_2\}.size\}$

A split can now be proposed as follows:

- $\omega' = \{\{Veh_1\}, \{Veh_2\}\}$
- $u' = \{Veh_1.colour, Veh_2.colour, Veh_1.size, Veh_2.size\}$

To ensure that the chain is ergodic, new values must then be proposed by q_U for at least the newly created attributes.

[2] An alternative interpretation for this knowledge base, more familiar to statisticians, is that there are an unknown number of balls in a bag; balls are pulled out one at a time, their colour is measured, and they are replaced.

Choosing the attribute values

One possible q_U proposal picks a value uniformly at random from the values available at each attribute. This approach is simple to implement and yields an acceptance ratio that is easy to calculate. The $q_U(u|\ldots)$ terms reduce to $1/\sum_{u_i}|u_i|$. This is the algorithm that we test experimentally below.

Unfortunately, this q_U proposal may suggest very unlikely value combinations for the attributes, resulting in low acceptance ratios. We might be better off with a more intelligent proposal mechanism, such as likelihood weighting which generates samples from $q_U(u|o,\omega) = \prod_{x_i \in u} P(x_i|Pa(x_i),\omega)$ together with weights $w(u|o,\omega) = \prod_{x_i \in o} P(x_i|Pa(x_i),\omega)$. Since

$$\pi(u|\omega,o) = \frac{w(u|o,\omega)q_U(u|o,\omega)}{\sum_U w(u=U|o,\omega)q_U(u=U|o,\omega)}$$

the fraction in the acceptance ratio can now cancel to give

$$\frac{\pi(\omega'|o)q_I(\omega|\omega',o)w(u'|o,\omega')\sum_U w(U|o,\omega)q_U(|o,\omega)}{\pi(\omega|o)q_I(\omega'|\omega,o)w(u|o,\omega)\sum_{U'} w(U'|o,\omega')q_U(U'|o,\omega')}$$

This ratio is no longer so simple, as performing the summations may be quite time-consuming. Fortunately, whenever the set of uncertain attributes can be split into several d-separated sets, the summations can be rewritten as products of summations over these sets. If the sets are all small, this algorithm may be feasible.

Yet another approach is to run a "local" MCMC algorithm inside q_U to generate attribute values that reflect the current values of o and hence are likely to be accepted. (When this method is used, care must be taken to ensure that the chain is run until convergence: stopping it prematurely may result in biased answers.)

5.3 Experimental results

To check our identity uncertainty algorithm, we first performed experiments analogous to those in Section 4.3. We began with a problem that was small enough for exact calculation: it consisted of five observations with known attribute values, and queried the posterior probability of the vehicle generating the first observation. Figure 8 shows that our algorithm's estimates converge to the correct value over time.

We then tested the scalability of the algorithm by, once more, applying it to randomly generated networks of increasing size, and measuring time to convergence using a standard diagnostic. The results shown in Figure 9 were obtained by constructing larger and larger sets of vehicle–observation pairs, and requesting a posterior distribution over the number of vehicles. As before, the algorithm appears to scale well.

Finally, we performed an additional experiment to demonstrate that our algorithm works as expected. It should be true that, if observations are generated from a specific set of vehicles, increasing the number of observations in the knowledge base will enable us to model characteristics of that set of vehicles with increasing accuracy. Figure 10 shows that adding more and more observations does enable us to infer the number of vehicles responsible for their generation.

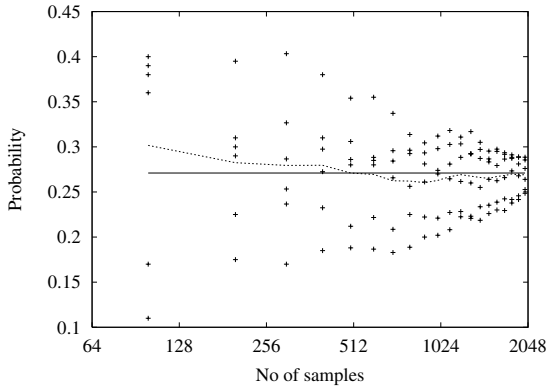


Figure 8: MCMC convergence on a simple example with five observed vehicles. The exact probability is 0.271 (horizontal line). Each point represents the estimate from one of 6 Markov chains after a given number of samples (log scale). The dotted line shows the average of the 6 individual estimates.

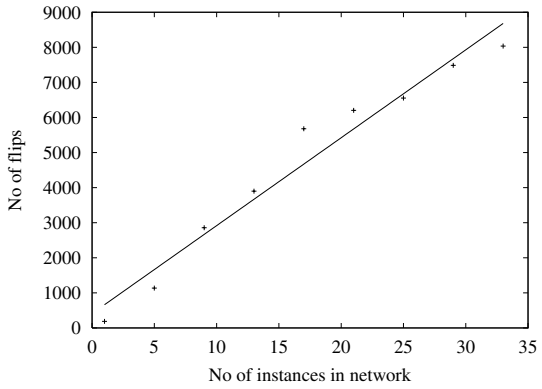


Figure 9: MCMC convergence as a function of network size. The y-axis measures the number of state transitions required to reach a preset convergence threshold for a standard diagnostic. The straight line is a regression fit to the data, which is averaged over 30 trials.

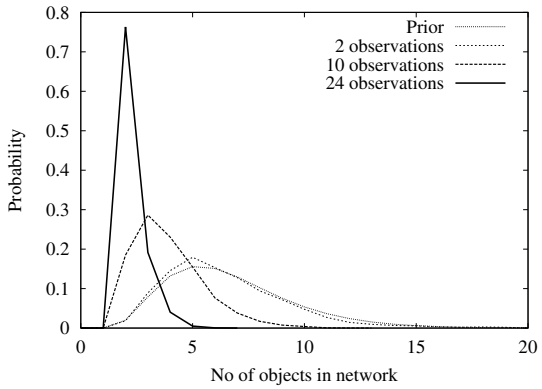


Figure 10: Distributions over the number of objects in the network. The prior distribution is shown; note that it insists that there are at least two vehicles. Observations are then generated from exactly two vehicles. When given just two observations, the algorithm converges to a distribution close to the prior, as expected. As more observations are added, the result moves closer to the true value of two vehicles.

6 Conclusions and further work

We have proposed an algorithmic approach for inference in first-order probabilistic languages, based on Markov chain Monte Carlo. Our preliminary investigations suggest that the approach is very promising. We believe that it can significantly increase the expressive power of languages that can be considered "practical" for knowledge representation and reasoning under uncertainty. For example, the lazy exploration approach should make it possible to handle infinite recursive models, including temporal models, with somewhat greater generality than so far envisaged [Koller and Pfeffer, 2000].

The Metropolis–Hastings algorithm allows for a wide variety of proposal distributions. In particular, *intelligent proposals* can be used to focus computation. One can easily construct *data-driven* and *query-driven* proposals, analogous to forward and backward chaining in logical systems, that essentially result in "activation" spreading outwards from query and evidence variables. These concepts can be subsumed by a general mechanism for proposing transitions based on the expected value of computation [Russell and Wefald, 1991]. We can also insert arbitrary domain-specific knowledge into the proposal mechanism—for example, proposing candidate advisors based on the student's research interest. Such knowledge-based proposals, which result in faster convergence, can also be *learned* via so-called *adaptive proposals*—that is, allowing the proposal distribution q to change over time based on the algorithm's experience in generating samples and computing acceptances.

Clearly, we have just scratched the surface of this topic. A large effort is needed to apply first-order probabilistic languages to real problems, in order to identify useful language features, common representation structures, and their effect on inference. Having a flexible and general—albeit sometimes slow—MCMC inference engine should help with this task. Finally, we also need to develop complexity results for approximate inference, using tools such as those provided by [Jerrum and Sinclair, 1997].

References

[Gelman, 1996] Andrew Gelman. Inference and monitoring convergence. In W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors, *Markov Chain Monte Carlo in Practice*, pages 131–143. Chapman and Hall, London, 1996.

[Gilks *et al.*, 1996] W.R. Gilks, S. Richardson, and D.J. Spiegelhalter, editors. *Markov chain Monte Carlo in practice*. Chapman and Hall, London, 1996.

[Halpern, 1990] J. Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3):311–350, 1990.

[Jerrum and Sinclair, 1997] M. Jerrum and A. Sinclair. The Markov chain Monte Carlo method. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*. PWS Publishing, Boston, 1997.

[Koller and Pfeffer, 1998] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, Wisconsin, July 1998. AAAI Press.

[Koller and Pfeffer, 2000] D. Koller and A. Pfeffer. Semantics and inference for recursive probability models. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, Austin, Texas, July 2000. AAAI Press.

[Pasula *et al.*, 1999] Hanna Pasula, Stuart Russell, Michael Ostland, and Ya'acov Ritov. Tracking many objects with many sensors. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, Sweden, August 1999. Morgan Kaufmann.

[Pearl, 1988] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California, 1988.

[Pfeffer *et al.*, 1999] A. Pfeffer, D. Koller, B. Milch, and K.T. Takusagawa. SPOOK: A system for probabilistic object-oriented knowledge representation. In *Uncertainty in Artificial Intelligence: Proceedings of the Fifteenth Conference*, Stockholm, August 1999. Morgan Kaufmann.

[Pfeffer, 2000] Avrom J. Pfeffer. *Probabilistic Reasoning for Complex Systems*. PhD thesis, Stanford University, Stanford, California, 2000.

[Russell and Wefald, 1991] Stuart J. Russell and Eric H. Wefald. *Do the Right Thing: Studies in Limited Rationality*. MIT Press, Cambridge, Massachusetts, 1991.

[Russell, 1999] Stuart Russell. Expressive probability models in science. In *Proc. of the 2nd Int'l Conf. on Discovery Science*, Tokyo, Japan, December 1999. Springer Verlag.

[Sato and Kameya, 1997] T. Sato and Y. Kameya. PRISM: A symbolic-statistical modeling language. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1330–1335, Nagoya, Japan, August 1997. Morgan Kaufmann.

[Wellman *et al.*, 1992] M. P. Wellman, J. S. Breese, and R. P. Goldman. From knowledge bases to decision models. *Knowledge Engineering Review*, 7(1):35–53, March 1992.

[Zhang and Poole, 1996] N.L. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328, 1996.

Knowledge Processing under Information Fidelity

Wilhelm Rödder

FernUniversität Gesamthochschule in Hagen
D-58084 Hagen, Germany
wilhelm.roedder@fernuni-hagen.de

Abstract

XSPIRIT is a professional expert system-shell for knowledge acquisition, inference and response using conditional logic and probability. Composed conditionals on propositional variables with finite domain are the communication tool between the user and the knowledge base, making the process of acquisition, inference and query comfortable and intelligible. XSPIRIT allows partial rather than complete information about the knowledge domain and supplements missing parts by the principle of information fidelity. By virtue of evident temporary information, knowledge undergoes a well-defined adaptation process, respecting this principle again. The construction and transformation of probability distributions as developed here, allow acquired knowledge, remaining uncertainty and strength of inference to be measured in the information units [bit]. XSPIRIT allows large-scale applications with hundreds of composed conditionals and umpteen variables.

1 Introduction

Humans memorise and deduce facts »in a conditional environment«. »If *A* then *B*« is a constituent of human knowledge. Such knowledge pieces together with concatenation mechanisms build a highly netted memory, the knowledge base. Once situativ things become evident, our knowledge undergoes a complex adaptation process resulting in conclusions. Conclusions cause reactions or response.

Probability distributions on event fields are a suitable means to model such deduction processes in a knowledge base. Bayes nets are the graphical structure which might help to organise knowledge and the corresponding distributions, and make them manageable, cf. [Lauritzen and Spiegelhalter, 1988] or [Pearl, 1988]. Bayes nets allow to break down the global distributions to local marginals, at the cost of conditional independence assumptions which rarely meet reality. Inference in Bayes nets then is the technically complex but unpretentious calculation of conditioned probabilities. We do not follow this idea, but want to enable the user to formulate any local knowledge pieces without forcing him to construct a directed acyclic graph. After an automatic aggregation of these pieces to a knowledge base he can adapt this knowledge to any situativ modification.

Thus, different from the established form of a probability model, we shall discuss an inference process which

- allows partial rather than complete information about the probability distribution. The missing parts are completed by the principle of information fidelity which turns out to coincide with the principle of minimum relative entropy.
- permits inference from virtual evidence. Vague assumptions about the actual situation can be processed as well as sure things.
- allows the user to »communicate« with the distribution in the comfortable language of composed conditionals.

There is a lot of related work to our paper in literature. To minimize relative entropy when adapting the dependency structure in a distribution to new information was discussed already in early works [Jaynes, 1978], [Cheeseman, 1983], and justified axiomatically [Shore and Johnson, 1980], [Paris and Vencovská, 1990], [Csiszár, 1991], [Kern-Isberner, 1998]. Even the idea to use such adaptation for inference is not new, see [Cheeseman, 1983], [Hájek et al., 1992], [Goldszmidt et al., 1993], [Rödder and Meyer, 1996].

All probabilistic inference presented here, however, is of purely information theoretic nature. The quantity of acquired knowledge can be measured in [bit], as well as the strength of evidence. Knowledge is reduced uncertainty, inference means uncertainty adaptation to new information. And even query and response are information demand and supply.

In section 2 we develop the language of composed conditionals as a communication tool between user and knowledge base, in section 3 we analyse the relations between probability and information, in section 4 we go through the entire process of knowledge acquisition, inference and response, as realized in the expert system shell XSPIRIT. A small example emphasises the conceptual difference between Bayes nets and the inference method developed here. Section 5 gives a short technical introduction to the shell and invites the reader to test it.

2. Conditionals and Probability

2.1 Conditionals in a Three-valued Logic

Concerning the semantics we consider a population of individuals or objects with their respective properties. Our intention is the description of relations between these properties. As far as the syntax concerns we model the knowledge domain as a finite set of finite-valued variables $V = \{V_1,\ldots,V_L\}$ and the respective event algebra F on the elementary events $v = v_1...v_L$, here v_l are values of V_l. As usual Ω is the all- and $\varnothing$ the empty event.

Each event from F can be identified in a natural way with a propositional sentence, built from literals $V_l = v_l$ by the connectives negation, conjunction and disjunction, cf. [Rödder and Kern-Isberner, 1997]. Simple conjuncts of such literals we often write as unordered tuples of the respective values, $v_1...v_l$, for instance. Hence each event may be expressed in an intelligible way, which we shall demonstrate by example 1 below. We do not distinguish between events and propositional sentences, further on.

Generic events from F or the corresponding propositions we denote as capital letters, indexed if necessary: A, B, ... , A_i, B_i, ... , E_i, F_i. If $v \subset A$, we call the proposition A true in the world v and false, otherwise: $A(v)$ = t or $A(v)$ = f. If A is true (false) on the entire Ω, we write A = t (A = f). The set of all propositional sentences forms the language L.

Calabrese [Calabrese, 1991] and earlier de Finetti [deFinetti, 1972] go a step further and define conditionals $B|A$ for B and A from L. Calabrese in [Calabrese, 1991] and [Calabrese, 1994] derives a rich language of composed conditionals. We go through that in more detail now.

The exact definition of a conditional reads

$$B|A(v) = \begin{cases} \text{t} & \text{if} \quad v \subset AB \quad (= A \wedge B) \\ \text{f} & \text{if} \quad v \subset A\overline{B} \quad (= A \wedge \overline{B}) \\ \text{u} & \text{if} \quad v \subset \overline{A}. \end{cases} \tag{1}$$

Barring means negation and t(rue), f(alse), u(ndefined) are values in a three-valued logic, cf. [Rescher, 1969]. In this logic the connectives $-,\wedge,\vee$ apply for t, f as usual and for u as follows:

$$\text{t} \wedge \text{u} = \text{t} \vee \text{u} = \text{t},\ \text{f} \wedge \text{u} = \text{f} \vee \text{u} = \text{f},\ \text{u} \wedge \text{u} = \text{u} \vee \text{u} = \text{u},\ \overline{\text{u}} = \text{u}. \tag{2}$$

The truth-value *undefined* does not change neither t, f nor u under conjunction or disjunction.

The set of all conditionals $B|A$ is the language $\mathsf{L}|\mathsf{L}$. Identification of $B|\Omega$ with B embeds L in $\mathsf{L}|\mathsf{L}$. Similar to above we write $B|A$ = t|A (= f|A) if the conditional is true (false) on the whole A.

Following Calabrese further, we compose conditionals by $-,\wedge,\vee$ and | to get a very rich linguistic structure for the communication between the user and the knowledge base, as shall be developed in the remainder of this section.

With the conventions in Reschers three-valued logic (2), the author in [Calabrese, 1991] defines pointwise:

- $\left[(B|A) \wedge (D|C)\right](v) = (B|A)(v) \wedge (D|C)(v)$,
- $\left[(B|A) \vee (D|C)\right](v) = (B|A)(v) \vee (D|C)(v)$, (3)
- $\overline{(B|A)}(v) = \overline{(B|A)(v)}$.

By these pointwise attributions any conjunct, disjunct or negation of conditionals is well-defined also globally.

Let the conditional operator | apply as in the following scheme:

|	t	f	u
t	t	u	t
f	f	u	f
u	u	u	u

(read e.g. f | u = f). (4)

With this convention a pointwise definition of a conditioned conditional reads:

- $\left[(B \mid A)(D \mid C)\right](v) = (B|A)(v) \,\big|\, (D|C)(v)$. (5)

Pointwise attribution of truth-values again permits the global definition of conditioned conditionals on Ω. More on the hierarchy of composed conditionals the reader might find in Calabrese's original paper, as well as the proof of the following theorem 1. Theorem 1 permits the reduction of composed conditionals to simple ones.

Theorem 1 (Composed conditionals)
With the conventions in (2), (4) and the definitions in (3), (5) the following equations hold:

- $(B|A) \wedge (D|C) = \left[(B \vee \overline{A})\,(D \vee \overline{C})\right] \,|\, A \vee C$,
- $(B|A) \vee (D|C) = (AB \vee CD) \,|\, A \vee C$,
- $(B|A) \,|\, (D|C) = B|A(D \vee \overline{C})$.

Composed conditionals are reducible to simple ones, but in their composed form allow an immense linguistic richness, as demonstrates the following example.

Example 1 (Linguistics and composed conditionals)
With the use of mnemonic letters the following propositions should be self explicatory:

- $\left[(M \,|\, (C_1 \wedge C_2)) \wedge ((C_1 \wedge C_2) \,|\, M)\right](v)$,
  in the world v machine M is o.k. iff the component C_1 is o.k. and C_2 is o.k.
- $\left[((G \vee A) \wedge (B|G)) \,|\, S\right](v)$,
  in the world v Herr Schmid is German or Austrian, if he is German he comes from Berlin.
- $\left[(C \vee (V \,|\, \overline{C})) \,|\, L\right](v)$,
  in the world v Mr. Lee is Chinese, if not, he is Vietnamese.

The reader might notice that we can make global considerations rather than »in the world v«, too. In a certain population or knowledge context, the machine M is o.k. iff C_1 and C_2 are o.k. with a probability of 90%, say. Probabilities of conditionals are constituents of our AI-concept. More on that in the next sections, cf. also [Rödder, 2000]. Because of theorem 1 it suffices to consider simple conditionals of the form $B|A$ further on. Bear in mind, however, that they represent the total hierarchy of arbitrarily composed conditionals and their respective linguistic pendants.

In what follows we define sets of conditionals with certain properties. These properties make them »a conditioning partition« on Ω.

Definition 1 (c-independent and disjoint conditionals)

i.) $B|A$ is conditionally or c-independent of $D|C$ iff $(B|A)|(D|C) = B|A$, and c-dependent otherwise.

ii.) $B|A$ and $D|C$ are disjoint iff $ABCD$ = f.

Definition 2 (Conditional basic set CBS)

i.) A set S of conditionals is sufficient for Ω, if any $v \subset \Omega$ can be expressed as a conjunct of elements from S.

ii.) A conditional basic set CBS is a set of conditionals sufficient for Ω such that, for any $B|A$, $D|C \in$ CBS one of three statements holds:
$B|A$ and $D|C$ are disjoint,
$B|A$ is c-independent of $D|C$,
$D|C$ is c-independent of $B|A$.

The following example gives an idea of conditional basic sets.

Example 2 (Conditional basic sets)

i.) $\{v = v_1...v_L\}$ is a CBS.

ii.) $\{v_1\} \cup \{v_2|v_1\} \cup ... \cup \{v_L|v_{L-1}...v_1\}$ is a CBS. Note that conditionals "more to the right" are c-independent of those "more to the left", if they include no distinct values $v_l \neq v_l$'. If they do include distinct values, they are disjoint. Any v can be resolved into "factors" $v = v_1 \wedge (v_2|v_1) \wedge ... \wedge (v_L|v_{L-1}...v_1)$. To show this, use (1) and (2).

iii.) $\{v_l...v_1\} \cup \{v_L...v_{l+1} \,|\, v_l...v_1\}$ is a CBS, as shows a similar reasoning like that in ii).

iv.) ii) or iii) for an arbitrary permutation $l_1...l_L$ of 1….L is a CBS.

Conditional basic sets will serve as a means for measuring the intrinsic uncertainty in a probability distribution on Ω and this measure will turn out to be independent of the special choice of the CBS.

2.2 The Probability of Conditionals

A probability measure P on Ω can be generalized to conditionals by the natural requirement, that for the conjunct of c-independent conditionals the probabilities should multiply. As $B|A$ is c-independent of A and because $(B|A) \wedge A = BA$, it follows immediately $P(BA) = P((B|A) \wedge A) = P(B|A) \cdot P(A)$ and hence $P(B|A) = P(BA) / P(A)$, if only $P(A) > 0$. The probability of a conditional is its conditioned probability.

From the logical factorisation

$v = v_1 \wedge (v_2|v_1) \wedge ... \wedge (v_L|v_{L-1}...v_1)$ (cf. example 2)

we get the well-known probabilistic factorisation

$P(v) = P(v_1) \cdot P(v_2|v_1) \cdot ... \cdot P(v_L|v_{L-1}...v_1)$.

Probabilistic independence reflects the logical-based c-independence, as it should be.

3. Probability, Uncertainty and Information

3.1 Basic Considerations

Entropy first was established by the physicists Carnot and Clausius in thermodynamics. It measures the portion of thermal energy which cannot be transformed back into mechanical energy. When the engineer Shannon [Shannon, 1948] founded information theory, he looked for an adequate denotation of $H(P) = -\sum_v P(v) \cdot ld\ P(v)$, where *ld* is the dual logarithm and 0· ld 0 = 0. In this theory signals v vary in a finite alphabet $v \in V$ and H measures the average information rate which can be transmitted from a source via a channel to a receiver. John v. Neumann suggested to name it *entropy*, following an anecdote in the Scientific American [Horgan, 1990] because "Shannon would have an advantage in debates about this theory."

In the remainder of this section we shall study entropy further.

If we learn that a proposition A with probability $P(A)$ becomes true, we receive the information $-ld\ P(A)$ [bit]. Information is reduction of uncertainty, any textbook of information theory goes through a proof of this fact. We learn that a conditioned proposition $B|A$ becomes true if BA becomes true, cf. (1). Though the reduced uncertainty is not $-ld\ P(BA)$ but rather $-ld\ P(BA) - (-ld\ P(A))$. This result comes from the additivity of uncertainty reduction for c-independent conditionals. We put it as a lemma and leave the obvious proof to the reader.

Lemma 1 (Information of c-independent conditionals) If for any c-independent $B|A$ of $D|C$ or $D|C$ of $B|A$, the uncertainty reduction $inf\ (B|A \wedge D|C)$ equals $inf\ (B|A) + inf\ (D|C)$, it follows $inf\ (B|A) = -ld\ P(BA) - (-ld\ P(A))$.

Let us now consider a similar situation as before, but – for whatever reason – our situativ conviction of $B|A$ to become true, has changed from $P(B|A)$ to $Q(B|A)$. This change is change of uncertainty and the difference equals

$\Delta inf = -ld\ P(B|A) - (-ld\ Q(B|A)) = ld\ (Q(B|A) / P(B|A))$,

cf. figure 1.

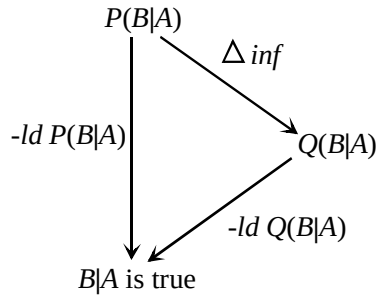


Figure 1: Sequential information gain

Note that the expectation of this uncertainty change is $Q(BA) \cdot ld\ (Q(B|A) / P(B|A))$, as $Q(BA)$ is the probability of $B|A$ to become true in the light of our new conviction.

3.2 Information Adaptation

In this section we study the adaptation of a distribution P to new information, supplied in the form of conditionals from $\mathsf{L}|\mathsf{L}$ and their respective desired probabilities. To distinguish between such conditionals and the elements of a CBS, we use lowercase letters $f|e$ for the latter. There should be no problems with this notation.

The following theorem shows that the overall expected uncertainty change does not depend on a specific CBS.

Theorem 2 (Unique uncertainty change)
Let CBS and CBS' be two arbitrary conditional basic sets with generic conditionals $f|e$ and $f'|e'$, respectively. Let Q and P be two distributions on Ω, such that Q is totally continuous with respect to P. Then the following equality holds ($0 \cdot ld\,(0/0) = 0$):

$$\sum_{f|e \in CBS} Q(fe) \cdot ld\,(Q(f|e)/P(f|e)) = \sum_{f'|e' \in CBS'} Q(f'e') \cdot ld(Q(f'|e')/P(f'|e')).$$

Proof:

$$\sum_{f|e \in CBS} Q(fe) \cdot ld\,(Q(f|e)/P(f|e)) =$$

$$\sum_{f|e \in CBS} \left(\sum_{v \subset fe} Q(v) \right) \cdot ld\,(Q(f|e)/P(f|e)) =$$

$$\sum_{v} Q(v) \cdot \left(\sum_{f|e \text{ with } v \subset fe} ld\,(Q(f|e)/P(f|e)) \right).$$

For each v the inner summation runs through conditionals, which are not disjoint and hence pairwise c-independent (one independent of the other). Because a CBS is sufficient we even have

$$v = \bigwedge_{v \subset fe} f|e\,.$$

As furthermore logical independence implies probabilistic independence we get

$$\sum_{v} Q(v) \cdot ld \prod_{f|e \text{ with } \bigwedge f|e = v} \left(Q(f|e)/P(f|e)\right)$$

and finally $\sum_{v \in \Omega} Q(v) \cdot ld\,(Q(v)/P(v))$.

The CBS was chosen arbitrarily, which completes the proof.

The basic idea of the proof was to show that either side of the equation equals $\sum_{v \in \Omega} Q(v) \cdot ld\,(Q(v)/P(v))$.

This expression is known as relative entropy $R(Q,P)$ between P and Q or directed divergence from P to Q. Expected uncertainty change from P to Q in any CBS equals the relative entropy between P and Q.

If $P = P^0$ is the uniform distribution, $H(Q) = ld\,|\Omega| - R(Q,P^0)$ is the (absolute) entropy of Q. Thus the uncertainty change from P^0 to Q is also $ld\,|\Omega|$ - H(Q). Both R and H measure in [bit].

As mentioned at the beginning of the present section we now study the problem of adapting the distribution P to new information supplied as a set of conditionals with desired probabilities $x_i \in [0,1]$: $\mathfrak{R} = \{B_i \mid A_i[x_i]\}$.

We demand this adaptation to be such that the former uncertainty structure in P is preserved as far as possible, i.e. we solve

$$\overline{P} = arg\,min\,\mathrm{R}(Q,P) \text{ s.t. } Q(B_i \mid A_i) = x_i\,,\ i = 1...I, \qquad (6)$$

and call $\overline{P}$ the $\mathfrak{R}$-adaptation of P. As (6) preserves the expected uncertainty reduction for any CBS as far as possible, it obeys the *principle of information fidelity*. There are axiomatic accesses to conditional knowledge adaptation, cf. [Csisár, 1991], [Shore and Johnson, 1980], [Paris and Vencovská, 1990], [Kern-Isberner, 1998]. The objective of our approach is to focus upon a cautious information work up. With the presentation of all logical, probabilistic and information-theoretic prerequisites we are now ready to develop knowledge processing as realized in XSPIRIT.

4 Knowledge Acquisition, Inference and Response

Probabilistic knowledge processing in XSPIRIT is performed in the four steps: Initialisation, knowledge acquisition, inference and response. We shall go through these steps in detail now.

Initialisation
Here we commit ourselves to a knowledge domain by defining the variables V_l and their respective values v_l. This determines $\Omega = \{v\}$ and the event algebra F. The uniform P^0 on F means absolute ignorance on F, as the variables V_l are mutually independent from each other.

Knowledge Acquisition
The knowledge engineer supplies (composed) conditionals and their respective desired probabilities $\mathfrak{R} = \{B_i|A_i\,[x_i], i = 1\ldots I\}$. $B_i|A_i\,[x_i]$, $i = 1\ldots I$ is the imperative to adapt P^0 to $\mathfrak{R}$. As developed in the last section we solve

$$P^* = arg\,min\,\mathrm{R}(Q,P^0) \text{ s.t. } Q(B_i \mid A_i) = x_i\,,\ i = 1...I. \qquad (7)$$

P^* is the adapted knowledge following the principle of information fidelity. $R(P^*, P^0)$ measures the knowledge increment in [bit]. Remind that minimizing relative entropy is equivalent to maximizing entropy, and verify the equation $H(P^0) - R(P^*, P^0) = H(P^*)$. Actual uncertainty is maximum uncertainty minus acquired knowledge.

Inference
The user supplies (composed) conditionals and their respective temporarily evident probabilities $\mathfrak{E} = \{F_j|E_j\,[y_j], j = 1\ldots J\}$. $F_j|E_j\,[y_j]$, $j = 1\ldots J$ is the imperative to find a probability distribution which meets evidence. Evidence is not additional knowledge to be acquired, but describes a

temporary reflection about things that might happen. "If very likely with y = 90 % a person's nationality were German, which conclusions could we infer from that?" Evident conditionals and their probabilities in almost all cases are contradictory to the knowledge supplied by $\mathfrak{R}$! Of course in our knowledge domain there is no 90 % probability to be German, in general. Because of the observations in the previous section we calculate

$$P^{**} = arg\ min\, \mathrm{R}(Q, P^{*}) \text{ s.t. } Q(F_j \mid E_j) = y_j,\ j = 1...J. \quad (8)$$

P** has minimal directed divergence from P*, subject to evidence. Among all Q, P** is the distribution with the minor change of expected uncertainty relative to P*. R(P**,P*) is the strength which evidence puts upon P* to infer P**, and measures in [bit].

Response
The user wants, in the evident situation, the knowledge base to answer a question, which also might be of the conditional form $(H|G)$. The response is $P^{**}(H|G)$. Response is inferred from the knowledge base $\mathfrak{R}$ **and** from evidence $\mathfrak{E}$.

While (7) and (8) is inference, as (former) knowledge is adapted to new information in a sophisticated transformation process, response is not. Response is the calculation of a conditional probability and hence the mere evaluation of a distribution.

W. Rödder and G. Kern-Isberner in [Rödder and Kern-Isberner, 1997] discuss the example Lea Sombé of people in a fictitious society, which originally was presented in [Sombé, 1992].

Example 3 (Lea Sombé)
In a fictitious society we consider the properties: to be a student or not $S = s / \bar{s}$; to be young or not $Y = y / \bar{y}$; to have the marital status single, married or corporate life M = s, m, c; to be parent or not $P = p / \bar{p}$. The following information is given:

- about 90 % of all students are young,
- about 80 % of all young people are single,
- 70 % of singles are young,
- 30 % of young people study,
- members of the society which live in corporate lives, are young with 80 %,
- students with children, in 90 % of all cases are married or live in corporate lives.

In the notation of the present paper we have the conditionals:

$$(Y = y \mid S = s)[.9], \quad (M = s \mid Y = y)[.8], \quad (Y = y \mid M = s)[.7],$$
$$(S = s \mid Y = y)[.3], \quad (Y = y \mid M = c)[.8], \quad (\overline{M = s} \mid S = s \wedge P = p)[.9].$$

After solving (7) to determine P*, we want to make conclusions about Lea Sombé who evidently is a student and has a child. Is she young? Calculating P** in (8) under certain evidence and evaluating $P^{**}(Y=y)$ yields 81 %. Lea Sombé very likely is young.

The information in P* only counts 1.05 [bit], the remaining uncertainty about the fictitious society is high, namely 3.53 [bit]. Imposing evidence on P* results in an information flow of 1.62 [bit]. All results were calculated with the shell XSPIRIT.

Besides of its simplicity example 3 is not accessible for Bayes nets, for two reasons:

- information about the society is incomplete inasmuch as it does not fully determine a probability distribution,
- supplied conditioned probabilities do not build a directed acyclic graph.

You can handle virtual evidence and augment the number of variables and conditionals almost arbitrarily. XSPIRIT is able to treat hundreds of composed conditionals on umpteen variables, as we shall relate in the next section.

5 The Shell XSPIRIT

The shell XSPIRIT 3.0, a JAVA-version, is a professional tool for knowledge processing under information fidelity. To acquire knowledge the user defines variables and supplies conditionals in a comfortable syntax. The syntax permits conjunctions, disjunctions and negations of literals and their respective composed conditionals, in an arbitrary hierarchy. Variables might be boolean, nominal and ordinal. The attributes might be associated with real numbers (utilities) allowing the calculation of expected utility for respective decision models. The shell supports an interactive process to untie inconsistent information if provided by the user. For an application to on-line banking and credit worthiness confer [Kulmann and Reucher, 2000], for a business-to-business-approach see [Kulmann and Rödder, 2001].

Once all conditionals and their respective probabilities are supplied to the system, XSPIRIT automatically generates a suitable graphical structure, a hypergraph, and builds the knowledge base P* by virtue of corresponding marginal distributions on the hyperedges –sometimes called LEGs (local events groups). In the presence of evidence XSPIRIT supplies P** and evaluates any query, asked by the user. All communication is performed in the rich syntax of composed conditionals. P* and P** are calculated by a generalised Iterative Proportional Scaling (IPS) procedure, whose convergence was proved by Csiszár [Csiszár, 1975]. The system at any time informs about the amount of information already acquired and the remaining uncertainty in [bit]. For more mathematical details see [Rödder and Meyer, 1996].

Bayes nets are importable and XSPIRIT also provides a frequentistic learning process. If knowledge about the domain under consideration is available as a sample of elementary events rather than estimated probabilities x_i , the shell adopts this information in a separate module.

The reader interested in the shell XSPIRIT might visit the homepage

http://www.xspirit.de.

References

[Calabrese, 1991] P. G. Calabrese: *Deduction and Inference Using Conditional Logic and Probability*, in: Conditional Logic in Expert Systems, I. R. Goodman, M. M. Gupta, H. T. Nguyen and G. S. Rogers (editors). Elsevier Science Publishers B. V., 71 – 100, (1991).

[Calabrese, 1994] P. G. Calabrese: *Theory of Conditional Information with Applications*, in: Special Issue on Conditional Event Algebra, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 24, No. 12, 1676 – 1684, (1994).

[Cheeseman, 1983] P. Cheeseman: *A method of computing generalized Bayesian probability values for expert systems*, Proc. 6th Int. Joint Conf. on AI (IJCAI – 83), Karlsruhe, (1983).

[Csiszár, 1975] I. Csiszár: *I-divergence Geometry of Probability Distribution and Minimisation Problems*. The Annals of Probability, 3, No.1, 146 – 158, (1975).

[Csiszár, 1991] I. Csiszár: *Why Least Squares and Maximum Entropy? An Axiomatic Approach to Inference for Linear Inverse Problems*, The Annals of Statistics, 19 (4), 2032 – 2066, (1991).

[deFinetti, 1972] B. de Finetti: *Induction and Statistics*, Wiley, New York, (1972).

[Goldszmidt et al., 1993] M. Goldszmidt, P. Morris, J. Pearl: *A Maximum Entropy Approach to Nonmonotonic Reasoning*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 15, No. 3, 220-232, (1993).

[Hájek et al., 1992] P. Hájek, T. Havránek, R. Jirousek: *Uncertain Information Processing in Expert Systems*, CRC-Press, (1992).

[Horgan, 1990] J. Horgan: *Claude E. Shannon, Unicyclist, juggler and father of information theory*, PROFILE in: Scientific American, 16 – 17, (1990).

[Jaynes, 1978] E.T. Jaynes: *Where do stand on maximum entropy*?, in: The Maximum Entropy Formalism, R.D. Levine and M. Tribus (editors), Cambridge Mass., MIT-Press, (1978).

[Kern-Isberner, 1998] G. Kern-Isberner: *Characterising the principle of minimum cross-entropy within a conditional-logical framework*, Artificial Intelligence, Vol. 98, 169 – 208, (1998).

[Kulmann and Rödder, 2001] F. Kulmann and W. Rödder: *Probabilistische Modellbildung auf der Basis von Scoring-Schemata*, Proc. GOR Operations Research, Dresden, 477-482, (2001).

[Kulmann and Reucher, 2000] F. Kulmann and E. Reucher: *Computergestützte Bonitätsprüfung bei Banken und Handel*, Die Betriebswirtschaft (DBW), 60, 113-122, (2000).

[Lauritzen and Spiegelhalter, 1988] S.L. Lauritzen and D. J. Spiegelhalter: *Local computations with probabilities in graphical structures and their applications to expert systems*, Journal of the Royal Statistical Society, 13 (2), 415 – 448, (1988).

[Pearl, 1988] J. Pearl: *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, San Mateo Cal., (1988).

[Paris and Vencovská, 1990] J.B. Paris, A. Vencovská: *A note on the inevitability of maximum entropy*, Int. J. of Approximate Reasoning, 14, 183-223, (1990).

[Rescher, 1969] N. Rescher: *Many-Valued Logics*, Mc Craw-Hill, New York, (1969).

[Rödder, 2000] W. Rödder: *Conditional Logic and the Principle of Entropy*, Artificial Intelligence, 117, 83 – 106, (2000).

[Rödder and Kern-Isberner, 1997] W. Rödder and G. Kern-Isberner: *Léa Sombé und entropie-optimale Informationsverarbeitung mit der Expertensystem-Shell SPIRIT*, OR Spektrum, 19, 41 – 46, (1997).

[Rödder and Meyer, 1996] W. Rödder and C.-H. Meyer: *Coherent knowledge processing at maximum entropy by SPIRIT*, Proceedings 12th Conference on Uncertainty in Artificial Intelligence, E. Horvitz and F. Jensen (editors), Morgen Kaufmann, San Francisco Cal., 470 – 476, (1996).

[Shannon, 1948] C. E. Shannon: *A mathematical theory of communication*, Bell System Tech. J., 27, 379 – 423 (part I), 623 – 656 (part II), (1948).

[Shore and Johnson, 1980] J. E. Shore and R. W. Johnson: *Axiomatic Derivation of the Principle of Maximum Entropy and the Principle of Minimum Cross Entropy*, IEEE Trans. Information Theory, 26 (1), 26 – 37, (1980).

[Sombé, 1992] L. Sombé: *Schließen bei unsicherem Wissen in der Künstlichen Intelligenz*, Vieweg, Braunschweig, Wiesbaden, (1992).

Constraints as Data: A New Perspective on Inferring Probabilities

Manfred Jaeger
MPI Informatik
Stuhlsatzenhausweg 85, 66121 Saarbrücken, Germany
jaeger@mpi-sb.mpg.de

Abstract

We present a new approach to inferring a probability distribution which is incompletely specified by a number of linear constraints. We argue that the currently most popular approach of entropy maximization depends on a "constraints as knowledge" interpretation of the constraints, and that a different "constraints as data" perspective leads to a completely different type of inference procedures by statistical methods. With statistical methods some of the counterintuitive results of entropy maximization can be avoided, and inconsistent sets of constraints can be handled just like consistent ones. A particular statistical inference method is developed and shown to have a nice robustness property.

1 Introduction

Probabilistic representations of uncertainty usually consist of a single probability distribution over a large (but finite) domain of possible states $D = \{d_1, \ldots, d_n\}$. It is thus required to assign a probability value p_i to each state d_i. Usually, a direct, full assessment of all these values is very difficult or impossible. All one usually is able to obtain are partial descriptions of $\boldsymbol{p} = (p_1, \ldots, p_n)$ by constraints of e.g. the form $\boldsymbol{p}(A \mid B) \leq z, \boldsymbol{p}(A) + \boldsymbol{p}(B) \leq \boldsymbol{p}(C)$, or "$A$ and B are independent", where A, B, C are subsets of D. Such constraints can be derived by knowledge elicitation from an expert, by direct observations of the domain, or by any other information gathering process.

A set $c_1, \ldots, c_N$ of constraints defines the set $\Delta(c_1, \ldots, c_N)$ of probability measures on D that are consistent with the constraints. Very rarely will $\Delta(c_1, \ldots, c_N)$ consist of a single probability distribution. Instead, it will either contain more than one element, or be empty (i.e., the constraints are inconsistent). A fundamental problem in probabilistic reasoning then is to select from the admissible set $\Delta(c_1, \ldots, c_N)$ a distribution $\boldsymbol{p} =: \mathit{sel}(c_1, \ldots, c_N)$ as the best guess for the true distribution the constraints describe.

This *measure selection problem* is well studied in the literature, particularly for the case where the constraints are linear and consistent. It is almost unanimously suggested that in this case one should select the distribution with maximal entropy from $\Delta(c_1, \ldots, c_N)$ [Shore and Johnson, 1980; Lemmer and Barth, 1982; Jaynes, 1982; Cheeseman, 1983; Paris and Vencovská, 1990; Rödder and Meyer, 1996]. A more general class of constraints is considered by Drudzel and van der Gaag [1995] who then employ the center of mass selection rule (according to this rule one selects the center of mass of the admissible region).

In this paper we propose a new selection rule which is radically different from either maximum entropy or center of mass. It is motivated by the observation that in spite of the very compelling justifications it has been given [Shore and Johnson, 1980; Jaynes, 1982; Paris and Vencovská, 1990], maximum entropy selection has some rather counterintuitive properties. These are illustrated by the following examples.

Example 1.1 Overhearing two strangers talking at an airport, we hear the first one saying " ... Jones got at least 45% of the votes ... ", and the second replying " ... Smith didn't get any less than 5% either ... ". Before the two disappear in the crowd, we also hear them both agreeing on the fact that if anyone else had bothered to run for mayor, then neither Smith nor Jones would have had a chance of winning the election. Suppose, now, that we need to assess the probability $P(\mathit{Smith})$ of an arbitrary voter in the unnamed home town of the two strangers having voted for Smith. The information we have establishes a lower bound of 0.05 and an upper bound of 0.55 on $P(\mathit{Smith})$. Moreover, we have learned that the relevant underlying state space only consists of *Smith* and *Jones*. If we base our probability assessment on entropy maximization, then we will obtain $P(\mathit{Smith}) = 0.5$. Intuitively, this assessment appears to be overly optimistic from Smith's point of view.

Example 1.2 For the construction of a medical diagnosis system ten different experts are asked for bounds on the two crucial conditional probabilities $P_1 = P(\mathit{stylosis} \mid \mathit{polycarpia})$, and $P_2 = P(\mathit{xylopserosis} \mid \mathit{anameae})$. Assume that 0.41 and 0.51 are the greatest lower bound and smallest upper bound, respectively, mentioned by any expert for P_1. Having complete confidence in the experts, we will then take it as given that the true value for P_1 lies in the interval [0.41,0.51]. Let [0.49,0.61] be the correspondingly defined interval for P_2. Applying maximum entropy to find the best values for P_1 and P_2 for our expert system, we will determine $P_1 = P_2 = 0.5$. This appears somewhat counterintuitive because we have chosen the same value for both probabilities,

even though the information provided would seem to indicate a smaller value for P_1 than for P_2.

The reasons why the maximum entropy solution appears counterintuitive in the two examples are very similar. In the first example, an equal percentage of 50% of votes for both Smith and Jones seems implausible, because the constraints are highly asymmetric. Experience tells us that the disparity of the given lower bounds probably reflects a similar disparity of the actual values, which will rather be assumed to be approximately 90% for Jones and 10% for Smith. Such an assessment could be based on a natural explanation for how the constraints were generated in the first place: one might suspect, for instance, that the constraints report the partial count of 50% of the votes, among which 45% were found to be for Jones, and 5% for Smith. In the second example it appears unlikely that the experts would systematically state larger upper and lower bounds for P_2 than for P_1 if these two probabilities were really the same.

In both examples we have thus argued that the maximum entropy distribution is a counterintuitive solution of the selection problem, because the given constraints are unlikely to be observed when this is the true distribution. Underlying this argument is a view of constraints that is fundamentally different from the view which (implicitly) underlies the use of the maximum entropy principle: entropy maximization is predicated on the view that the given constraints are just a description of a state of knowledge: the knowledge that the true distribution is a member of the admissible region defined by the constraints. We call this the *constraints as knowledge* perspective. In our examples – and, we would claim, in most cases where we encounter the measure selection problem – the given constraints are not only a description of our knowledge, they also are the source of our knowledge. They thereby carry not only the face-value information consisting of a restriction of the admissible region; they also carry the meta-information consisting of the fact that we observed exactly these constraints. This meta-information is relevant for the solution of the measure selection problem as it allows us to reason about the likelihood of observing the given constraints for different true distributions. We call the view of constraints that tries to take into account this meta-information the *constraints as data* perspective: constraints are seen as randomly sampled pieces of information. The distribution of this constraint data (the *constraint distribution*) is in part determined by the true distribution on the domain (the *domain distribution*), which we want to determine. In other words, the domain distribution is a parameter of the constraint distribution. Our problem thus becomes a statistical one: to infer a parameter of a distribution from random samples drawn from that distribution.

All statistical methods rely in part on considerations of likelihood. The most direct way to use likelihood is by maximum likelihood inference: select the parameter that gives highest probability to the observed sample. The measure selection rule we develop in this paper is likelihood maximization for the observed constraints. The main problem we face in a formal development of this intuitive principle is that statistical methods usually require a specific model on how the distribution of observed data depends on the parameter of interest, i.e. the stipulation of some underlying parametric family. Our goal, however, is to define a general rule for measure selection that does not require any knowledge about the random mechanism that produces the constraints. Our approach towards solving this dilemma is that of robust statistics: we do propose a specific model for the random generation of constraints, but this model is chosen such that in the long run it will lead to correct inferences for a fairly wide class of constraint distributions.

The constraints as data perspective coupled with statistical approaches to measure selection permits us to handle inconsistent sets of constraints just like consistent ones. Our statistical model for the constraint observation only must allow for the observation of wrong constraints, i.e. constraints not satisfied by the true distribution (as an erroneous assessment given by an expert, the premature and incorrect report of an election result, etc.). Such a model then assigns nonzero likelihoods to inconsistent sets of constraints, and a maximum likelihood solution can be found just as for consistent constraint sets.

The idea of measure selection by likelihood maximization for the observed constraints was already expressed in [Jaeger, 1998], but no concrete formalization of the idea was developed. The view of constraints as data has also been taken in somewhat different form by Dickey [1980], who proposed a model in which partial specifications of a probability distribution P were treated as random variables with a distribution depending on P. A major difference between Dickey's and our work is that Dickey does not consider partial specifications by arbitrary linear constraints, but only by values for a fixed set of "aspects" of P. It is interesting to note that Dickey takes it for granted that in most cases the specified aspects will overdetermine the model, i.e. be inconsistent, whereas authors in artificial intelligence assume underdetermined models.

In this paper we can only give an overview of our maximum likelihood approach to measure selection. Goal of this paper is to convey the main ideas, and to provide some insight into the feasibility of their mathematical development. More technical details, including proofs of the theorems here stated, will be given in a full technical paper.

2 The Constraint Sample Space

To treat constraints as random samples we have to view them as elements of some sample space on which probability distributions can be defined. Throughout we assume that the constraints refer to a distribution on a domain of n elements. The set of all these distributions can be identified with

$$\Delta^n := \{(p_1, \ldots, p_n) \in \mathbb{R}^n \mid p_i \geq 0, \sum_{i=1}^{n} p_i = 1\}.$$

A linear constraint then has the general form

$$c: \quad x_1 p_1 + \ldots + x_n p_n \leq z \qquad (x_1, \ldots, x_n, z \in \mathbb{R}). \quad (1)$$

We could identify this constraint with its parameters $x_1, \ldots, x_n, z$, and thus take $\mathbb{R}^{n+1}$ as our sample space. However, this would mean to view two equivalent constraints like

$p_1 - 2p_2 \leq 0.2$ and $2p_1 - 4p_2 \leq 0.4$ as different sample points. As it does not seem sensible that our method should depend on such representational variants of constraints, we prefer to distinguish constraints only according to the subsets of distributions they define. This can be done by writing constraints in a normal form

$$s_1 p_1 + \ldots + s_n p_n \leq 0, \tag{2}$$

where $\boldsymbol{s} := (s_1, \ldots, s_n)$ is an element of the $n-1$-dimensional unit sphere

$$S^{n-1} = \{(s_1, \ldots, s_n) \mid \sum_i s_i^2 = 1\}.$$

As every linear constraint (1) can be transformed into a unique normal form (2), we henceforward identify linear constraints c with points $\boldsymbol{s} \in S^{n-1}$, and model randomly observed constraints by probability distributions on S^{n-1}.

In the binomial case ($n = 2$), a constraint (2) is a (nontrivial) lower bound on p_1 iff $s_1 < 0$ and $s_2 > 0$; it is a (nontrivial) upper bound iff $s_1 > 0$ and $s_2 < 0$. The following definition generalizes this classification of constraints.

Definition 2.1 A *sign vector* is any vector with components in $\{-1, 0, 1\}$. For $r \in \mathbb{R}$ we define $sign(r)$ as $-1, 0$ or 1, depending on whether $r < 0$, $r = 0$, or $r > 0$. The sign vector $sign(\boldsymbol{s})$ for $\boldsymbol{s} \in S^{n-1}$ is the vector $(sign(s_i))_{i=1,\ldots,n}$. Each sign-vector ζ of length n defines a *sector* S^ζ in S^{n-1}:

$$S^\zeta := \{\boldsymbol{s} \in S^{n-1} \mid sign(\boldsymbol{s}) = \zeta\}. \tag{3}$$

The intuition behind this definition is that sectors contain constraints of the same qualitative type. The classification of constraints according to sectors gives rise to the following coarser, three-way distinction: a constraint $\boldsymbol{s}$ is *vacuous* iff $sign(s_i) \neq\in \{-1, 0\}$ for all i (a vacuous constraint is satisfied by all $\boldsymbol{p} \in \Delta^n$); $\boldsymbol{s}$ is a *support constraint* iff $sign(s_i) \in \{0, 1\}$ for all i (a support constraint is satisfied by all $\boldsymbol{p} \in \Delta^n$ whose set of support is a subset of $\{i \mid sign(s_i) = 0\}$); $\boldsymbol{s}$ is *proper* iff $sign(s_i) = 1$ and $sign(s_j) = -1$ for some i, j (a proper constraint $\boldsymbol{s}$ divides the interior of Δ^n, i.e. there exist $\boldsymbol{p} \in int\,\Delta^n$ that satisfy $\boldsymbol{s}$, and $\boldsymbol{p}' \in int\,\Delta^n$ that do not satisfy $\boldsymbol{s}$).

Figure 1 illustrates constraints from different sectors. Shown in the figure is the polytope Δ^3 with its 3 vertices corresponding to domain distributions that assign unit mass to one of the states in D. Six different constraints from three different sectors are represented by the halfplanes of points satisfying the constraint. Halfplanes are shown by their boundary line, and a shading that indicates to which side of the boundary the halfplane extends.

For the rest of the paper we make two simplifying assumptions. *Assumption 1:* All constraints in the observed sample are proper. *Assumption 2:* The model $\boldsymbol{p} \in \Delta^n$ we want to determine lies in the interior $int\,\Delta^n$ of Δ^n. The two assumptions are somewhat connected. Non-proper constraints are essentially constraints on the set of support of $\boldsymbol{p}$. Thus, both assumptions will be satisfied if in an initial inference step we use all observed non-proper constraints to determine a set of support for our model, and then use the method we shall develop on the remaining proper constraints to determine $\boldsymbol{p}$ with

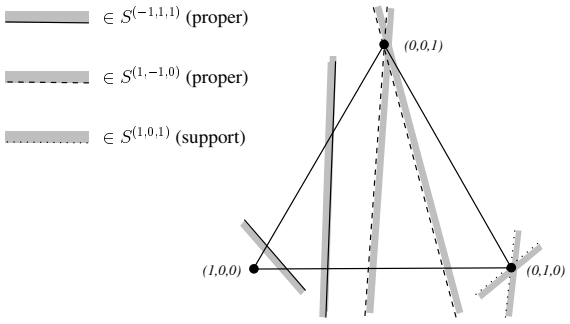


Figure 1: Constraints from different sectors

that set of support. With these assumptions, the measure selection problem consists of finding selection rules in the following sense.

Definition 2.2 A *selection rule* is any function that for every $n \in \mathbb{N}$ maps any tuple $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N \in (S^{n-1})^N$ ($N \in \mathbb{N}$) of proper constraints to a set $sel(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N) \subseteq int\,\Delta^n$.

This definition of a selection rule is not directly tied to constraints as data, and is very similar to Paris and Vencovská's [1990] notion of an *inference process*. It is very general in two respects: first, it is not required that $sel(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N)$ consists of a unique point. While it is obviously desirable that a selection rule yields unique solutions as often as possible, one needs to take the possibility into account that no principled statistical method can guarantee unique solutions in all cases. Second, it is not demanded that $sel(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N)$ be a subset of $\Delta(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N)$ (the distributions on D that actually satisfy the constraints). Such a demand, which is natural from the constraints as knowledge perspective, is not required from the constraints as data perspective. To see why, recall that in order to deal with inconsistent constraint sets (and also for greater realism) we should work with probabilistic models according to which it is possible to observe false constraints. This means that even for consistent constraint sets we must take the possibility into account that it contains false constraints, and that therefore the true domain distribution does not belong to $\Delta(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N)$.

3 Invariance and Equivariance

A selection rule in the sense of Definition 2.2 is a maximum likelihood selection rule, if it takes the following form: for every $N \in \mathbb{N}$, and for every $\boldsymbol{p} \in int\,\Delta^n$ a probability distribution $F_{\boldsymbol{p}}^N$ on $(S^{n-1})^N$ is defined, and for a sample $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N$ of N constraints we select the distributions in $int\,\Delta^n$ that maximize the likelihood of the sample:

$$sel_F(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N) := \underset{\boldsymbol{p}}{arg\,max}\, f_{\boldsymbol{p}}^N(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N), \tag{4}$$

where $f_{\boldsymbol{p}}^N$ is the density function of $F_{\boldsymbol{p}}^N$. Usually, constraints will be assumed to be independent, so that with $f_{\boldsymbol{p}} := f_{\boldsymbol{p}}^1$

$$f_{\boldsymbol{p}}^N(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N) = \prod_{i=1}^N f_{\boldsymbol{p}}(\boldsymbol{s}_i). \tag{5}$$

Whenever some information about the random process that generates the constraints is available, then one will choose in (4) a family $(F_{\boldsymbol{p}}^N)_{\boldsymbol{p}}$ that is a plausible model for this random process. The question we shall be concerned with, however, is what to do when no particular information about the generation of the constraints is available. Thus, we address the same inference problem as addressed by maximum entropy inference: input of the selection problem is a set of constraints, and nothing else.

The justification of the maximum entropy principle, in broad outline, takes the following form: because there is no information except the constraints, one should select the domain distribution $\boldsymbol{p}$ that encodes the least additional information beyond the face-value information provided by the constraints. Minimal information content, in turn, is realized by distributions $\boldsymbol{p}$ that, roughly speaking, maximize independencies and uniformity. Our approach to dealing with the lack of information is somewhat similar, only applied to the constraint distribution: because we have no information on the family $(F_{\boldsymbol{p}}^N)_{\boldsymbol{p}}$, we should assume the least specific structure of this family. In particular, we will assume that the constraints are independent, and that the family $(F_{\boldsymbol{p}})_{\boldsymbol{p}}$ is homogeneous in $\boldsymbol{p}$, in a sense that will be formalized by the notion of *G-invariance*, which is developed in this section.

We derive the concept of G-invariance from the semantic concept of a random constraint generating mechanism that works uniformly for all $\boldsymbol{p}$. Additional support for the G-invariance assumption on $(F_{\boldsymbol{p}})_{\boldsymbol{p}}$ will be given by the observation that maximum likelihood selection with respect to G-invariant families is *G-equivariant*, and that this can be understood as the formalization of the intuitive principle that a uniform shift applied to all constraints should induce a similar shift of the selected distributions (cf. Example 1.2).

To motivate the concept of a random constraint generating mechanism that works uniformly for all $\boldsymbol{p}$, reconsider Example 1.1, and the subsequently given explanation of how the constraints might have been generated from a partial count of 50% of the votes. If the true distribution is indeed $\hat{\boldsymbol{p}} = (\hat{p_1}, \hat{p_2}) = (.1, .9)$ $(= (P(\textit{Smith}), P(\textit{Jones})))$, then the observation of the constraints $.05 \leq p_1 \leq .55$ follow as a result of a sequence of chance events: the partial count of 50% of the votes becomes known, this partial count happens to be an accurate projection of the full count, and two strangers happen to mention these partial counts in their conversation. This sequence of chance events does not depend on the distribution $\hat{\boldsymbol{p}}$. If the true distribution was $\boldsymbol{p}^* = (.4, .6)$, then the same sequence of events would occur with the same likelihood, but now generate the constraints $.2 \leq p_1 \leq .7$.

Generalizing from this example, we obtain the (yet informal) notion of a random constraint generating process that works uniformly for all $\boldsymbol{p}$: the constraint generating mechanism will produce a constraint $\hat{\boldsymbol{s}}$ when the true domain distribution is $\hat{\boldsymbol{p}}$ with the same likelihood as it will produce a constraint $\boldsymbol{s}^*$ corresponding to $\hat{\boldsymbol{s}}$ when the true distribution is $\boldsymbol{p}^*$. To make this idea precise, we have to find a transformation g on constraints that maps every $\boldsymbol{s} \in S^{n-1}$ to a corresponding $g(\boldsymbol{s}) \in S^{n-1}$, such that an observation of $\boldsymbol{s}$ under the true distribution $\hat{\boldsymbol{p}}$ corresponds to an observation of $g(\boldsymbol{s})$ under $\boldsymbol{p}^*$. The correspondence expressed by g should preserve elementary qualitative properties of constraints. Two natural preservation conditions are:

Sector preservation: g maps every sector S^ζ bijectively onto itself.

Implication preservation: For all $k \in \mathbb{N}$, $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k$:

$$\bigcap_{i=1}^{k-1} \Delta(\boldsymbol{s}_i) \subseteq \Delta(\boldsymbol{s}_k) \Leftrightarrow \bigcap_{i=1}^{k-1} \Delta(g(\boldsymbol{s}_i)) \subseteq \Delta(g(\boldsymbol{s}_k)) \quad (6)$$

Sector preservation means that two corresponding constraints should be of the same qualitative type, as expressed by their membership in a sector. Implication preservation says that logical relationships between constraints should be preserved. Implication preservation is equivalent to the conjunction of two simpler conditions: $g(-\boldsymbol{s}) = -g(\boldsymbol{s})$ for all $\boldsymbol{s}$, and $\bigcap_{i=1}^{k-1} \Delta(\boldsymbol{s}_i) = \emptyset \Leftrightarrow \bigcap_{i=1}^{k-1} \Delta(g(\boldsymbol{s}_i)) = \emptyset$ (consistency preservation).

The following definition introduces a class of transformations that satisfy both properties.

Definition 3.1 Let $\boldsymbol{r} = (r_1, \ldots, r_n) \in (\mathbb{R}^+)^n$. The transformation $g_{\boldsymbol{r}} : S^{n-1} \to S^{n-1}$ is defined by

$$g_{\boldsymbol{r}}\ ((s_1, \ldots, s_n)) := \frac{(r_1 s_1, \ldots, r_n s_n)}{\|(r_1 s_1, \ldots, r_n s_n)\|}.$$

We write G_n for the set $\{g_{\boldsymbol{r}} \mid \boldsymbol{r} \in (\mathbb{R}^+)^n\}$.

It is obvious that transformations $g_{\boldsymbol{r}}$ satisfy sector preservation. They also satisfy a slightly strengthened version of implication preservation. For this, denote by $H(\boldsymbol{s}) \subseteq \mathbb{R}^n$ the set of all real solutions of (2), without the restriction to solutions $\boldsymbol{p} \in \Delta^n$ (so that $\Delta(\boldsymbol{s}) = H(\boldsymbol{s}) \cap \Delta^n$). In analogy to (6) we can then define *global implication preservation* of g by the condition

$$\bigcap_{i=1}^{k-1} H(\boldsymbol{s}_i) \subseteq H(\boldsymbol{s}_k) \Leftrightarrow \bigcap_{i=1}^{k-1} H(g(\boldsymbol{s}_i)) \subseteq H(g(\boldsymbol{s}_k)). \quad (7)$$

With condition (7) we look at constraints as defining sets of real numbers, not sets of probability distributions. In our context condition (6) seems to be the more pertinent one. We nevertheless here introduce the global version (7), because with this version we can prove the following representation theorem.

Theorem 3.2 Let $n \geq 3$, $g : S^{n-1} \to S^{n-1}$. g preserves sectors and is globally implication preserving iff $g \in G_n$.

The theorem does not hold for $n = 2$. The proof is by reduction to a classical representation result in projective geometry which characterizes mappings that preserve collinearity. We may conjecture that the theorem also holds when the condition of global implication preservation is replaced by implication preservation in our preferred sense (6). A proof of this modified theorem appears to be considerably harder, however.

In light of Theorem 3.2 we see the transformations $g_{\boldsymbol{r}} \in G_n$ as the adequate realization of the concept of correspondence of constraints. To relate this correspondence to different domain distributions, we define dual transformations on Δ^n.

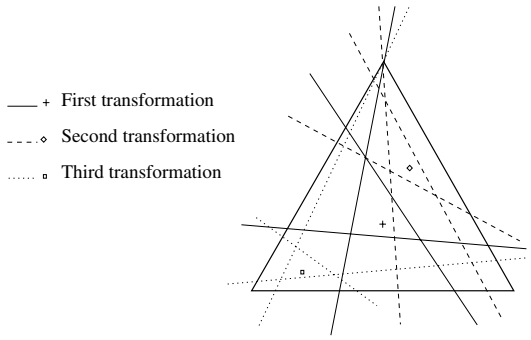


Figure 2: G-transformations and equivariant selection

Definition 3.3 Let $\boldsymbol{r} = (r_1, \ldots, r_n) \in (\mathbb{R}^+)^n$. The transformation $\bar{g}_{\boldsymbol{r}} : \Delta^n \to \Delta^n$ is defined by

$$\bar{g}_{\boldsymbol{r}}\ ((p_1, \ldots, p_n)) := \frac{(p_1/r_1, \ldots, p_n/r_n)}{\sum_{i=1}^n p_i/r_i}.$$

We write $\bar{G}_n$ for the set $\{\bar{g}_{\boldsymbol{r}} \mid \boldsymbol{r} \in (\mathbb{R}^+)^n\}$.

The mapping $\bar{g}_{\boldsymbol{r}}$ is dual to $g_{\boldsymbol{r}}$ in that it is the only transformation of Δ^n such that for all $\boldsymbol{p}$,$\boldsymbol{s}$:

$$\boldsymbol{p} \in \Delta(\boldsymbol{s}) \quad \Leftrightarrow \quad \bar{g}_{\boldsymbol{r}}(\boldsymbol{p}) \in \Delta(g_{\boldsymbol{r}}(\boldsymbol{s})). \tag{8}$$

Our initial intuition of corresponding observations of constraints now can be phrased as follows: the observation of constraint $\boldsymbol{s}$ under the true domain distribution $\boldsymbol{p}$ corresponds to the observation of constraint $g_{\boldsymbol{r}}(\boldsymbol{s})$ under the true domain distribution $\bar{g}_{\boldsymbol{r}}(\boldsymbol{p})$. Figure 2 shows three different transformations of a set of three constraints, and the dual transformations of one probability measure inside the admissible region of the constraints. Each of the three sets of constraints can be transformed into any of the other two sets by unique $g_{\boldsymbol{r}} \in G_n$. The dual transformations $\bar{g}_{\boldsymbol{r}}$ at the same time transform the indicated points in Δ^3 into each other.

With the transformations G_n and $\bar{G}_n$ we can now finally formalize the idea of a constraint generating mechanism that works uniformly for all $\boldsymbol{p}$:

Definition 3.4 Let $(F_{\boldsymbol{p}})_{\boldsymbol{p} \in int\, \Delta^n}$ be a family of distributions on S^{n-1}. The family is called *G-invariant* if for all $g_{\boldsymbol{r}} \in G$ and all $\boldsymbol{p} \in int\, \Delta^n$:

$$g_{\boldsymbol{r}}(F_{\boldsymbol{p}}) = F_{\bar{g}_{\boldsymbol{r}}(\boldsymbol{p})}. \tag{9}$$

When the distributions $F_{\boldsymbol{p}}$ are represented by densities $f_{\boldsymbol{p}}$ relative to a suitably chosen underlying measure on S^{n-1}, then (9) can be expressed by the condition

$$f_{\boldsymbol{p}}(\boldsymbol{s}) = f_{\bar{g}_{\boldsymbol{r}}(\boldsymbol{p})}(g_{\boldsymbol{r}}(\boldsymbol{s})). \tag{10}$$

By using such appropriate density functions, it is immediate that the maximum likelihood selection rule given by (4) and (5) becomes *G-equivariant* in the sense of the following definition.

Definition 3.5 A selection rule *sel* is called *G-equivariant* iff for samples $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N)$ of constraints, and every $g_{\boldsymbol{r}} \in G_n$

$$sel(g_{\boldsymbol{r}}(\boldsymbol{s}_1), \ldots, g_{\boldsymbol{r}}(\boldsymbol{s}_N)) = \bar{g}_{\boldsymbol{r}}(sel(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N)). \tag{11}$$

Note that the concept of G-equivariance does not, in turn, rely on maximum likelihood selection rules. Indeed, independently from the constraints as data interpretation, G-equivariance captures the idea that when the given constraints undergo a shift in one directions, then the selected distributions should undergo a similar shift. In Figure 2, a G-equivariant rule would have to select the distribution indicated by a cross given the solid constraints iff it selects the distribution indicated by a diamond given the dashed constraints iff it selects the distribution indicated by a box given the dotted constraints.

A second homogeneity assumption one will make about $(F_{\boldsymbol{p}})_{\boldsymbol{p}}$ in the absence of any information to the contrary is *permutation invariance*: if π is any permutation of $1, \ldots, n$, then for all $\boldsymbol{p}$

$$\pi(F_{\boldsymbol{p}}) = F_{\pi \boldsymbol{p}}. \tag{12}$$

Maximum likelihood selection with respect to a permutation invariant family $(F_{\boldsymbol{p}})_{\boldsymbol{p}}$ leads to a *permutation equivariant* selection rule:

$$sel(\pi \boldsymbol{s}_1, \ldots, \pi \boldsymbol{s}_N) = \pi(sel(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N)). \tag{13}$$

4 Robust Estimation

In the previous section we have argued that when no particular information about the constraint generating family $(F_{\boldsymbol{p}})_{\boldsymbol{p}}$ is given, then reasonable assumptions on this family are G- and permutation invariance. These assumptions alone are not nearly sufficient to identify a unique such family: if F is any distribution on S^{n-1} that satisfies $F(\pi \boldsymbol{s}) = F(\boldsymbol{s})$ for all $\boldsymbol{s} \in S^{n-1}$ and all permutations π, then F gives rise to a G- and permutation invariant family $(F_{\boldsymbol{p}})_{\boldsymbol{p}}$ by letting $F_{\boldsymbol{u}} := F$, where $\boldsymbol{u} = (1/n, \ldots, 1/n)$ is the uniform domain distribution, and $F_{\boldsymbol{p}} := g_{\boldsymbol{r}}(F_{\boldsymbol{u}})$, where $g_{\boldsymbol{r}} \in G$ is the transformation uniquely determined by $\boldsymbol{p} = \bar{g}_{\boldsymbol{r}}(\boldsymbol{u})$. Conversely, every G- and permutation invariant family $(F_{\boldsymbol{p}})_{\boldsymbol{p}}$ is uniquely determined by its member $F_{\boldsymbol{u}}$, which has to satisfy $F_{\boldsymbol{u}}(\pi \boldsymbol{s}) = F_{\boldsymbol{u}}(\boldsymbol{s})$.

In the following, we define a particular family $(L_{\boldsymbol{p}})_{\boldsymbol{p}}$ by way of defining $L_{\boldsymbol{u}}$. The motivation for this family comes out of the robustness of maximum likelihood selection with respect to this family (Theorem 4.2). $L_{\boldsymbol{u}}$ can be thought of as a mixture of multivariate Laplace distributions that are separately defined on each sector. The usual multivariate Laplace distribution on $\mathbb{R}^n$ has a density $f(\boldsymbol{x})$ that depends on the Euclidean distance between $\boldsymbol{x}$ and the mean $\boldsymbol{m}$ of the distribution. To define Laplace-like distributions on the sectors of S^{n-1}, we first introduce a suitable metric on sectors:

Definition 4.1 Let $\zeta \in \{-1, 0, 1\}^n$, $\boldsymbol{s}, \boldsymbol{s}' \in S^\zeta$. Define

$$d^\zeta(\boldsymbol{s}, \boldsymbol{s}') := \left(\sum_{i,j:\, \zeta_i \neq 0, \zeta_j \neq 0} Log^2\left(\frac{s_i' s_j}{s_j' s_i}\right) \right)^{1/2}. \tag{14}$$

A density $l_{\boldsymbol{u}}(\boldsymbol{s})$ now is defined as a function of the distance between $\boldsymbol{s}$ and a reference constraint $\boldsymbol{m}(\zeta) \in S^\zeta$, which can be thought of as the mean constraint in sector S^ζ.

In order for $L_{\boldsymbol{u}}$ to satisfy $L_{\boldsymbol{u}}(\pi \boldsymbol{s}) = L_{\boldsymbol{u}}(\boldsymbol{s})$ for all permutations π, the $\boldsymbol{m}(\zeta)$ have to be chosen such that $\boldsymbol{m}(\pi\zeta) =$

$\pi \boldsymbol{m}(\zeta)$ for all sign vectors ζ and permutations π. Apart from this condition, no restriction has to be imposed on the choice of the $\boldsymbol{m}(\zeta)$ in order to obtain our robustness result. We therefore only assume at this point that some $\boldsymbol{m}(\zeta) \in S^\zeta$ have been appropriately fixed, and define

$$l_{\boldsymbol{u}}(\boldsymbol{s}) := exp(-d^\zeta(\boldsymbol{s}, \boldsymbol{m}(\zeta))). \quad (\boldsymbol{s} \in S^\zeta) \qquad (15)$$

The function $l_{\boldsymbol{u}}(\boldsymbol{s})$ is the density of a probability distribution $L_{\boldsymbol{u}}$, which induces a G- and permutation invariant family $(L_{\boldsymbol{p}})_{\boldsymbol{p}}$. The maximum likelihood selection rule sel_L based on this family is distinguished by the following robustness property.

Theorem 4.2 Let $n \geq 3$. Let $(F_{\boldsymbol{p}})_{\boldsymbol{p}}$ be a G- and permutation-invariant family of probability distributions on proper constraints such that $F_{\boldsymbol{u}}(S^\zeta) > 0$ for all proper sectors S^ζ. Let $F_{\boldsymbol{p}}^\infty$ denote the distribution of an infinite sequence $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots$ of independent constraints drawn according to $F_{\boldsymbol{p}}$. Then

$$F_{\boldsymbol{p}}^\infty(lim_{N\to\infty} sel_L(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N) = \boldsymbol{p}) = 1. \qquad (16)$$

The theorem says that in the long run we will select with probability 1 the correct distribution $\boldsymbol{p}$ by using sel_L, even when the constraints are actually generated according to distributions $(F_{\boldsymbol{p}})_{\boldsymbol{p}}$. The conditions $n \geq 3$ and $F_{\boldsymbol{u}}(S^\zeta) > 0$ make sure that with probability 1 $sel_L(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_N)$ will be a unique point for all sufficiently large N. To obtain an analogous result for $n = 2$ a mild additional condition on $(F_{\boldsymbol{p}})_{\boldsymbol{p}}$ must be added. The proof of theorem 4.2 follows the proof of a general robustness result given as theorem 1 in [Huber, 1967].

Theorem 4.2 provides a good justification for using sel_L on "large" samples. It does not provide any guarantee that sel_L will show a sensible behavior on small samples. In particular, the behavior on small samples can be strongly affected by the special choice of the reference constraints $\boldsymbol{m}(\zeta)$. Thus, the definition of sel_L and Theorem 4.2 do not yet provide a full answer to the measure selection problem from the constraints as data perspective. To extend these first results towards a fully satisfactory solution, one will have to develop suitable criteria by which to judge the performance of a maximum likelihood selection rule on small samples, and to specialize or modify the definition of sel_L to obtain a selection rule that performs well according to these criteria (but retains the asymptotic behavior (16)).

5 Conclusion

We have seen that an interpretation of constraints as data, not as knowledge, leads to a completely new perspective on the measure selection problem. This perspective calls for statistical methods of parameter estimation as the tool for measure selection. The key problem we then face is that statistical methods call for a statistical model for the data generation, but that (according to the traditional problem statement that we deal with) no information about the appropriate statistical model is given. We have argued that in the absence of any such information G- and permutation invariance are natural homogeneity assumptions for the constraint distributions. $(L_{\boldsymbol{p}})_{\boldsymbol{p}}$ is a relatively simple G- and permutation invariant family of constraint distributions that leads to a robust maximum likelihood selection rule.

Future work will have two major directions: first, the definition of sel_L will be refined in order to obtain a sensible small sample behavior of the selection rule. Second, it will be explored to which degree the assumptions made in Theorem 4.2 on the constraint generating family $(F_{\boldsymbol{p}})_{\boldsymbol{p}}$ can be relaxed without loosing (16) for sel_L.

References

[Cheeseman, 1983] P. Cheeseman. A method of computing generalized Bayesian probability values for expert systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 198–202, 1983.

[Dickey, 1980] J. M. Dickey. Beliefs about beliefs, a theory of stochastic assessment of subjective probabilities. In J.M. Bernardo, M.H. DeGroot, D.V. Lindley, and A.F.M. Smith, editors, *Bayesian Statistics*, pages 471–487. Valencia, Spain: University Press, 1980.

[Druzdzel and van der Gaag, 1995] M. J. Druzdzel and L. C. van der Gaag. Elicitation of probabilities for belief networks: Combining qualitative and quantitative information. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, pages 141–148, Montreal, Quebec, Canada, 1995.

[Huber, 1967] P.J. Huber. The behavior of maximum likelihood estimates under nonstandard conditions. In *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability, Vol.1*, pages 221–233. University of California Press, 1967.

[Jaeger, 1998] M. Jaeger. Measure selection: Notions of rationality and representation independence. In *Proceedings of UAI-98*, 1998.

[Jaynes, 1982] E.T. Jaynes. On the rationale of maximum-entropy methods. *Proceedings of the IEEE*, 70(9):939–952, 1982.

[Lemmer and Barth, 1982] J. F. Lemmer and S. W. Barth. Efficient minimum information updating for Bayesian inferencing in expert systems. In *Proceedings of AAAI 82*, pages 424–427, 1982.

[Paris and Vencovská, 1990] J.B. Paris and A. Vencovská. A note on the inevitability of maximum entropy. *International Journal of Approximate Reasoning*, 4:183–223, 1990.

[Rödder and Meyer, 1996] W. Rödder and C.-H. Meyer. Coherent knowledge processing at maximum entropy by SPIRIT. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI–96)*, pages 470–476, Portland, Oregon, 1996.

[Shore and Johnson, 1980] J.E. Shore and R.W. Johnson. Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on Information Theory*, IT-26(1):26–37, 1980.