KRF

Knowledge Representation Framework Project

Department of Computer and Information Science, Linköping University,
and Unit for Scientific Information and Learning, KTH, Stockholm

Erik Sandewall

An Introduction to Decision Trees and Probabilistic Causal Networks

This series contains technical reports and tutorial texts from the project on
the Knowledge Representation Framework (KRF).

The present report, PM-krf-007, can persistently be accessed as follows:

Project Memo URL:	http://www.ida.liu.se/ext/caisor/pm-archive/krf/007/
AIP (Article Index Page):	http://aip.name/se/Sandewall.Erik.-/2009/002/
Date of manuscript:	2009-04-19

Related information can also be obtained through the following www sites:

KRFwebsite:	http://www.ida.liu.se/ext/krf/
AIP naming scheme:	http://aip.name/info/
The author:	http://www.ida.liu.se/~erisa/

1 Introduction

This report is a concise introduction to decision trees and causal networks, also known as Bayes nets. It differs from most other texts by using formal representations for these trees and networks whereas other texts usually only use diagrams. The formal representation can be used immediately for input to the Leonardo software platform. The report is intended both as a tutorial text in a course in artificial intelligence, and as an introduction to the Leonardo implementation of decision trees and causal networks.

The present section will address decision trees; causal networks will be addressed in Section 2.

1.1 Deterministic Interpretation of Decision Trees

Let `X` be a set of *terms* and `V` be a function from terms to a *range* for each term, where the range is a sequence or an implicitly ordered set of possible *values* . Furthermore let `v` (the *value function*) be a function from terms to specific values such that `v.x` is a member of `V.x` for each `x` in `X` . Common cases for a range is: a finite sequence of symbolic values, for example <`red, green, amber`> ; an action that may be performed by an autonomous agent; a botanical species; a real number representing a reward e.g. in monetary terms.

Furthermore let `∅` be a similarly ordered set of possible *outcomes* .

A *decision element* is a record of the form

```
[x? r1 r2 ... rn]
```

where `x` is a member of `X` and `V.x` is a sequence of length `n` so that there is a one-to-one correspondence between arguments in the record and elements in `V.x` . Each argument `ri` must be either a member of `∅` or another decision element. If all arguments are members of `∅` then the decision element is said to be *terminal* .

The *outcome* of a terminal decision element for a given value function `v` is that argument in the decision element that corresponds to the value of `v.x` .

Example. Let `(pedx-signal 12)` be a term representing the current state of the pedestrian crossing signal number 12 in a particular city scenario. This signal may show "green man" allowing pedestrians to cross, "red man" requesting them to stop, or "off" meaning neither red nor green is shown. Let `∅= {stop, walk, walk-if-clear}` , representing the possible actions that a pedestrian may take when seeing the signal. Then

```
V.(pedx-signal 12) = <red-man, green-man, off>
```

Furthermore, at a moment where `v.(pedx-signal 12) = green-man` , the outcome of the decision element

```
[(pedx-signal 12)? stop walk walk-if-clear]
```

is the entity `walk` .

In some particular situations we shall use decision elements that are records of the more general form

```
[x? r1 r2 ... rn :range < ... >]
```

The symbol `:range` and the sequence following it are not considered as arguments and are not affected by the above definition.

A *decision tree* is a twotuple `[∅, D]` where `∅` is a set of outcomes, like before, and `D` is a decision element (usually not a terminal one) where all the arguments in all the decision elements on all levels in `D` are either members of `∅` , or themselves decision elements.

The *outcome* of a decision tree for a given value function `v` is obtained by recursive application of the definition of the outcome of a terminal decision element. For a given decision element

```
[x? r1 r2 ... rn]
```

on some level, if the element `ri` that corresponds to `v.x` is another decision element, then the outcome of that element is also the outcome of the given decision element.

A decision tree with a finite set of outcomes is called a *classification tree* whereas a decision tree whose set of outcomes is a continuous set of values is called a *regression tree* .

Decision trees are widely used in operations research, and the reader is recommended to consult the following wikipedia pages for additional information:

```
http://en.wikipedia.org/wiki/Decision_tree
http://en.wikipedia.org/wiki/Decision_tree_learning
```

Some of the important techniques in the context of decision trees are *reorganization of decision trees* and *acquisition of decision trees* from empirical information. They are also important in artificial intelligence systems, in particular in the contexts of data interpretation, decision-making and learning.

1.2 Probabilistic Interpretation of Decision Trees

Probabilistic interpretation of decision trees differs from the above in the sense that the value function `v` does not produce a single outcome for each term, but instead a probability distribution over the term's range. This is appropriate for representing situations with incomplete information.

For simplicity we restrict the presentation to the case of discrete ranges, in which case the value of `v(x)` can always be written as a sequence of probabilities whose sum is 1, and which correspond one-by-one to the elements in the range of `x` represented as a sequence.

Normally there is a mixed situation so that some terms have probabilistic values and some are deterministic. Then it is natural to identify the case where `v.x` is a member of `V.x` with the case where `v.x` is a probability vector where one element is 1 and the others are 0. The actually written-out function `v` can then mix probabilistic and deterministic values, but for the purpose of defining the outcome one can assume that all values of `v` are probabilistic.

The outcome of a decision tree for a probabilistic value function `v` is a probability distribution over $\emptyset$, that is, a function that assigns a probability to each member of $\emptyset$ such that the sum of the probabilities is 1. It is defined recursively as follows, assuming that $\emptyset$ is

```
<o1 o2 ... ok>
```

The outcome of a terminal decision element

```
[x? r1 r2 ... rn]
```

when `v.x = (p1 p2 ... pn)` is a probability distribution `(q1 q2 ... qk)` over the elements of $\emptyset$ where each `qj` is the sum of those `pi` for which `ri = oj` .

This is straightforward. It is then also straightforward to see the generalization to non-terminal decision elements: the outcome the decision element

```
[x? r1 r2 ... rn]
```

when `v.x = (p1 p2 ... pn)` is obtained by first calculating the outcome of each of the elements `ri` which will be a probability distribution

```
(ui1 ui2 ... uik)
```

over the elements of $\emptyset$. The outcome of the given record is then a probability distribution `(q1 q2 ... qk)` where each `qj` is the sum of `pi * uij` where `i` ranges from 1 to `n` . This is essentially a matrix-vector multiplication.

In matrix algebra terms this can be expressed somewhat more concisely as follows. The outcome is defined both for a term and for a decision element; in both cases it is a vector of length `k` (i.e. the size of $\emptyset$) representing a probability distribution. The outcome of a term `xk` in $\emptyset$ is a unit vector with 1 in its k'th element and 0 in the others. The outcome of a decision element

```
[x? r1 r2 ... rn]
```

is formed by first forming a $k \times n$ matrix U where the i'th column equals the outcome of `ri` , and then forming the product of U and the vector `v.x` .

We have made the distinction between deterministic and probabilistic *interpretations* of a decision tree, but the tree itself has the same structure in both cases. Some presentations, e.g. the wikipedia pages for decision trees actually turn this into a distinction in the trees themselves, by using square items in the graphical representation of a decision tree to represent "decision nodes" (deterministic) and circular items to represent "chance nodes" (probabilistic). This is useful from a practical point of view and for readability of the representation, but it has no formal significance.

1.3 Terminal Element Probability Distributions

[c? blue white]] [b? [c? white red] [c? green blue]]]

If it is given that `a` has the value `true` and the other two terms have the value `false` then the outcome of this decision tree is `white`. If some of the terms have probabilities assigned (probability for `true` or `false`, in this case) then the outcome from this decision tree will be a probability

distribution over the colors. The use of *terminal element probabilities* is a generalization where the decision-tree itself assigns probabilities even if all the terms being used assign specific values. The following is a modification of the example using this generalization:

```
[txt
   [a? [b? [c? red green]
           [c? blue <0.12  0.02  0.09  0.77>] ]
       [b? [c? white red]
           [c? green blue] ]]
```

In order to interpret this tree one must also have access to the order defined for the outcome set ∅. Suppose that it is

```
 <red green blue white>
```

Then the outcome of the tree, still given that `a` has the value `true` and the other two terms have the value `false` will be the probability distribution where `red` occurs with probability `0.12`, `green` with probability `0.02`, and so forth. This generalization will be used in Section 2 for the topic of causal nets.

1.4 Partial Interpretation of Decision Trees

We have described how decision trees can be interpreted when values have been set for all the terms, either as a specific value, or as a probability distribution. One important generalization occurs if values are entirely missing for a few of the terms. In this case the result of the interpretation is not a specific value or probability distribution, but a *set of alternative* values or distributions, namely, one for each possible combination of values for the unassigned terms. This is called *partial interpretation*.

Consider again the decision tree used above:

```
[a? [b? [c? red green]
        [c? blue white] ]
    [b? [c? white red]
        [c? green blue] ]]
```

If it is given that `a` has the value `true` and the other two terms have the value `false` then the outcome of this decision tree is `white`. However, if the value of `b` is not known then one must consider two cases: `b` is `true`, and `b` is `false`. In the former case the outcome is `green`, in the second case it is `white`. If several terms are unassigned then one obtains a case for every combination of their values.

For such situations it is natural to consider the outcome, in a generalized sense, as a simplified decision tree only using the unassigned terms. In our example this generalized outcome would be the following simple decision tree:

```
[b? green white]
```

Partial interpretation of decision trees can be used when one wishes to analyze the consequences of several alternative courses of action. The alternatives are characterized as one or a few unassigned terms, the step-by-step analysis of the entire situation is represented as a decision tree, partial

interpretation is done for this tree, and one obtains a summary of the consequences for each of the alternatives being considered.

The concept of partial interpretation originates from the theory of programming languages, where one may partially interpret a function definition or a procedure where some but not all the arguments are known. The result is then a simplified function or procedure.

1.5 Hierarchical Decision Trees

The following is an example of a hierarchical decision tree:

```
[?[a? [b? [c? red blue]
          [c? blue white] ]
      [b? [c? white red]
          [c? red blue] ]]
 rose bluebell waterlily
 :range <red blue white>]
```

It is similar to the following decision tree:

```
[color? rose bluebell waterlily]
```

except for two things: the line beginning with `:range` , and the fact that there is a composite expression instead of the single term `color?`. In fact, this composite expression is itself a decision tree that is embedded inside the main one. We write the question-mark before the subordinate decision-tree rather than after it since this makes the expression easier to read.

This structure is called hierarchical because it is intended as follows: given assignments of values to `a`, `b` and `c`, first obtain the outcome of the sub-decision-tree that uses these terms. The possible outcomes are `red`, `blue` and `white`, depending on the values of `a`, `b` and `c`. When that outcome has been obtained, use it in the top-level decision tree for obtaining the choice between `rose`, `bluebell` and `waterlily`.

Notice that the subordinate decision tree does not need to have the same outcome set as the main tree; in fact there is no reason why it should. On the other hand it is necessary to have a specification of the ordering of the possible outcomes of the sub-tree in order to interpret the top-level tree, and this is the reason for including the `:range` parameter which uses a facility in the Leonardo notation that is being used here.

A hierarchical decision tree can always be rewritten as an ordinary, flat one, but often at the expense of obtaining a much larger tree. Consider the following modification of the example:

```
[?[a? [b? [c? red blue]
          [c? blue white] ]
      [b? [c? white red]
          [c? red blue] ]]
 [choice? red-rose poppy pelargonia]
 [choice? bluebell forget-me-not violet]
 [choice? waterlily lily-of-the-valley white-rose]
 :range <red blue white>]
```

When this decision tree is interpreted, there is first a decision in the sub-decision-tree for the choice of color, and then a second decision according to the term `choice` between several available flowers having the color in question. In order to flatten this tree one has to replace the entities for the colors (`red`, `blue`, etc) with the appropriate subexpression beginning with `choice?`. The result will be that there are eight occurrences of `choice?` -expressions, instead of three in the original tree. The use of hierarchical decision trees is therefore a way of modularizing the tree.

We shall see an additional use of hierarchical decision trees in the next main section, which addresses causal nets.

1.6 Modes of Use

There are many examples of simple, deterministic decision trees in ordinary life. Large and complex decision trees are used in several disciplines and they have been extensively studied both in operations research and in artificial intelligence.

A well-known, practical use of decision trees is for debugging or error identification in machines for personal use, such as automobiles or washing-machines. They are usually represented in graphical form as flowcharts where each decision element is represented as a node containing a question, corresponding to a term in the notation used here, and arrows from one node to the next indicate how to proceed depending on the answer to the question.

A related use is for error identification "wizards" in many computer programs or computer-attached devices, where the decision tree is represented in software and the "wizard" asks the user successive questions as it interprets the decision tree.

A classical use of decision trees has been for schemata for identifying the genus and species of plants, as published in botanical books. The terms in the decision tree refer to various physical characteristics of the plant, and in particular its reproduction-related parts, and the tree leads to the identification of the species provided that the user is successful. Decision trees of this kind were introduced in the pioneering work *Systema Naturae* by Carolus Linnaeus.

Professional use of decision trees often make use of probabilistic and/or partial interpretation. Their use for evaluation of alternative courses of action has already been described.

A particularly important case arises in probabilistic interpretation of decision trees where the outcomes are numerical quantities, typically representing a profit or loss in a business enterprise. Since the interpretation is probabilistic, one obtains a set of possible, numerical outcomes, each of them with an associated probability. The *expected outcome* of the decision tree is then, as one can expect, the sum of the alternative outcomes, weighted with their respective probabilities. In combination with partial interpretation as already described, one obtains one expected outcome for each of the alternatives being produced, which can then serve as the basis for making a decision.

With respect to artificial intelligence, there are two major applications of decision trees, namely, for perception and for decision-making. In the case of perception the framework of use may be as follows: incoming sensory data are first analyzed through processes that reduce them to a limited number of *features*. Then these features are fed into a decision tree in order to obtain a classification of the object being seen or otherwise perceived.

The use of decision trees for decision-making in autonomous agents is similar to their use in operations research and in commercial contexts as described above. For example, the main computational cycle of the SOAR architecture [1] contains, among others, one step whose role is to generate a number of alternative courses of action, and a next step for making a choice between the suggested alternatives. A decision tree is a possible and natural device in that step.

It is not practical to engineer large decision trees by hand, even in the deterministic case, and it is entirely unrealistic to identify probability settings manually for their probabilistic interpretation. Automatic acquisition of these structures and quantities is therefore an important topic in the area of *machine learning*.

1.7 Implementation and Examples

A simple implementation of decision-trees and their interpretation is available for the Leonardo system and can be used in the `Leordo` distribution [2] . The decision-tre module can also be downloaded from the `Leordo` webpage. It uses the same notation as has been used here, except that the `V` and `v` operators are done differently.

The following is a brief summary of how the implementation works. Consider first the following excerpt from the demonstration example.

```
-----------------------------------------------------------
-- term-1

[: type decitree-term]
[: has-range <a b c>]
-----------------------------------------------------------
-- term-2

[: type decitree-term]
[: has-range <x y>]
-----------------------------------------------------------
-- termas-1

[: type term-assignment]
[: det-mapping {
    [: term-1 b]
    [: term-2 x] }]
[: prob-mapping {
    [: term-1 <0.1 0.8 0.1>]
    [: term-2 <0.2 0.8>] }]
```

[1] See e.g. our separate tutorial article about autonomous agents.

[2] `http://www.ida.liu.se/ext/leordo/`

```
-----------------------------------------------------------
-- decitree-1

[: type decision-tree]
[: has-outcomes <H J K>]
[: has-tree
       [term-1? :range <a b c> ^
           [term-2? H K]
           [term-2? H J]
           [term-2? J K] ]]
-----------------------------------------------------------
```

Three types are used: `decitree-term` for terms that are used in decision trees, `term-assignment` for entities that name an assignment of values to terms, and `decision-tree` for entities that name a decision tree. Entities of the second kind correspond to a `v` function in the present text.

The use of the `has-range` attribute is evident. The use of the ordered set $\emptyset$ in the present text is replaced by the use of the `has-outcomes` attribute of each decision-tree identifier. Each term assignment entity can have both a deterministic mapping and a probabilisitic mapping from decision-tree terms to their values or distributions, as the example shows. Probabilistic mappings may contain deterministic assignments when appropriate, but not vice versa. For example, one could have

```
[: prob-mapping {
    [: term-1 <0.1 0.8 0.1>]
    [: term-2 x] }]
```

Two commands have been implemented for the interpretation of decision trees in this notation, called `dede` for deterministic interpretation and `prode` for the probabilistic one. They are invoked like in

```
dede decitree-1 termas-1
```

and similarly for `prode`, the difference being that `dede` uses the `det-mapping` attribute of its second argument, and `prode` uses its `prob-mapping` attribute.

The use of the `range` parameter in `decitree-1` is only informative since the term `term-1` already has an attribute specifying its range, but it may facilitate reading the definition to have it there: it specifies that the expressions in the following lines apply for `a`, `b` and `c` in that order.

Terminal-entity probability distributions are supported, but not partial interpretation or hierarchical decision trees.

1.8 Integration in Larger Systems

A decision tree in an artificial intelligence application will typically be used repeatedly with different assignments of values for the terms, that is, for different choices of the `v` function. This is true both in the case of interpreting or classifying incoming data, and in the case of making a decision given some alternative choices. For the formal presentation in the present text it is sufficient to say that one has different choices of the function `v`, and our tutorial implementation uses multiple instances of the type `term-assignment`

for the same purpose.

When this technique is to be integrated in a larger system one may need different conventions in order to adapt to the representation of the terms – and their values – that is already used in the larger system. In general, a decision tree is a device for inferring and aggregating information about *one particular object* each time it is interpreted, and repeated use of the decision tree is worthwhile if one addresses a new object, or if some of the properties of the object may have changed since the previous use.

For the purpose of system integration one must therefore start from the actual representation of the objects being considered and their associated information, and adjust the implementation of decision trees so that it combines without problems with the main system.

1.9 Integration in a Leonardo-based Application

In the particular case of systems that are built on the Leonardo platform, objects for the purpose of decision-tree interpretation are likely to be Leonardo entities, and the terms to be considered may simply be attributes of those entities. However there are also other possibilities:

- One may wish to obtain an attribute of an attribute of the given entity
- One may wish to pre-process an attribute of the given entity with some computational transformation
- Our household example (see our main lecture notes in knowledge representation) introduced the use of `features` for representing properties of an object that change over time. A term in a decision tree may therefore need to be defined as the current value of a feature of the entity at hand

A general way of accomodating these possibilities as well as several others, in the Leonardo context, is to generalize the decision trees so that the terms in them (the symbols ending with a question-mark) are allowed to be arbitrary Leonardo terms where the variable `.obj` is defined and is bound to the entity that is subject to interpretation. In this way one will be able to write e.g. the following decision tree when the object `.obj` represents a particular door that an autonomous agent wishes to pass:

```
[(get .obj requires-access-card)?
    [(I-have (card-for .obj))?
        [soact [insert-card .obj][pass-door .obj]]
        [fail] ]
    [(is-locked .obj)?
        [(I-have (key-for .obj))?
             [soact [unlock-door .obj][pass-door .obj]]
             [fail] ]
        [pass-door .obj] ]]
```

For a given value of `.obj`, for example `door-19`, this decision tree returns either the fail message `[fail]` or one of the following proposed actions or action sequences:

```
[soact [insert-card door-19][pass-door door-19]]
[soact [unlock-door door-19][pass-door door-19]]
[pass-door door-19]
```

The operator `soact` stands for "sequence of actions." One will assume that the entity `door-19` has a truth-valued attribute for `requires-access-card` since this is a fairly permanent property of the door. One also assumes that `card-for` and `key-for` are functions whose values are reifications [3] of the properties of being an access card for the door in question and being a key for the same door, and finally one assumes that the function `I-have` takes such a reification as its argument and produces a truth-valued feature whose value at each point in time is either `true` or `false` depending one whether the agent currently possesses an object with the specified property.

2 Causal Nets

A *causal net* consists of a set of *independent terms*, a set of *dependent terms*, and the assignment of a *dependency expression* to each dependent term. Like in the case of decision trees, there must also be a mapping `V` from terms (both dependent and independent ones) to a range for each term. The dependency expression for a dependent term specifies a set of other terms that the given term depends on, called its *dependees*, and a way of calculating the value of the given term in terms of the values of its dependees. The graph that is defined by the links from dependent terms to their dependees must not contain any cycles.

2.1 Propositional Probabilistic Causal Nets

The case of deterministic causal nets is the simplest one, but we shall proceed directly to the case of probabilistic nets. On the other hand we begin by making the assumption that all the ranges consist of merely two values, `true` and `false`, which simplifies the treatment. The generalization to ranges of more than two values follows in the next subsection.

Probabilistic causal nets are often called Bayes nets since some of the important and nontrivial operations on them are based on *Bayes' rule* in probability theory. We prefer to use the term "causal nets" since it expresses what the nets are intended to represent, and since not all of the operations on these nets are based on Bayes' rule.

Consider now the following example, as shown in Figure 1. We have three independent terms, `b`, `e`, and `f`, and three dependent terms, `d`, `c`, and `a`. The value of `d` depends on the values of `e` and `f` in a way that is defined by the following dependency expression for `d`:

```
[e? [f? <0.4 0.6><0.7 0.3>]
    [f? <0.1 0.9><0.3 0.7>]]
```

We notice that the dependency expression is a decision tree with terminal-entity probability distributions. It is to be read as follows. If the value of both `e` and `f` is `true` then the probability is 0.4 that the value of `d` is `true` and, consequently, the probability is 0.6 that the value of `d` is `false`. If `e` is

[3] See our KR lecture notes for an explanation of this concept.

`true` and `f` is `false` then the probability is 0.7 that the value of `d` is `true`, and so on.

The example specifies both a probability p and the complementary probability $1-p$ which is a bit tedious, and we introduce the abbreviated notation where e.g. `40%` represents the tuple <`0.4 0.6`>. The same example can then be written as follows:

```
[e? [f? 40% 70%]
    [f? 10% 30%]]
```

Notice that this is a nonstandard notation, and that we will not write e.g. `40%` to signify `0.4`, only to signify <`0.4 0.6`>.

If a propositional probabilistic causal net is given, let `v` be an assignment of probabilities for the independent terms of that net. Then the *probability for* a dependent term in the net is defined as the outcome of its dependency expression using the outcomes of its dependees, as defined in the previous section. This is well-defined since the dependency graph is assumed not to have any cycles.

2.2 An Example

The following example is originally due to Eugene Charniak, in his article in the *AI Magazine* in 1991. It has also been used by C.R. Dyer in his on-line lecture note on Reasoning under Uncertainty [4] and I quote from Dyer's note:

"Consider the problem domain in which when I go home I want to know if someone in my family is home before I go in. Let's say I know the following information: (1) when my wife leaves the house, she often (but not always) turns on the outside light. (She also sometimes turns the light on when she's expecting a guest). (2) When nobody is home, the dog is often left outside. (3) If the dog has bowel-troubles, it is also often left outside. (4) If the dog is outside, I will probably hear it barking (though it might not bark, or I might hear a different dog barking and think it's my dog)."

This information is represented using the following terms: `noone-home`, `lights-are-on`, `dog-outside`, `dog-sick`, and `I-hear-dog`. The last of them shall be understood as "I hear a dog and I think it's mine." Of these terms, `noone-home` and `dog-sick` are independent terms since we do not have any information about how they could depend on any of the other terms, in the sense of being caused by any of them. The other three terms are dependent ones and we posit the following dependency expressions for them:

```
lights-are-on   [noone-home? 70% 20%]

dog-outside     [noone-home? [dog-sick? 80% 70%]
                             [dog-sick? 70% 30%]]

I-hear-dog      [dog-outside? 80% 10%]
```

For example, if someone is home and the dog is not sick then the probability is stated as 0.3 that the dog will be outside and 0.7 that it will not.

[4] `http://pages.cs.wisc.edu/~dyer/cs540/notes/uncertainty.html`

Consider now a function `v` that assigns probabilities to the independent terms, for example `v.noone-home = 0.6` and `v.dog-sick = 0.05`. It is straightforward to calculate the probabilities for the dependent terms.

Practical examples will of course be very much larger. Anyway, one natural way of using this representation is if both a causal network and an a priori definition of the `v` function are given, but in a particular situation one has some additional information that overrides some parts of the `v` function. If my wife has said that she would maybe go and see a movie with a friend this evening, but she was not sure if this friend was able to join, then I would adjust the value of `v.noone-home` and recalculate the dependent probabilities. This is an example of *direct evaluation* of a probabilistic causal net.

Notice by the way that there is no such thing as the "probability" or the "outcome" of the entire causal net. The net makes it possible to calculate the probability for each dependent term in the net. Sometimes maybe one of the dependent terms is the one that one is particularly interested in, so that the probability for it is the useful output, but this is merely a question of how the causal network is used.

2.3 Inverse Evaluation in a Causal Net

Although direct evaluation of probabilistic causal nets is sometimes useful, a much more important application occurs if values are known for some of the *dependent* terms and one wishes to infer revised values for some of the independent ones. In the example above, if I am approaching home and I hear the dog bark, how should I then revise my a priori probabilities for `noone-home` and `dog-sick` in view of this additional information? This is in line with the quoted problem statement above.

This is an example of an *inverse problem*: the direct evaluation of a probabilistic causal net is simple, but if some of the outputs of the direct problem are instead going to be inputs, then it is more complicated. This is where Bayes' rule from probability theory comes in.

At this point, please proceed by reading the on-line articles by Dyer that was mentioned above.

2.4 Non-propositional Probabilistic Causal Nets

We have first addressed the case where the range of each term is the set of the two truth-values `true` and `false`. We have mentioned the "probability for" a particular term, meaning the probability that the value of the term is `true`.

In many cases it is desirable to use a larger range for some of the terms. This does not change anything in essence, but it leads to somewhat larger dependency expressions. Let us first rewrite the above example without the percent notation so as to represent the two values of each range explicitly:

```
lights-are-on   [noone-home? <0.7 0.3><0.2 0.8>]

dog-outside     [noone-home? [dog-sick? <0.8 0.2><0.7 0.3>]
                             [dog-sick? <0.7 0.3><0.3 0.7>]]
```

```
I-hear-dog      [dog-outside? <0.8 0.2><0.1 0.9>]
```

To read this out, one notices that if `noone-home` is `true` then the probability is 0.7 that `lights-are-on` is `true` and it is 0.3 that `lights-are-on` is `false`. Suppose now that the term `noone-home` is generalized to the term `someone-home` with three possible values, namely `none`, `wife`, and `someone-else`, but the value of `lights-are-on` is still just `true` or `false`. Then the first dependency rule might become:

```
lights-are-on   [someone-home? <0.2 0.8><0.7 0.3><0.8 0.2>]
```

Notice that there are three sublists after the term `someone-home` since its range has three elements, and that each of those three sublists has two elements since the range of `lights-are-on` has two elements. If there are more than two values for `lights-are-on`, for example for indoors and outdoors lights, then the number of elements in the sublists would have to change accordingly.

2.5 The Implementation

The implementation of decision trees for `Leordo` which was described above contains also a facility for direct evaluation of probabilistic causal nets. The following is the representation of Charniak's example for this implementation:

```
----------------------------------------
-- noone-home

[: type causal-net-node]
[: has-range <true false>]
----------------------------------------
-- lights-are-on

[: type causal-net-node]
[: has-range <true false>]
[: dependency-expression [noone-home? 75% 20%]]
----------------------------------------
-- dog-outside

[: type causal-net-node]
[: has-range <true false>]
[: dependency-expression
        [noone-home? [dog-sick? 80% 70%]
                     [dog-sick? 70% 30%] ]]
----------------------------------------
-- dog-sick

[: type causal-net-node]
[: has-range <true false>]
----------------------------------------
-- I-hear-dog

[: type causal-net-node]
[: has-range <true false>]
```

```
[: dependency-expression [dog-outside? 80% 10%]]
-------------------------------------------------------
-- termas-3

[: type term-assignment]
[: prob-mapping {
    [: noone-home 60%]
    [: dog-sick   5% ] }]
-------------------------------------------------------
```

Most of this is self-explanatory. The type `term-assignment` is used like before, and there is a new type called `causal-net-node` that is used for both dependent and independent terms in a causal net. They are distinguished since the former have a value for the attribute `dependency-expression` which shall have the same structure as the value for `has-tree` that was used above. The percent notation can be used with the same meaning as has been used in the present report.

The command `caev` is used for evaluation of a node in a causal net, like in for example

```
caev I-hear-dog termas-3
```

This operation will interpret the `dependency-expression` of its first argument. If, in the course of that interpretation, it encounters an entity that is not assigned a value by the term assignment at hand (namely, the one that was given by the second argument) but which does have a value for `dependency-expression` then it will call itself recursively for that expression, and proceed from there on the return. The implementation therefore does not preserve and reuse the values from the interpretation of a particular `dependency-expression`; it recalculates the outcome every time it is needed.

Notice, by the way, that what happens during such an evaluation process is essentially a case of hierarchical decision trees, except that the subordinate trees are not physically embedded in their superiors; they are merely mentioned by name there.

3 Conclusion

If the reader is a participant in my present AI course, he is encouraged to return to the course webpage at this point and to continue with the other links for the general area of uncertainty in reasoning systems.