Theory of Evidence (DRAFT)

Nicolas Burrus and David Lesage

Technical Report n^o0307 - 07/07/03

The theory of evidence, also called Dempster-Shafer theory or belief functions theory, has been introduced by Glenn Shafer in 1976 as a new approach for representing uncertainty. Nowadays, this formalism is considered as one of the most interesting alternatives to Bayesian networks and fuzzy sets. This report makes an overview of both theoretical and implementation aspects of this theory. After a short survey of the historical motivations for this theory, we present its interesting properties through the Transferable Belief Model formalism. From a more practical point of view, we propose a review of the existing optimizations for facing the #P complexity of Dempster-Shafer computations. This report introduces a new, promising concept to compute repeated fusions: the delayed mass valuation. Finally, we present `eVidenZ`, our general-purpose C++ library for designing efficient Dempster-Shafer engines.

Keywords
`eVidenZ` engine, Dempster-Shafer theory, evidence theory, belief functions theory, Decision under uncertainty.

Laboratoire de Recherche et Développement de l'Epita
14-16, rue Voltaire – F-94276 Le Kremlin-Bicêtre cedex – France
Tél. +33 1 53 14 59 47 – Fax. +33 1 53 14 59 22
`lrde@epita.fr` – `http://www.lrde.epita.fr`

General introduction

Today's research and applications in the field of quantitative reasoning and decision under uncertainty are dominated by Bayesian networks and their variants. This is somewhat surprising, since many situations involving uncertainty can not be represented properly within the classical probability framework. There is, for example, no adequate way of representing total ignorance.

One of the most promising alternatives is the theory of evidence, also known as Dempster-Shafer theory or belief functions theory. This theory has been introduced in the late seventies by Glenn Shafer as a way of representing epistemic or uncertain knowledge, starting from a sequence of seminal works of Arthur Dempster, Shafer's advisor.

This report is first a bibliographical survey of the foundations aspects of Dempster-Shafer theory. We also introduce `eVidenZ`, our general-purpose, C++ Dempster-Shafer engine. Thus, this report has a second, practical-oriented aspect. It consigns our analysis, experiments and propositions and makes a survey of the existing implementation efforts.

This paper tries actually to answer the questions we faced while discovering this formalism and implementing our reasoning engine, `eVidenZ`.

Issues concerning Dempster-Shafer theory

Why are Dempster-Shafer bases so controversial? Dempster-Shafer theoretical bases have raised passionate discussions among researchers. Several approaches exist and lead to several justifications that still have some controversial aspects. Lots of researchers present Dempster-Shafer theory as a generalization of the probability framework. Some opponents argue that towards probability fundamentals, Dempster-Shafer theory is not consistent.

What is Dempster-Shafer purpose and applicability? In the field of reasoning under uncertainty, some frameworks, like bayesian networks or fuzzy sets, are very popular and commonly applied to a large amount of problems. At the contrary, Dempster-Shafer possibilities are rarely applied to concrete problems. The capabilities of this formalism are often misunderstood so that it is generally hard to define its real field of applicability. In the meantime, classical methods like bayesian networks are often used for applications where Dempster-Shafer theory would be more relevant.

Are Dempster-Shafer computations practically feasible? The truly remarkable representation power of this theory should not hide the practical difficulties. Indeed, computation time and memory load are critical problems, since the core mechanism, Shafer's fusion, is a #P-complete problem (Orponen, 1990).

Report outline

The first part of this report is dedicated to the description of Dempster-Shafer fundamentals. After some theoretical clarifications about uncertainty notions, we will try to demystify Dempster-Shafer basements and desambiguate its applicability.

We shall see that Dempster-Shafer computations are theoretically alarming. Anyway, we will cope with the challenge of making them feasible for moderated, in fact real-scale applications. So, in the second part, we will focus on the critical implementation aspects of this formalism. We will analyze the existing optimization heuristics from low-level to high-level ones and introduce our personal conclusions and propositions. In particular, our new computation scheme, the *delayed mass valuation*, will be presented.

The third part of this report is dedicated to our main contribution: a C++, general-purpose Dempster-Shafer engine, `eVidenZ`. We will justify its existence through a simple survey of the existing engines and will describe its features and design. This will lead us naturally to the future improvements of our implementation. We will conclude with the research perspectives `eVidenZ` will be applied to.

Notations

In the rest of the report, "D-S" will stand for "Dempster-Shafer".

Part I

Theoretical aspects

Introduction

Evidence theory is often misunderstood because its theoretical foundations can be quite confusing. This is due to the great number of models and justifications that can be found in the literature. The first part of this report aims at clarifying and making an overview of the theoretical background of evidence theory.

First, we will define some basic concepts which will be necessary for the rest of the report. This begins with the notions of ignorance, uncertainty and imprecision, which, even if they are rather intuitive, have a precise meaning. Then the standard aspects of reasoning process are presented, since all the models presented in this report share the same basis.

D-S theory was initiated because of the limitations of the classical probability models. In the second chapter, to discover the foundations of evidence theory, we retrace the successive steps of its history. This is quite essential since almost all the D-S derived models share the same background ideas.

Among the great number of evidence theory models, we decided to present the Transferable Belief Model, which seemed to have the best justifications. This model was developed by Philippe Smets to overcome the common criticisms against D-S related models.

Finally we will make an overview of other well-known models for reasoning with uncertainty. The goal is to provide a global view of the field and to give the main properties of each model, in order to simplify the choice of a particular model and to point out the contribution of evidence theory to the field of reasoning under uncertainty.

Chapter 1

Reasoning with ignorance

Reasoning under uncertainty is a quite vague notion. What does mean reasoning? What is uncertainty? After a short introduction on classical reasoning methods, this chapter focuses on the notion of ignorance and introduces the common components of the reasoning models presented in this report.

1.1 Classical reasoning methods

Classical reasoning methods generally follow this scheme:

- Static pool of induction rules.
- When new truth is available, combine induction rules to prove some hypothesis.

This scheme works fine if one has good and certain data (truths). But what if only uncertain or imprecise data is available? What induction rules should we use? Many theories have been developed to deal with reasoning under uncertainty or more generally under ignorance. Evidence theory is one of these.

Before going further, we should clarify the notion of ignorance. Indeed, the kind of ignorance characterizing a problem will help choosing a good model to make appropriate conclusions.

1.2 What is ignorance?

Ignorance characterizes the fact that some information is missing. But it can take many different forms. We will focus here of the three main notions of ignorance, more details can be found in (Smets, 1991, 1997) and in (Smithson, 1989):

- Incompleteness: some data is missing
- Imprecision: we have an imprecise measure of the data
- Uncertainty: the data might be wrong

These notions and their directly related theories are represented in figure 1.1.

Incompleteness Some data is unknown, but all the available information is correct and complete. These problems are generally solved by brute force using non monotonic logic (which handles backtracking). We won't consider this kind of ignorance in the remaining part of this report. Example: "We don't know the name of John's wife".

Imprecision You have some information, but it is not precise. However, there is no doubt about the information, you are certain that the imprecise value is right. This is generally the field of application of fuzzy sets (see Section 4.3). Example: "John's wife is young".

Uncertainty The information you have can be wrong. This is where probability theory is widely used, as you can represent the fact that the information might be wrong by its probability to be incorrect. Example: An unreliable person (maybe corrupted or deficient) says "John's wife is Mary".

Combination of forms of ignorance Real situations often combine two or more forms of ignorance, for example an unreliable person can tell you than John's wife is young. Evidence theory and possibility theory have been proposed to reason with both imprecision and uncertainty.

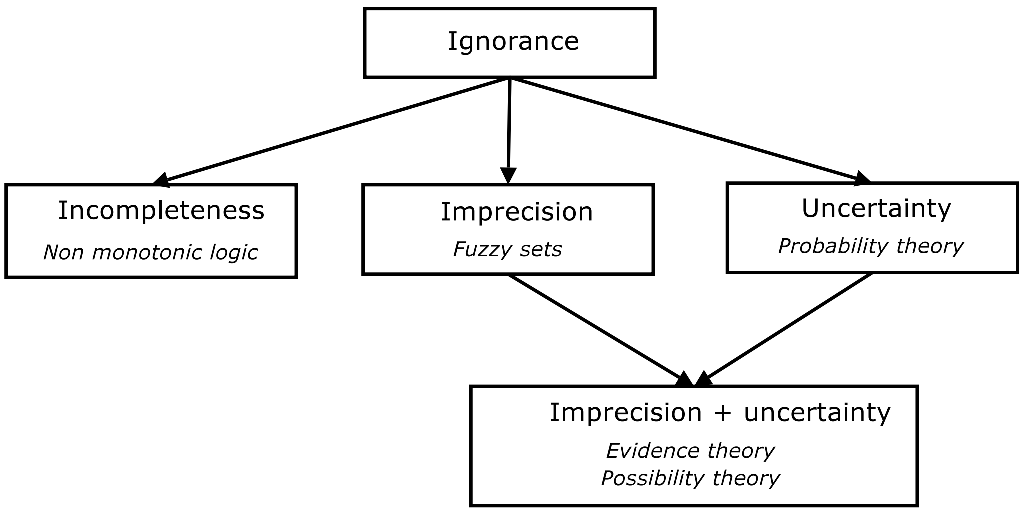

Figure 1.1: Varieties of ignorance

Whatever the kind of problems they are designed for, reasoning methods always respect the same model, as detailed in the next section.

1.3 Principles of reasoning methods

A reasoning process can be divided into three main parts: a static one where knowledge is stored, a dynamic one where knowledge is entertained, and a decision one where a final hypothesis or action is chosen.

1.3.1 Static part

It is necessary to have a way to store already available information, in order to use them later to make deductions. For classical logical models, this can be the set of induction rules and already proved formulas. In a probability formalism, this will be the set of already known probabilities or conditional probabilities, etc.

1.3.2 Dynamic part

If a system can only store knowledge, its role is reduced to a big dictionary. To be useful, a system has to provide a way to analyze and use new data when they are available, and then update the current state of knowledge. This is done in probability theory using Bayes rule for example.

1.3.3 Decision making

Concretely, we use reasoning process to know what to do or to conclude for a given problem. So the system, after having analyzed all the available data should give us the best action to choose if it is possible (otherwise it should tells us that no decision is well founded). For example, in probability theory people generally choose the hypothesis with the higher probability.

1.4 Terminology

In the remaining part of this report, we will use the following terminology:

Frame of discernment (Ω) The frame of discernment is a finite set representing the space of propositions or hypothesis. Basically, making a decision is choosing the best element of Ω. Example for John's wife problem: $\Omega = \{Mary, Jane, Sabrina\}$. Only one element can be true. It is not possible to have several true hypothesis simultaneously. The frame has to be redefined if one wants to give John the possibility to have several wifes.

Open and closed world In a closed world context, the truth has to be into the frame of discernment. In an open world context, the truth may be elsewhere. In the remaining part of this report, we will usually consider only the closed world assumptions, as it is the most convenient for decision making.

Evidence An evidence is an available, certain fact. It is generally the result of an observation. Example: "John's wife has long hair". Basically, a reasoning process consists in using evidences to choose the best hypothesis in Ω.

Having these basic notions about reasoning methods, we can start our study of evidence theory. The next chapter introduces the initial ideas of Arthur Dempster.

Chapter 2

Towards evidence theory

This chapter aims at presenting the original ideas which constitute the foundations of most of the evidence theory models. We will first show the limitations of the classical probability model, the most widely used model to deal with uncertainty. The upper and lower probability model (ULP) was then developed to overcome the classical model, but being too general it appeared to be quite hard to use in concrete applications. This introduced the initial ideas of Arthur Dempster, who restricted the ULP to a special case, enabling more precise judgment. This work was then translated into a more mathematical model using belief functions by Glenn Shafer. This seminal work is the foundation of the various models which were developed later, that's why it is quite important to have some notions about it.

2.1 The cancer diagnosis example

We will look at the differences and the limitations of each model using a simple example, derived from (Smets, 1994). Our goal is not to demonstrate formally the differences between the following models, but rather to give a general intuition.

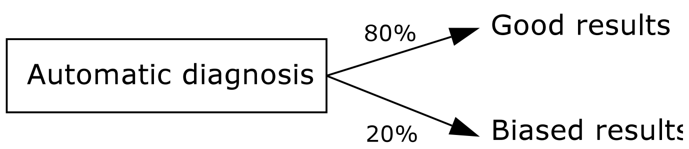

Figure 2.1: The cancer analysis problem

We are considering an imperfect cancer analysis (figure 2.1):

- 80% of the time it gives good and justified results.
- 20% of the time it gives wrong results, biased by the position of the cells.

We know that when the analysis is biased, the result is unrelated to the actual disease.

The problem can be analyzed using the frame of discernment represented on figure 2.2.

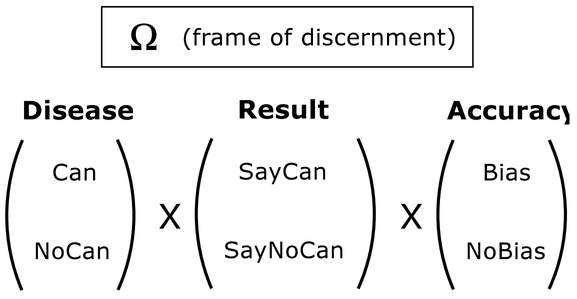

Figure 2.2: Frame of discernment of the cancer diagnosis problem

We use three distinct sets:

- $Disease$: the actual disease or the patient
  - ▷ Can: there is a cancer
  - ▷ $NoCan$: there is no cancer
- $Result$: the result given by the analysis
  - ▷ $SayCan$: the analysis indicates a cancer
  - ▷ $SayNoCan$: the analysis indicates no cancer
- $Accuracy$: was the analysis biased?
  - ▷ $Bias$: the analysis was biased by the position of cells
  - ▷ $NoBias$: the analysis was not biased

Ω is the Cartesian product of the three sets.

Now suppose we make an analysis and the result is "There is a cancer" ($SayCan$). What can we infer about the actual disease of the patient? Let us first analyze the problem using the classical probability model.

2.2 Classical probability model

2.2.1 Quick presentation

Static part

To represent the current knowledge, a probability distribution is assigned on 2^Ω. Probability functions satisfy the rule of additivity, so assigning probabilities to single events of Ω defines the global distribution on 2^Ω, using the simple rule $P(A \cup B) = P(A) + P(B)$.

Dynamic part

When new evidence (certain facts) become available, probabilities can be updated using Bayes' rule:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

It is also possible to express the conditional probability of an event knowing the inverse conditional probability:

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)}$$

2.2.2 Application to our example

We learn that the result of the analysis is $SayCan$. We are interested in $P(Can)$ knowing $SayCan$: $P(Can|SayCan)$.

We have the following properties:

- $P(A|B) = \sum_{i=1}^{n} P(A|B, C_i)$ if $(C_i)_{1 \leq i \leq n}$ is a partition of Ω (extended Bayes rule).
- $P(Can|SayCan, Nobias) = P(NoCan|SayNocan, NoBias) = 1$ as the analysis always tells the truth when there is no bias.
- $P(Can|SayCan, Bias) = P(Can)$ since the result of the analysis when there is a bias is unrelated to the actual disease.
- $P(SayCan|NoBias) = P(Can)$ since if we know that there is no bias, the result of the analysis corresponds to the actual disease.

$$\begin{aligned}
P(Can|SayCan) &= P(Can|SayCan, NoBias)P(NoBias|SayCan) + P(Can|SayCan, Bias)P(Bias|SayCan) \\
&= 1.\frac{P(SayCan|NoBias)P(NoBias)}{P(SayCan)} + P(Can)\frac{P(SayCan|Bias)P(Bias)}{P(SayCan)} \\
&= \frac{0.8P(Can) + 0.2P(Can)P(SayCan|Bias)}{P(SayCan)} \\
&= \frac{P(Can)(0.8 + 0.2P(SayCan|Bias))}{P(SayCan|NoBias)P(NoBias) + P(SayCan|Bias)P(Bias)} \\
&= \frac{P(Can)(0.8 + 0.2P(SayCan|Bias))}{0.8P(Can) + 0.2P(SayCan|Bias)}
\end{aligned}$$

We notice that we need two additional probabilities to compute $P(Can|SayCan)$:

- The *a priori* probability $P(Can)$
- The conditional probability for the analysis to answer "there is a cancer" when the analysis is biased $P(SayCan|Bias)$.

What value should we give to these two probabilities? If it is possible, we should measure these two probabilities and reconsider the problem. But it is not always possible to get these probabilities. We consider the second hypothesis: we cannot get the real values of these probabilities. To continue the reasoning process, a probabilist has to give arbitrary values to these probabilities. Finally, the results of the study will only reflect these arbitrary assumptions.

To reason with this kind of situations, classical probabilities were generalized, introducing upper and lower probability model.

2.3 Upper and lower probability model (ULP)

2.3.1 Quick presentation

The main idea of the ULP model is to work not only on one probability distribution, but on the set of possible probability distributions, induced by the different values the unknown probabilities can take. This set can be represented by its inferior and superior bounds, Π_{inf} and Π_{sup}, as shown on figure 2.3.

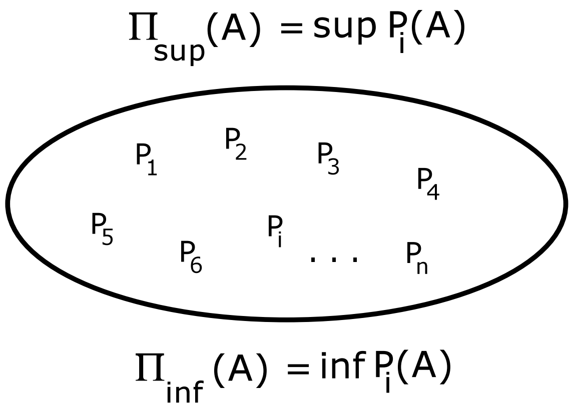

Figure 2.3: Upper and lower probabilities model

2.3.2 Application to our example

In our example, we have 2 unknown probabilities, $x = P(SayCan|Bias)$ and $y = P(Can)$. We can only say that they are comprised between 0 and 1 since they are probabilities. This gives the following values for $\Pi_{inf}(Can|SayCan)$ and $\Pi_{sup}(Can|SayCan)$:

- $\Pi_{inf}(Can|SayCan) = inf_{(x,y)\in[0,1]^2} P(Can|SayCan) = 0$
- $\Pi_{sup}(Can|SayCan) = sup_{(x,y)\in[0,1]^2} P(Can|SayCan) = 1$

So even after knowing the result of the analysis, we are totally ignorant yet. This is due to the high degree of liberty we left in our problem. If we knew at least the *a priori* probability y, we would have:

- $\Pi_{inf}(Can|SayCan) = inf_{x\in[0,1]} P(Can|SayCan) = \frac{y}{0.8y+0.2}$
- $\Pi_{sup}(Can|SayCan) = sup_{x\in[0,1]} P(Can|SayCan) = 1$

So we need to know $P(Can)$ to leave the state of total ignorance. But even in this case, decision making in the ULP model is still an open problem.

In conclusion, classical probability model appeared to be too restrictive, whereas the ULP one seems to be too general, resulting in difficult or impossible decision making.

To avoid this problems, Arthur Dempster (Dempster, 1967) tried to reason without *a priori* probabilities in the ULP model.

2.4 Dempster initial ideas

The main idea of Dempster is to use only available information to reason. Indeed, in the previous models, we used a goal to source approach: we were interested in a particular hypothesis and computing its probability required predefined knowledge about some other probabilities. Thus, if some required probabilities were not available, it was problematic.

2.4.1 Static part

Dempster considers a special case of ULP where:

- probabilities are completely known on a space X.
- for each element of X, we can determine the set of compatible events in Ω.

In our example, we have a well defined probability space on the variable $Accuracy$. We have $P(Bias) = 0.2$ and $P(NoBias) = 0.8$. If we know that the analysis was not biased, we know that $(Can, SayNoCan, NoBias)$ in Ω is not possible. The same stands for $(NoCan, SayCan, NoBias)$. Continuing this process, we can determine for every element of $X = Accurary$ the set of possible events in Ω. This gives a multi-valued mapping Γ from X to Ω, represented on figure 2.4.

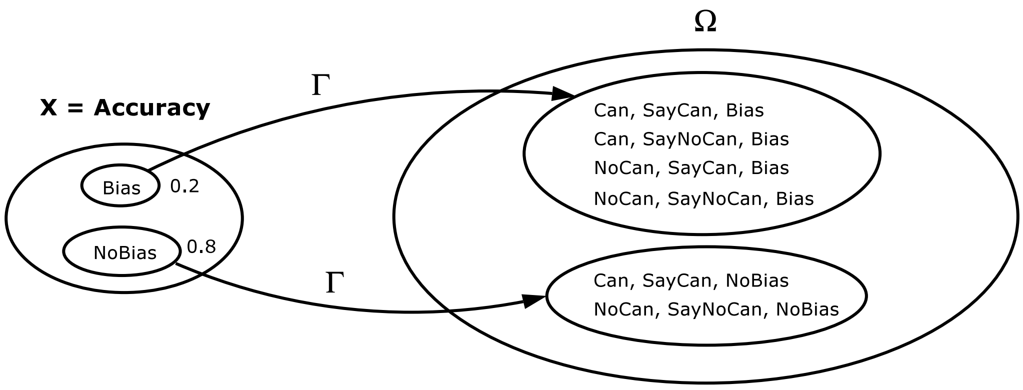

Figure 2.4: Dempster multi-valued mapping

From this mapping, Dempster defines a lower and an upper probability functions on Ω:

- $\Pi_{inf}(A \in \Omega) = \sum_{x \in X; \Gamma(x) \subseteq A} P(x)$
- $\Pi_{sup}(A \in \Omega) = \sum_{x \in X; \Gamma(x) \cap A \neq \emptyset} P(x)$

Mathematically we can verify that Π_{inf} and Π_{sup} are valid upper and lower probability measures.

Less formally, the lower probability of $A \in \Omega$ is the sum of the probability of all events of X whose all compatible events are in A. The idea is that if $x \in X$ is compatible only with events of A, having x implies having A. So the probability of x completely supports A.

The upper probability can be seen as the complement of the lower probability: $\Pi_{sup}(A \in Omega) = 1 - \Pi_{low}(\Omega\ A)$. The idea is that if one element $x \in X$ has no compatible events in A, having x will never support A. In the opposite, if x has at least one compatible event in A, having x may imply A. The upper probability is the lower probability plus the probabilities of the elements of X giving only partial support.

2.4.2 Dynamic part

We learn $SayCan$. How should we update the system? Dempster just intersects Ω with the new evidence. X is not modified. The new mapping for our example is represented on figure 2.5. Elements incompatible with $SayCan$ (those containing $SayNoCan$) have been removed.

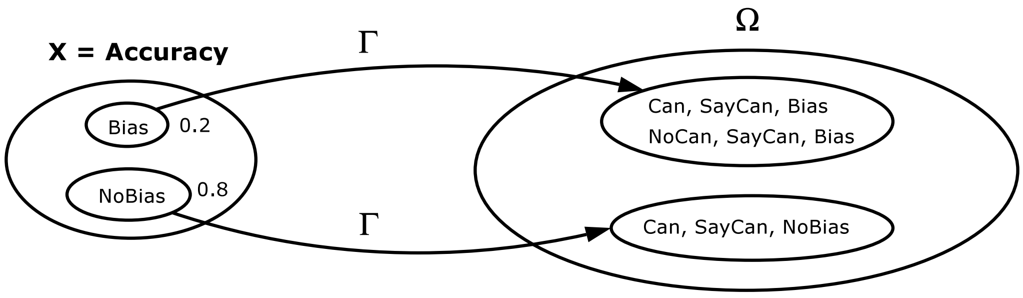

Figure 2.5: Mapping after having learnt $SayCan$

One may ask why probabilities on $X = Accurary$ are not updated when we learn $SayCan$. This would mean $P(Bias|SayCan) = P(Bias)$ and $P(NoBias|SayCan) = P(NoBias)$, which leads to strange and ill founded assumptions. This point is the main weakness of Dempster formalism, as it was argued in (Levi, 1983).

Dempster also provided the so called Dempster combination rule, which defined how to combine mappings induced by different observations. This rule will be explained in Chapter 3, with a better formalization to explain it.

2.5 Shafer's work and other proposed models

Two fundamental notions have been introduced by Dempster, and will be the common part of all derived models:

- Describing the current knowledge on well known spaces.
- Having a relationship between this well known space and the actual frame of discernment.

- Having bounding measures, the lower one considering only the elements completely supporting the set of hypothesis, the upper one also considering the elements partially supporting the set.

In his book (Shafer, 1976), Shafer used Dempster work, and introduced the notion of belief functions. He mainly worked on the mathematical aspects, not considering a multi-valued mapping but basic mass functions, as we will see in Chapter 3. But Shafer did not corrected the justification weaknesses of Dempster theory.

Many other formalisms were developed around these seminal ideas, using random sets, propositional logic (Provan, 1990) or hints (Kohlas and Monney, 1995). A review of the different approaches can be found in (Kohlas and Monney, 1994). In the remaining part of this report, we will focus on the Transferable Belief Model by Smets, as it appeared to be the most natural and justified one. In practice, choosing a particular model isn't that important since numerical results generally remains identical.

Chapter 3

The Transferable Belief Model

Relying on the Transferable Belief Model, this chapter introduce all the important aspects of evidence theory, including static part, dynamic part and decision making. Then we will extend the framework of the TBM to multi-variable problems, in order to simplify and optimize big systems. Finally, this chapter will give a short summary of the properties of evidence theory.

3.1 General principles

The TBM is a mathematical model designed to represent degrees of beliefs. It is quite close to Shafer's approach with belief functions and basic belief masses, excepted that Smets does not assume any unknown probability distribution on Ω. Smets voluntarily take some distance from probability theory. The TBM is not a generalization neither a special case of it. This approach leads to a better justification.

The TBM (Smets et al., 1991) is a two-level model:

- the **credal** level where beliefs are assessed, updated and combined.
- the **pignistic** level when decisions have to be made.

3.2 Credal level: entertaining knowledge

3.2.1 Static part

Mass functions

The TBM is based on the notion of basic belief mass function (m). A mass is a finite amount of support given to some groups of hypothesis of Ω. This support is generally induced by the available data, or by a probability distribution on a well-known space (as Dempster mapping does). Assigning a mass $m(A)$ on a subset A of Ω gives some support to **exactly** the subset A, and not to a more specialized subset.

A mass function must verify $\sum_{A\subseteq\Omega} m(A) = 1$. It is different from a probability function in that it does not have to respect additivity: $m(A \cup B) \neq m(A) + m(B)$. So a mass function is not determined by its distribution on the singletons of Ω, values can be given to subsets of Ω.

The **focal elements** are the subsets having a non null mass: $m(A) \neq 0$. The set of focal elements is called the focal set FS.

Assigning mass values Mass values can be deduced using several ways:

- some probability distribution on a well-known space.
- betting behaviors and varying betting frames (Smets and Kennes, 1994).
- subjective assumptions.

Subjective assumptions are necessary when the problem is not rigorously determined by numerical values, as in the following example:

- We search the name of John's wife.
- It must be either Mary, Jane or Jill.
- An old person tells you it is Mary or Jane.
- You know that sometimes, this person makes assertions based on obsolete memories.

The figure 3.1 gives the mass assignment for this example. The values of 0.8 and 0.2 were chosen subjectively, representing our confidence into the person. 0.2 is assigned to Ω and not to John because when the person bases her assertion on obsolete memories, she can say anything, not only wrong things.

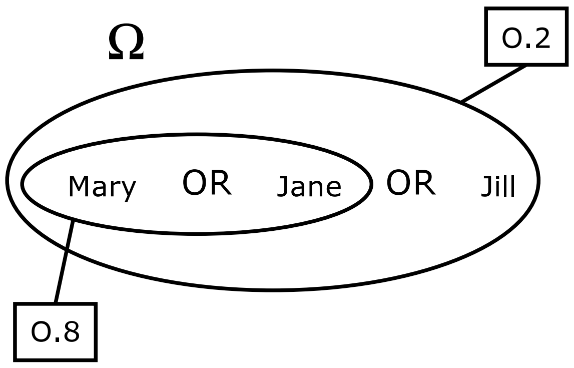

Figure 3.1: Mass assignments for John's wife example

Interesting functions

Now we have defined our basic belief mass (bbm) supports on Ω, several interesting functions have to be introduced:

Belief on $A \subseteq \Omega$ is the amount of **justified** support to A. Every $m(B)$ such as $B \subseteq A$ completely supports A, so it provides justified support to A. Mathematically, this gives:

$$bel(A) = \sum_{\emptyset \neq B \subseteq A} m(B)$$

In probabilistic formalisms, belief is generally seen as the lower probability function of Dempster. But here belief is totally unrelated to any probability distribution on any space.

Plausibility on $A \subseteq \Omega$ is the amount of potential support to A. Plausibility takes in considerations the support not totally allocated to A, but which at least support some part of A, in others words, support which is not strictly given to $\overline{A}$. Mathematically:

$$pl(A) = 1 - bel(\overline{A}) = \sum_{B \cap A \neq \emptyset} m(B)$$

Of course, justified support is part of the potential support, thus $pl(A) = bel(A) + partial_support(A)$. This results in $pl(A) >= bel(A)$. In probabilistic formalisms, plausibility is generally seen as the upper probability function of Dempster.

Ignorance on A is the difference between $pl(A)$ and $bel(A)$. It is the partial only support given to A. So it is the sum of the masses $m(B)$ such that $B \cap A \neq 0$ and B is not included in A. These masses are allocated to non singletons elements of Ω because there is not enough information to discriminate among the elements of B. This lack of information is what we call ignorance.

$$ign(A) = pl(A) - bel(A)$$

Doubt on A is the measure of the support which will never be assigned to A.

$$dou(A) = 1 - pl(A) = bel(\overline{A})$$

Commonality on A does not have a real intuitive sense, but it can be used to optimize some computations, as shown in Section 8.2. It is a measure of the support which can move to every elements of A, new information should become available. The may support can move to more specific subsets is explained in section 3.2.2.

$$com(A) = \sum_{A \subseteq B \subseteq \Omega} m(B)$$

Interactions between measures Plausibility, belief, commonality and mass functions are dual functions. Knowing one function, we can calculate any other one. For example, using the Möebius transform on bel, we get:

$$m(A) = \sum_{B \subseteq A} (-1)^{|A|-|B|} bel(B)$$

Particular case when $\forall A, ign(A) = 0$ This corresponds to the case where m is only assigned to singletons. Thus $bel = pl$ and is a probability function. So when we are in the special case where there is no ignorance, we fall back to the classical probability theory, but for the static part only.

3.2.2 Dynamic part

Conditioning

Suppose we have a support $m(A)$ on A. Now we learn that a set of hypothesis $B \subseteq \Omega$ is impossible. So the mass originally supporting A should now support $A \cap \overline{B}$. Thus, $m(A)$ is transferred on $m(A \cap \overline{B})$. This corresponds to Dempster's mapping intersection in subsection 2.4.2.

Let's reconsider $ign(A)$. It was the amount of support which was only partially assigned to hypothesis of A. The ignorance stands in the fact we don't know if this amount will be transferred to hypothesis of A only, or transferred to hypothesis of $\overline{A}$ if new evidences become available.

Conditioning is a particular case of a more general dynamic process: combination.

Combination using Dempster's rule

Notion of potential Suppose you ask the name of John's wife to another person. This person induces a new mass function or our system. If the two persons are independent and do not influence each other, we have two mass functions induced by different observations. Such a mass function is called a potential, noted ϕ. Combination is the process which defines how to combine several potentials induced by distinct observations into a single one.

Many rules exist in the literature, depending on the properties wanted. Some rules can combine non totally independent potentials. A survey of usual combination rules is given in (Sentz and Ferson, 2002). In the remaining part of this report we will consider only the so-called Dempster combination rule applied to the TBM. More details can be found in (Smets, 1990a).

Suppose we have two potentials ϕ_1 and ϕ_2, with their associated mass functions m_1 and m_2. We want to combine them to get a global potential ϕ_{12}. Dempster combination relies on the following natural idea: the product $m_1(X) * m_2(Y)$ supports $X \cap Y$. The rule is axiomatically constructed in (Smets, 1990a), but it can be compared to the rule $P(A \cap B) = P(A) * P(B)$ to understand it intuitively. Thus, we get:

$$m_{12}(A) = m_1 \oplus m_2 = \sum_{X \cap Y = A} m_1(X) * m_2(Y)$$

However, a problem appears when $X \cap Y = \emptyset$, $m(X) \neq 0$ and $m(Y) \neq 0$, that is, $X \in FS(\phi_1)$ and $Y \in FS(\phi_2)$. This means we have a conflict between ϕ_1 and ϕ_2, because one is strictly supporting a set of hypothesis, and the other one supports a completely disjoint set of hypothesis. If we are in an open world (the truth can be elsewhere), it sounds good to give support to external hypothesis, not in Ω. So the mass $m_1(X) * m_2(Y)$ would go to $\emptyset$. In a closed world, to keep the properties of mass values adding to 1, it is necessary to normalize the mass function with the lost mass. The total conflict is:

$$k_{12} = \sum_{X,Y : X \cap Y = \emptyset} m_1(X) * m_2(Y)$$

Then combination becomes:

$$m_{12}(A) = \frac{\sum_{X \cap Y = A} m_1(X) * m_2(Y)}{1 - k_{12}}$$

Now the combination rule is stable, two combined potentials result in a valid potential. In addition, Dempster's rule is commutative and associative, which are valuable properties.

Weighted combination Sources of information may have different reliability. It could be interesting to have a way to weight the potentials by their reliability. This can be done by assigning a non null mass m to Ω. The bigger m is, the less importance the potential has during the combination. Indeed, intersecting a focal element with Ω let it unchanged in the resulting potential.

Conditioning is a particular case of combination, since combination with a potential defined by $m(\Omega \cap X) = 1$ is equivalent to conditioning by X.

Complexity It should be noticed that the combination rule makes intersections between the focal elements of each potential, and the number of focal elements is up to $2^{|\Omega|}$. Actually, the complexity of combination is #P complete (Orponen, 1990). To get a practical idea about the time and memory required for Dempster's combination, you can refer to the appendix A. These performance problems are the origin of the computational issues discussed in the second part of this report.

3.3 Multi-variable TBM

In many practical cases, the frame of discernment is composed of several **variables**. In our example of cancer diagnosis, there were 3 variables, $Disease$, $Result$ and $Accuracy$. Usually sources of informations concern a few variables only, for example information about the probability to have a biased analysis only concern $Accurary$. A set of variables is called a **domain**. The possible values of a variable are called its **realizations** and an element on a domain is called a **configuration**.

We may want to define a potential on a particular domain only. Working on multi-variable problem has several advantages:

- It simplifies the assessments of mass values as they can be defined on the interesting part of the problem only.
- Potentials may be combined cleverly to work locally on the smallest possible domains using propagation networks, thus reducing the complexity of the combination. In huge systems, this can save a lot of time and memory (see Section 8.2).

Further details can be found in (Lehmann, 2001).

It is possible to combine several potentials defined on different domains using two basic operations: extension and projection.

3.3.1 Extension

Let ϕ a potential on a domain D. Its extension on $D2 \supset D$ is done by applying a cylindrical extension of the focal elements of ϕ. For example, if:

- Ω has two variables X and Y with respectively $\{x1, x2\}$ and $\{y1, y2\}$ as possible values (called realizations)
- ϕ is defined on $D = X$
- $m(\{x1\}) = 0.2$ and $m(\{x2\}) = 0.8$

The extension of ϕ on $D2 = X, Y$ noted $\phi^{\uparrow D2}$ is defined by the mass:

- $m(\{(x1, y1)\} \cup \{(x1, y2)\}) = 0.2$
- $m(\{(x2, y1)\} \cup \{(x2, y2)\}) = 0.8$

3.3.2 Marginalization

Marginalizing is the opposite transformation of extension. Let ϕ a potential on a domain D. Its marginalization on $D2 \subset D$ is done by projecting its focal elements on $D2$. Of course projection can result in different focal elements becoming identical, so regrouping may be necessary. For example, if:

- Ω has two variables X and Y with respectively $x1, x2$ and $y1, y2$ as possible values (called realizations)
- ϕ is defined on $D = X, Y$
- $m(\{(x1, y1)\}) = 0.1$, $m(\{(x1, y2)\}) = 0.1$ and $m(x2, y1) = 0.8$

The marginalization of ϕ on $D2 = X$, noted $\phi^{\downarrow D2}$ is defined by the mass function:

- $m(\{x1\}) = 0.1 + 0.1 = 0.2$
- $m(\{x2\}) = 0.8$

3.3.3 Combination of two multi-variable potentials

Let ϕ_1 a potential defined on the domain $D1$ and ϕ_2 on $D2$, we have:

$$\phi_{12} = \phi_1^{\uparrow D1 \cup D2} \oplus \phi_2^{\uparrow D1 \cup D2}$$

The resulting potential ϕ_{12} is defined on $D = D1 \cup D2$. We may want the resulting potential to be restricted on a particular domain $D3$, this operation is called fusion:

$$fusion(\phi_1, \phi_2) = (\phi_1^{\uparrow D1 \cup D2} \oplus \phi_2^{\uparrow D1 \cup D2})^{\downarrow D3}$$

3.3.4 Application to the cancer diagnosis problem

Initially, we are is a state of total ignorance:

- $m(\Omega) = 1$ (closed world)

We know that the analysis is right if there is no bias: $(Can, SayNoCan, NoBias)$ and $(NoCan, SayCan, NoBias)$ are impossible. We can apply conditioning, then we get:

- $m(\Omega \backslash \{(Can, SayNoCan, NoBias), (NoCan, SayCan, NoBias)\}) = 1$

On the variable $Accurary$, we have ϕ_2 defined by $m(Bias) = 0.2$ and $m(NoBias) = 0.8$. Extending it on the $Disease, Result, Accurary$ and combining with the current global ϕ, we get ϕ:

- $m(\{(*, *, Bias)\}) = 0.2$
- $m(\{(Can, SayCan, NoBias), (NoCan, SayNoCan, NoBias)\}) = 0.2$

With "$*$" a shortcut representing all the possible values of the variable.

Then we learn $SayCan$, using conditioning we have:

- $m(\{(*, SayCan, Bias)\}) = 0.2$
- $m(\{(Can, SayCan, NoBias)\}) = 0.8$

We are interested in the actual disease, so we marginalize on $Disease$:

- $m(\{Can, NoCan\}) = 0.2$
- $m(\{Can\}) = 0.8$

The justified support for a cancer is $bel(Can) = 0.8$ and the potential support for a cancer is $pl(Can) = 1$. Indeed, all available information at least partially support Can.

The justified support for not having a cancer is null. But its potential support is 0.2 (the case where the analysis was biased).

3.4 Pignistic level: decision making

The actual goal of reasoning processes is to make decisions. After having entertained our knowledge, combined potentials and updated our beliefs, etc., we want to use these collected informations to choose the best hypothesis or group of hypothesis. There is no standard rule for decision making in evidence theory. Many rules are dedicated to a particular problem, and tuned to fit the requirements.

Studies on decision in evidence theory can be found in (Smets, 2001) and (Yager, 1990).

3.4.1 Ad hoc rules

Problem dedicated rules are generally based on the two following criteria:

- Maximum of belief. This rule will require justified, dedicated support to the concerned hypothesis.
- Maximum of plausibility. This rule tries to find the less contradicted hypothesis, even if it does not have any specific support.

People often use a combination of these two criteria, such as $max(bel(A) + pl(A))$, or condition it with more severe rules like $max(bel(A))$ if $bel(\overline{A})$ is not too big, etc. This kind of rule has been discussed by several authors, especially in (Appriou, 1991), (Guan and Bell, 1991) and (Shafer, 1976).

Using these rules is quite problematic since the Dutch book arguments state that decision making for a rational agent must be done using probabilities to be consistent ((Teller, 1973), (Jeffrey, 1988)). One may accepts irrational decisions, but it would be more convenient to have a well-founded decision process. This is the goal of Smets's pignistic transformation.

3.4.2 TBM decision process

The goal of the pignistic level is to construct probabilities from belief functions when decision must be made. Probabilities are constructed only at this final stage, in that the TBM is different from the Bayesian theory. Indeed, knowledge was entertained using evidence theory rules, not Bayes conditioning rule. In so doing, the TBM resists to Dutch book arguments, as stated in (Smets, 1993).

Let A and B be two hypothesis of Ω, such that $m(A \cup B) = 0.8$. What probabilities should we assign to A and B individually? We have to find them to make decision. The insufficient reason principle tells us to assigns the same probability to A and B: $0.8/2$. If we had a mass on three hypothesis A, B and C, the probability allocated individually to A, B, and C would be $0.8/3$. We obtain the following formula to find the probability function called Bet_p:

$$\forall x \in \Omega, Bet_p(x) = \sum_{x \in A \subseteq \Omega} \frac{m(A)}{|A|}$$

This is the only rule satisfying then consistency constraint. An axiomatic derivation of this probability function is detailed in (Smets, 1990b).

Using the TBM pignistic probability function, we can rely on usual probabilistic decision rules. This enables the use of well-known decision analysis tools, such as probabilistic decision trees, etc.

3.5 Review of the important properties

This section wants to fix the ideas about the possibilities evidence theory brings.

3.5.1 Knowledge representation

Evidence theory has interesting properties for knowledge representation:

- Represent uncertainty, imprecision and (even total) ignorance.
- No extra assumptions needed. The model only uses available information and does not require (for the static part) any additional information.
- Provides a simple and flexible way to give support to a set of hypothesis.
- The support may be partial, only a degree between 0 and 1 is assigned.
- Several measures can be retrieved, such as ignorance, doubt, belief and plausibility.

It may be difficult in practice to determine accurate mass values. But this is a problem concerning every numerical methods. Symbolic methods should be used if it is impossible to find accurate numerical values (Parsons and Hunter, 1998).

3.5.2 Knowledge fusion

Multi-source fusion is the most frequent use case of evidence theory, it has several appreciable properties:

- Dempster's rule is quite simple, commutative and associative.
- Other rules are available (including disjunctive rules).
- Conflict evaluation is possible.
- It is possible to fusion different sources of information even if they are only relevant on a subpart of the problem.
- The reasoning process is clear and understandable (this is particularly useful for medical diagnosis assistance for instance).

3.5.3 Decision making

Since many ad hoc rules are possible, it is generally easy to find one that fits correctly into the problem. It no ad hoc rule seems really obvious or if one needs probabilistic decision tools, a probability distribution can be constructed.

Now that we have a global vision of evidence theory, the next chapter will compare it to its most famous competitors. This will hopefully raise the usefulness of D-S's framework.

Chapter 4

Comparison with other numerical reasoning models

In artificial intelligence, it is usually impossible to prove formally that a model is better than another, or even that a model is accurate for a given problem. Fortunately, considering the fundamental axioms and the expressive power of every model, a human can choose the one which best fits into his problem. This chapter will briefly present the main properties of ad hoc models, classical probability theory, fuzzy sets and possibility theory. Our goal is to give a general overview of the most common models to establish the position of evidence theory. A more complete review of these models can be found in (Parsons and Hunter, 1998) and (Smets, 1997).

4.1 Ad hoc models

The most widely used ad hoc reasoning method is certainly the certainty factor model. It is rule based system, initially developed for MYCIN (Shortliffe, 1976). The idea was to give a factor of certainty to the rules of the system. Unfortunately, it cannot be justified theoretically, and even in practice rule based system are known to give wrong results in the field of uncertainty (Heckerman, 1986).

4.2 Probability theory and Bayesian networks

4.2.1 Knowledge representation

Probability theory deals with uncertainty only. A value has to be assigned on every hypothesis. Therefore, if there is some ignorance, extra assumptions are required (see Section 2.2). In addition, probability theory is only well suited for *random* processes, whereas evidence theory can work with *arbitrary* processes.

4.2.2 Knowledge fusion

The common way to update probabilities is Bayes' rule. In practice, this rule is generally used under the form:

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)}$$

Using it requires both the *a priori* probabilities of A and B, and the other conditional probability. The number of probabilities to give to a probabilist framework can be quite big, and some probabilities are generally unknown, resulting in arbitrary values. Then the reasoning process only reflects the value of these arbitrary assumptions.

It is also possible to fusion several probability distributions, using opinion pools (Genest and Zidek, 1986).

The number of initial probabilities can be greatly reduced using causal networks, also called Bayesian networks (Pearl, 1990b,a). The idea is to add structural information to the system, representing independence between variables. Thus, the dimension of the problem can be greatly reduced. This changes the complexity of probabilistic computations, but does not add expressive power to probability theory.

4.2.3 Decision making

Decision within the probability framework has been heavily studied. Many tools are available, for instance using an expected utility function. More simple approaches just choose the hypothesis with the best probability. The same rules can be used in evidence theory using the pignistic transformation of the TBM.

4.3 Fuzzy sets

4.3.1 Knowledge representation

Fuzzy sets theory (Zadeh, 1965) is well-suited to represent imprecision and vagueness (see section 1.2) to get more details about the notion of imprecision. The idea is to extend the notion of belonging to a set. Instead of having a complete or null membership to a set Ω, a degree of membership is assigned, using a function μ_Ω. For example, to represent the degree of membership to the fuzzy set $S =$ "young people", one may use the function μ_S, represented on figure 4.1.

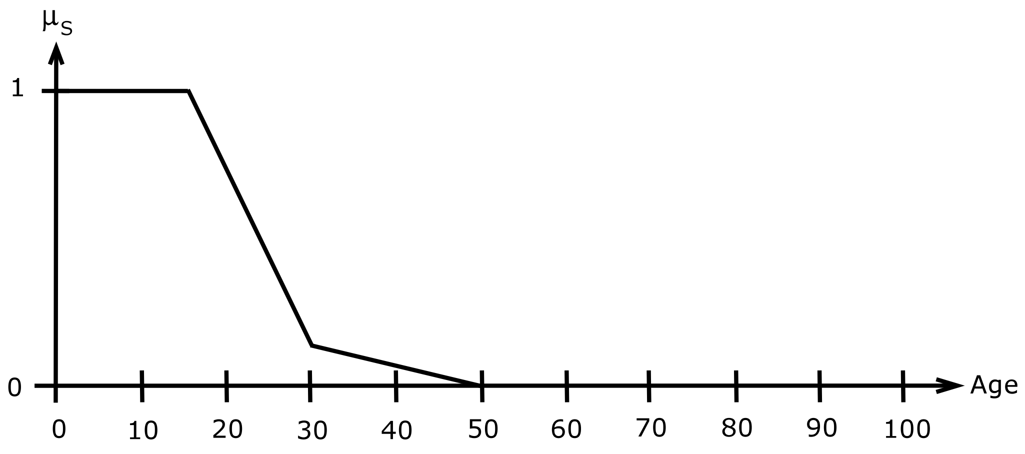

Figure 4.1: μ_S for young people

Fuzzy sets theory cannot handle uncertainty, we have to be certain of the degree of membership of an element. Especially, it is not possible to assign a degree of membership to a set of hypothesis. This is why this theory has to be reserved to imprecise but certain problems, other ones should certainly use evidence theory.

4.3.2 Knowledge fusion

Several operators can be used to fusion fuzzy membership functions. The most current ones are min and max operators. They represent respectively the notion of "the element has to belong to all the given sets" and "the element has to belong at least to one of the given set". Average operators may also be used, representing the notion of "global adequacy with the given sets". The variety of combination operators makes fuzzy sets theory quite flexible and powerful (Bloch, 1996).

4.3.3 Decision making

Simple rules can be used, depending on the problem. For instance, one can choose the element which has the best membership. Ambiguity detection can be added by checking the membership value of the second best element. Further information can be found in (Dubois and Prade, 1982).

4.4 Possibility theory

4.4.1 Knowledge representation

Although not necessary so, possibility theory is generally defined over fuzzy sets (Zadeh, 1978). The goal is to add support for uncertainty to the fuzzy sets theory. To do this, a possibility and a necessity measures are defined on 2^Ω such that:

$$\Pi(A \cup B) = max(\Pi(A), \Pi(B))$$

and

$$N(A \cap B) = min(N(A), N(B))$$

The possibility distribution on the events is done by considering how possible they are. The main difference with evidence theory is in the notion it deals with: possibility is quite different from belief. An event may be possible even if we don't believe it will happen. The same difference exists with probability theory, an event may be possible but not likely to happen. Zadey explained it with the example of Hans's breakfast:

- Hans ate X eggs for breakfast with $X = \{1, \ldots, 8\}$
- We can associate a possibility distribution based on the ease with which Hans can eat eggs $\Pi_X(u)$
- We can associate a probability distribution based on the observations of the actual number of eggs eaten by Hans $P_X(u)$

The possibility and probability distributions are detailed on figure 4.2.

u	1	2	3	4	5	6	7	8
$\Pi_X(u)$	1	1	1	1	0.8	0.6	0.4	0.2
$P_X(u)$	0.1	0.8	0.1	0	0	0	0	0

Figure 4.2: Probability and possibility distributions for Hans's breakfast

So even if Hans can eat 8 eggs, he can be very unlikely to do so.

The possibility theory is often confronted to evidence theory, as belief and plausibility functions may seem similar to necessity and possibility functions. Differences on concrete applications are discussed in (Smets, 1990c; Dubois et al., 2001).

4.4.2 Knowledge fusion

Several rules are possible, generally problem dependent. Both conjunctive or disjunctive rules are possible, and other combination modes enabling reinforcement effects were introduced (Dubois and Prade, 1990). This results in a great flexibility for the fusion of possibility distributions.

4.4.3 Decision making

Standard decision rules are quite close to D-S ad hoc rules, using the max of possibility, the max of necessity, etc. People usually add checking constraints to ensure that decision are not ambiguous. The decision rule to choose is almost always dependent of the context.

4.5 Conclusion

Each model has a quite well defined field of application. Choosing the best model for a given problem consists in determining which notions need to be handled. Sometimes several notions have to be represented, thus several models may have to be used together.

We saw that probabilities are well-suited to random processes without imprecision, fuzzy sets theory is well suited for imprecise data only, and evidence theory and possibility theory can handle both uncertainty and imprecision. The differences between the two latter ones essentially consists in the notions they deal with. So even if evidence theory can handle simple uncertainty or imprecision, its usefulness is actually raised when dealing with complex problems including both forms of ignorance. The next chapter will present the concrete range of applications where evidence theory has a real advantage over other theories.

Chapter 5

Applications of evidence theory

Many examples of DS applications can be found in the literature (Sentz and Ferson, 2002). Through three simple examples, this chapter wants to present some typical use cases of evidence theory. The first section describes the common scheme where people use the DS model, and the following illustrates it with image classification, color image segmentation, medical diagnosis and belief decision trees.

5.1 Classical scheme

Evidence theory is a flexible and powerful theory. However, as it requires high computation time, it is used only when it has a real advantage. The previous chapter pointed out the domains which are already dealt with efficiently. Thus, evidence theory is mainly in the following scheme:

- Several imprecise or unreliable sensors.
- Each of them only give partial information or concerns a few number of variables.

This scheme intensively use the fusion power and the possibility to give support to set of hypothesis. Indeed, a sensor which only have partial information won't be able to assign individual support to each hypothesis, but on the sets of hypothesis it could discriminate. Here fuzzy sets and probability theory cannot handle the situation properly and directly.

This scheme is far from being unrealistic. It will be illustrated in the following sections, though real life examples. Other use cases of evidence theory can be found in (Smets, 1999), and a great and quite exhaustive bibliography is detailed in the appendix A of (Sentz and Ferson, 2002).

5.2 Image classification

In (Le Hégarat-Mascle et al., 1998), the problem of satellite image classification is considered. The authors dispose of two sources of information, an optical image sensible to cloud cover and a radar image, not sensible to clouds but with a low quality. We get the two following potentials:

- One induced by the optical image. It there are clouds on some parts of the image, this potential cannot make a good classification on these regions. On these regions the mass functions should allocate a great value to Ω to represent the high ignorance.

- The other one induced by the radar image. Radar image are generally disturbed by speckle. This induces a determined amount of ignorance.

In this example, the first contribution of evidence theory is to represent the imprecision in the radar image and the absence of knowledge in the optical image (when there are clouds). Then the fusion with Dempster's rule allows the classification to give more importance to the radar image when ignorance in the optical one is too high.

5.3 Color image segmentation

In (Vannoorenberghe et al., 1999), the author wants to make color image segmentation using the three color channels red, green and blue as three distinct sources of informations. Segmentation of each color plane gives a particular set of classes. The intersection of the classes induced by each color place is the frame of discernment, as shown on figure 5.1.

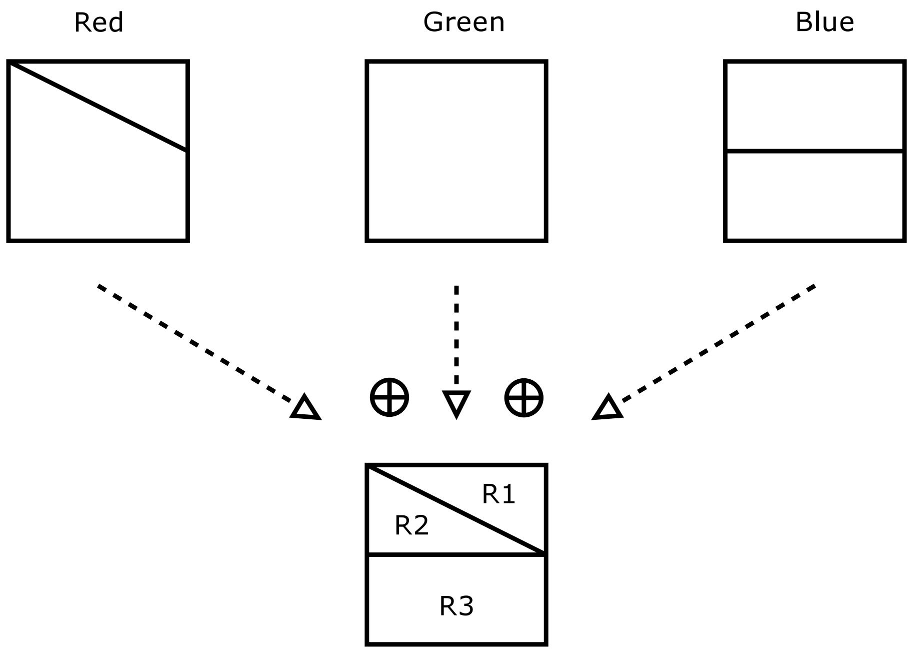

Figure 5.1: Classes induced by the color planes

Now each color plane constitutes a potential, with mass support on set of classes. For instance, the red plane cannot discriminate $R2$ and $R3$, so it will assign a mass to $R2 \cup R3$. Then the three potentials are combined to make the final segmentation.

In this example, we are clearly in the classical scheme of the section 5.1.

5.4 Medical diagnosis

Several source of informations are given by the different measures or symptoms observed. One symptom generally supports a set of diseases. So we are again is the scheme where evidence theory is completely adapted. One of the best diagnostic system handles more than 60 diseases and 100 symptoms, using the Pathfinder system (see Chapter 9).

5.5 Belief decision trees

Decision trees are widely used in the field of supervised learning. The goal is to classify objects by considering their attributes. Classical decision trees are organized as follows:

- A training set T stores pairs of (attributes, class).
- A tree is then constructed, with each node representing an attribute, each branch a value of the nodes' attribute and the leafs are the final classes.

We illustrates this with the following example:

- Classes are men, women and unknown.
- Attributes tested are the size of the person (tall or small) and the length of the hairs (long or short).
- Our training set is $\{(tall, short), man\}$, $\{(tall, long), unknown\}$, $\{(small, short), unknown\}$, $\{(small, long), woman\}$.

The figure 5.2 shows a possible decision tree for this example. Algorithms to construct decision trees generally take into consideration the entropy induced by the attributes to choose the best partitioning (?).

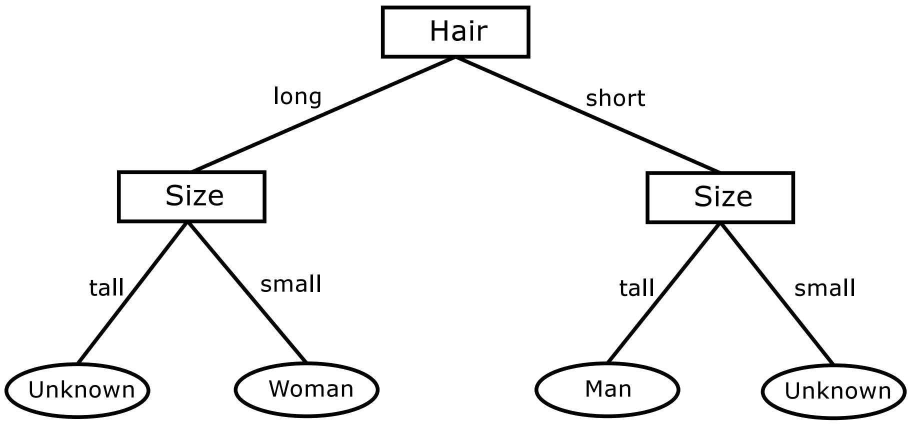

Figure 5.2: Decision tree for men and women

This kind of decision tree is useful when classes and attributes are well determined. Evidence theory has been applied to decision tree when the classes in the training set are not completely determined (Elouedi et al., 2001, 2000). In this case, classes are replaced by a basic belief assignment over the classes (a mass function).

One may wonder if this kind of tree is really useful. Let's consider a classification between two diseases A and B induced by a decision tree using symptoms as attributes. You make a statistical study on 1000 persons to get a representative training set and construct the decision tree. A few month later, scientists discover that the disease B can be subdivided into two distinct diseases C and D. So you make a new statistical study to discriminate between C and D, but it would be fine to use the already existing study on the first 1000 persons to classify between A and $B = C \cup D$. In this simplified situation, you may use the two training sets on A, C, D, by assigning a mass on $C \cup D$ for the examples which were classified as B in the first study.

Another great interest of evidence theory in decision tree would be to replace completely defined attributes by belief attributes. This has not yet been studied in depth in the literature.

Conclusion

This first part of the report was dedicated to the theoretical basis and interests of evidence theory. We showed that Dempster and then Shafer's seminal ideas were initiated by the limitations of classical probability and upper and lower theories, the former being too restricted, and the latter being too general.

Among the great number of models developed around evidence theory, we presented the Transferable Belief Model, whose justification seems natural and healthy. Using this formalism, we could point out the expressive power of D-S theory.

Having in mind the properties of the theory, we compared it with the other common models: certainty factors, probability, fuzzy sets and possibility theories. This raised the contribution of evidence theory to the field of uncertain reasoning.

Finally, thought simple but representative examples, we wanted to give a general intuition of the fields of application where D-S theory may be relevant. The classical use cases generally consist in the fusion of several imprecise or partial sources of information. This characterizes a large set of concrete problems, such as image classification, color image segmentation, medical diagnosis and classification using decision trees.

Now that we have an overview of the theoretical aspects of evidence theory, the second part of this report will concentrates on the efficient implementation of D-S computations.

Part II

Implementation aspects

Towards an efficient implementation

D-S theory encounters a certain success as a well-founded model of human reasoning under uncertainty, as we saw it in the first part of this report. In spite of this theoretical success, D-S theory was rarely applied to concrete problems so far. One of the main points of criticism this framework has to face is its computational complexity. Indeed, combining evidence pieces using Dempster's rule of combination is known to be #p-complete (see Subsection 3.2.2) so that computations become quickly infeasible in practice. Researchers have raised empirical and theoretical principles to make this formalism more usable. To cope with this huge challenge of making D-S computations feasible, solutions proposed concern both structural, implementation aspects and algorithms. This part of the report has the aim to make a global overview of the efforts made for improving D-S usability.

This part is structured as follows: first, we will propose relevant data structures to represent D-S core components. We will principally discuss the representation of focal elements and potentials. Last but not least, we will present a study of a new design principle, the *delayed mass valuation*, which induces a new vision of belief masses.

Second, we will describe efficient algorithms to implement D-S mechanisms: projection, extension, combination, marginalization and fusion.

Third, we will introduce advanced heuristics proposed by researchers for speeding up D-S computations. We will then present approximation methods, fast Moebius transformation and hypertree-based optimizations.

Chapter 6

Efficient data structures

Designing a D-S reasoning engine, one must consider the way main data structures will be implemented. As we will see, structural choices significantly influence the final performance. This chapter is dedicated to an overview of the relevant solutions proposed in existing papers, like (Lehmann, 2001), (Haenni and Lehmann, 2001) and (Xu and Kennes, 1994). We discuss principally the representation of focal elements and potentials. We also present our conclusions concerning a new design principle, the delayed mass valuation, which affects our vision of belief masses.

6.1 Prerequisites: the Clue game example

In order to illustrate the different solutions proposed in this chapter, we will use the example of a murder inquiry based on a simplified Clue game. The Dr. Black was murdered and players are looking for the murderer, the room and the weapon. This game directly implies a simple modeling of an underlying D-S system. This system includes 3 variables:

- $Murderer$ with 3 realizations:
  - ▷ "Colonel Mustard"
  - ▷ "Miss Scarlett"
  - ▷ "Mrs. Peacock"
- $Room$ with 2 realizations:
  - ▷ "Dining room"
  - ▷ "Kitchen"
- $Weapon$ with 2 realizations:
  - ▷ "Dagger"
  - ▷ "Candlestick"

Clues distilled in the game induce belief in focal elements defined on some of these variables. Two players can have different and even contradictory informations that can be totally or partially combined. Actually, each player represents a belief potential.

6.2 Representing belief masses: usual approach v.s. *delayed mass valuation*

This section introduces an innovative point of view on the representation of belief masses. This principle called *delayed mass valuation* is based on practical observations and seems appealing when dealing with usual D-S applications. A comparison with the usual approach for representing belief masses and some empirical tests will help us to conclude on the practical feasibility of this new principle. Tests were implemented thanks to `eVidenZ` (see Chapter 10) and performed on a 2.4 Ghz Xeon.

6.2.1 Usual mass valuation

Most of the articles dealing with D-S theory consider belief masses as numerical values associated to focal elements. Mass functions are generally described as follows:

$$\begin{array}{cccc} m: & 2^{\Omega} & \rightarrow & [0,1]_{\Re} \\ & A & \rightarrow & v \end{array}$$

A mass function m associates a mass value v to a focal element A (subset of the frame of discernment Ω).

6.2.2 Combination and valuation: dissociated mechanisms

Value setup is theoretically dissociated from combination mechanisms. Given a problem, one must define the support of his belief, i.e. the subsets of the frame of discernment he has information about. In fact, one can build reasoning rules without valuating the exact belief. Combination mechanisms are mainly based on set intersections and give general rules for redistributing belief masses. These rules define how masses are multiplied (combination) or added (regrouping), but one must understand that the algorithm does not depend on the final mass values.

6.2.3 Drawbacks of masses as values: towards *delayed mass valuation* principle

Consider, for example, a toy application for medical diagnosis based on the D-S theory. The aim of this application could be to characterize a disease through series of questions targeting the symptoms:

- "Do you have headache?";
- "Do you have fever?";
- ...

The answers depend on the patient. They constitute the *context* that leads to the real belief valuation. In the usual approach, the belief masses are progressively valuated after each answer. This common attack leads to rebuilding the entire system for every patient, even if the problem, the focal elements and their combination remain the same.

Empirical test A simple test engine reveals the practical limitations of rebuilding the system for each *context*. Consider, for example, a system that discriminates among 100 diseases thanks to 20 questions. Building and interrogating such a system lasts about 0,02 seconds. This could seem acceptable, but consider now running such a system on 1 million patients. The total runtime becomes 5 hours and 33 minutes. If one hardly imagines running a medical diagnosis on 1 million patients, lots of real applications match such requirements. In the image processing field, some D-S applications, like satellite image segmentation (see Section B.1), require the treatment of every pixel of an image. Rebuilding the system for each pixel becomes abnormally constraining w.r.t. processing time.

In the previous subsection, we saw that we theoretically can build the medical reasoning system without having the answers of the patient. Doctors can establish general rules to define the focal elements corresponding to the questions. For example, the "fever" question has the aim to focus on or to eliminate some kinds of diseases. Combining the questions (i.e., their associated focal elements), we build the general system once, independently from the patient. This general system can now be applied on the patient's answers to valuate the final belief on every class of diseases. This principle can be called *delayed mass valuation*.

6.2.4 Another representation of belief functions

To achieve an engine implementation with the *delayed mass valuation* principle, we have to reconsider the usual representation of mass functions.

$$\begin{array}{rccc} m: & 2^{\Omega} \times E & \rightarrow & [0,1]_{\Re} \\ & (A, e) & \rightarrow & v \end{array}$$

A mass function m can now be viewed as a function that associates a mass value v to a focal element A, in a *context* e.

Implementation of "delayed" masses in OOP If one adopts the *delayed mass valuation* principle, one can not implement masses by *float* or *double* values anymore. In the object-oriented programmation framework, masses can be viewed as objects waiting for a *context* to return a floating value. Actually, this representation implies the use of *functors* (i.e. functions as objects, see (Coplien, 1992)) for belief masses.

6.2.5 A new D-S computation scheme

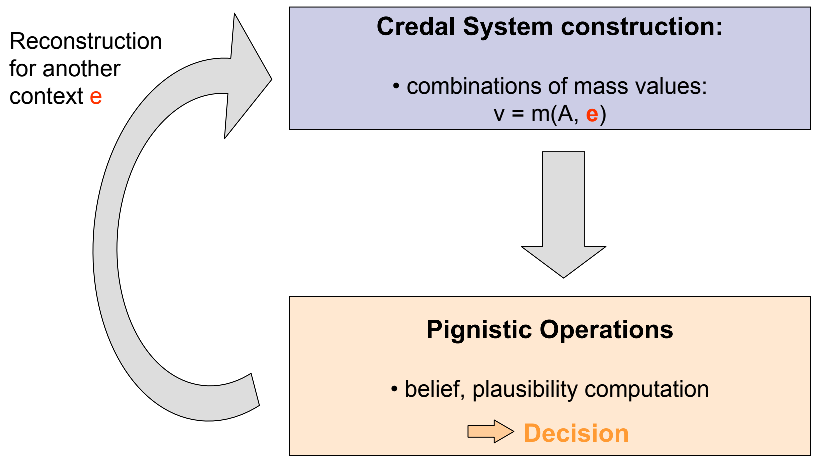
Reconstruction for another context e
Credal System construction:
• combinations of mass values:
v = m(A, e)
Pignistic Operations
• belief, plausibility computation
Decision

Figure 6.1: Usual model (TBM)

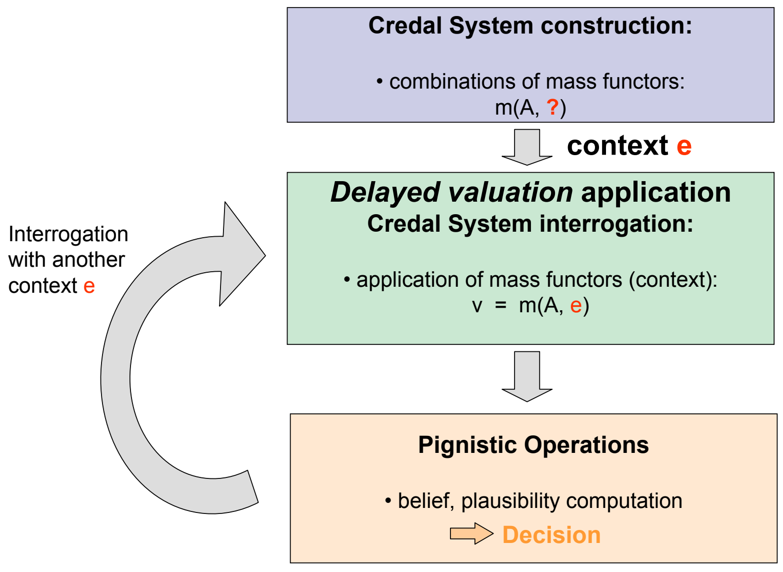
Credal System construction:
• combinations of mass functors:
m(A, ?)
context e
Delayed valuation application
Credal System interrogation:
• application of mass functors (context):
v = m(A, e)
Interrogation with another context e
Pignistic Operations
• belief, plausibility computation
Decision

Figure 6.2: *delayed mass valuation* model

Introducing the *delayed mass valuation* principle leads finally to a new modeling of the local computation scheme. This modeling, described in figure 6.2, can be viewed as an extension of the Transfer Belief Model (TBM, see Chapter 3), described by P. Smets in (Smets et al., 1991) and summed up in Figure 6.1.

This new modeling is composed of three main stages:

- credal system construction: pieces of evidence are combined without valuating the belief masses (i.e. without giving *context*);

- *delayed mass valuation* application: the *context* is given, belief masses are valuated, belief and plausibility interrogations become feasible;
- pignistic operations: processes that lead to decision making.

6.2.6 On *delayed mass valuation* (un)feasability and (un)applicability

The *delayed mass valuation* principle seems interesting for problems where the same reasoning system has to be applied to many final use cases. In practice, this potentially concerns lots of real D-S applications.

Performance gain

The main performance gain of *delayed mass valuation* is avoiding the reasoning system to be rebuilt for each new *context*. For lots of applications, like the ones we are interested in, the gain could be interesting. For cancerous cells characterization (see Appendix B.2), the system is built once and applied to thousands of cells. For satellite images segmentation (see Appendix B.1), the system is applied to thousands of different pixels.

But this interesting property also has important drawbacks:

- more complex potentials than necessary;
- incompatibility with almost all approximation methods;
- loss of performance due to functor indirections.

More complex potentials than necessary

The idea of the *delayed mass valuation* is basically to perform potential combinations without valuating the belief masses. But, as exposed in Subsection 3.2.1, focal elements are precisely the configuration sets with non-null belief masses. Performing combinations without knowing the mass values, it is possible to keep focal elements that will finally have null masses. Consider the medical diagnosis example proposed in this section. For such a general application, a relevant implementation would exclude lots of focal elements (i.e. disease classes) after getting some indubitable answers. Focal element number would be reduced, leading to “simple” potentials. Dealing with *delayed valuation*, we are not able to exclude the irrelevant focal elements. This unavoidable drawback can significantly increase the computation complexity for applications where focal elements are partially induced by the *context*. Note that for some applications, like satellite image segmentation (see Appendix B.1), no such problem occurs: focal elements are totally defined by the modeling and do not depend on any *context*.

delayed mass valuation and approximations incompatibility

Approximations are often used to improve computations performance (see Section 8.1). Unfortunately, almost all approximation methods require mass values to be available from the very beginning since they evaluate the relevance of focal elements thanks to their masses. The *delayed mass valuation* is unfortunately incompatible with the most promising approximation methods because it relies on building the reasoning system without valuating the belief masses. Note that some of the approximation methods, like the Bayesian one (see 8.1.1), do not rely on the belief masses. These kinds of approximations can thus be mixed with the *delayed mass valuation* principle.

The dark side of the *delayed mass valuation*: functor indirections

Manipulating belief masses as functors, the combination process induces the creation of lots of new functors by generating lots of new focal elements. Indeed, the masses of the new focal elements are expressed in function of the initial masses of their parents: it implies the creation of combination ("times") and regrouping ("plus") functors that link the initial masses. At the end of the construction of the credal system, we obtain thousands of functor indirections forming the evaluation tree of each mass. The evaluation time - i.e. the functor call with a given context - depends heavily on the number of indirections and can become practically problematic.

Note that the functors corresponding to initial masses can not be *inlined* since their addresses in memory rely on dynamic values, hash keys or balanced tree indexes, depending on the potential representation (see Section 6.4).

Empirical test Figure 6.2.6 illustrates the indirection problem by comparing usual and *delayed mass valuation* approaches. We consider a reasonable problem: the combination of 5 potentials with 10 initial focal elements of 10 configurations, defined on 1 variable of 10 realizations. The table shows the time needed for constructing and querying the final potential. The interrogation consists in valuating the belief mass of 100 focal elements of the final potential. In the case of *delayed mass valuation*, the system is built once and then interrogated 100 times. With the usual approach, the system is built and interrogated 100 times. For each interrogated focal element, we evaluate the average number of each type of indirection induced by the *delayed mass valuation*.

	delayed mass valuation	**Usual approach**
memory load	1 Mb	1 Mb
final focal elements	333	333
total construction time	0.02 second (1 time)	2 seconds (for 100 times)
total interrogation time (100 Focal elements)	28 hours	0.36 seconds
total run time	28 hours	2.36 seconds
average regrouping indirections	1.24×10^{10}	0
average combination indirections	2.25×10^{10}	0
average initial mass indirections	3.54×10^{10}	0

Figure 6.3: *delayed mass valuation* vs usual approach

One can guess that systems implemented with *delayed mass valuation* may become unusable because of this indirection problem.

Figures 6.4 and 6.5 illustrate the explosion of the indirection number and interrogation time with *delayed valuation* principle. These curves were obtained by combining potentials of 10 initial focal elements with 10 configurations, defined on one variable with 8 realizations. The indirection number grows exponentially with the combination number: the time needed for querying the system is quickly unacceptable. When one remembers that computing belief or plausibility values can induce thousands of mass valuations, there is no hope such a system could be practically usable.

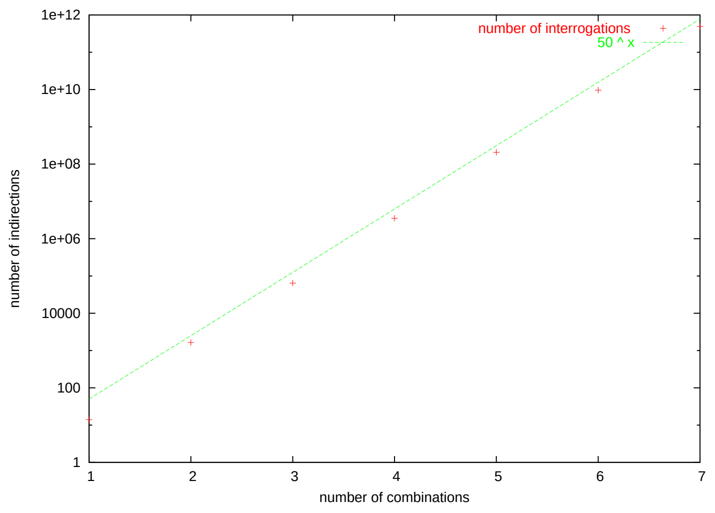

Figure 6.4: Indirection number

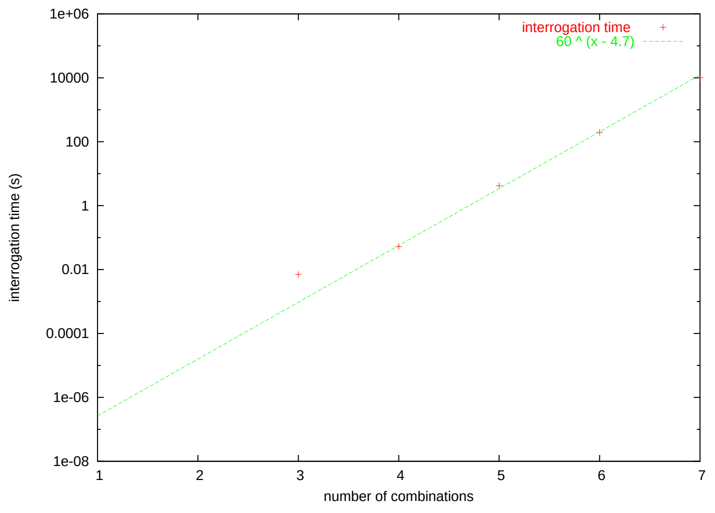

Figure 6.5: Interrogation time

6.2.7 Conclusion: usual approach winner by KO, but ...

A principle like *delayed mass valuation* could seem appealing when considering real D-S applications. However, this new computation scheme induces very strong drawbacks: potentials more complex than needed, incompatibility with the most interesting approximation methods and functor indirections. These drawbacks, especially the last one, make the *delayed mass valuation* unusable when it is implemented a naive way. Simple empirical tests prove that rebuilding the system for every use case is much more interesting than dealing with incredible indirection times.

Anyway, the original principles of *delayed mass valuation* remain interesting since the usual approach is not totally satisfactory. We are evaluating solutions like code generation to make D-S systems and partial *delayed mass valuation* compatible. Some ideas based on the combination of the classical approach and the *delayed mass valuation* are currently studied (see Section 11.2).

6.3 Representing focal elements

As it was already explained in Subsection 3.2.1, focal elements represent the core belief support. They are in fact sets of variable configurations.

These core structures are involved in every D-S mechanism. Projection and extension perform operations directly on the focal elements and combination and fusion manipulate them intensively. Then, one can guess that the encoding of these focal elements heavily affects the performance of the final computations. We can sum up the requirements for a good representation as follows:

- fast projection computation;
- fast extension computation;
- fast intersection computation;
- fast union computation;
- fast equality testing;
- limited memory size;
- independence from the initial representation of variables.

Of course, every candidate we will examine can fill the last requirement. Choosing the right encoding, it is generally easy to find a single internal representation for the variable configurations. For example, by using some kind of indexing, one can use only indexes, whenever variable configurations are initially floating values, strings or anything else...

$$\begin{aligned}
Murderer &= \{``ColonelMustard", ``MissScarlett", ``Mrs.Peacock"\} \\
``ColonelMustard" &\Leftrightarrow 0 \\
``MissScarlett" &\Leftrightarrow 1 \\
``Mrs.Peacock" &\Leftrightarrow 2
\end{aligned}$$

Figure 6.6: Variable configuration indexing

Figure 6.6 illustrates the indexing of the $Murderer$ configurations. Even if configurations are initially represented by strings, such an indexing allows us to use only integers to refer to configurations. So, the configuration 0 of the $Murderer$ variable unambiguously represents the $``Colonel$

$Mustard''$ configuration. Note that introducing indexing induces an ordering of variable configurations.

In the following subsections, we will use the example proposed in Section 6.1 and focus on the two variables $Murderer$ and $Weapon$.

6.3.1 List representation

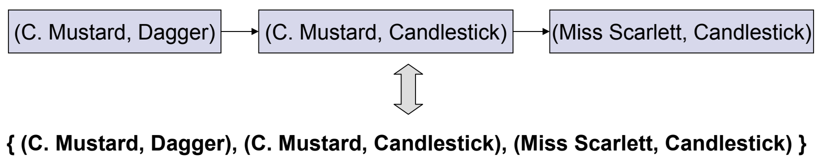
(C. Mustard, Dagger)
(C. Mustard, Candlestick)
(Miss Scarlett, Candlestick)
{ (C. Mustard, Dagger), (C. Mustard, Candlestick), (Miss Scarlett, Candlestick) }

Figure 6.7: list representation

The size of a list of configurations can be prohibitive. Besides, projection and extension algorithms on lists are inefficient: lists are not adapted to fast research, suppression and addition when dealing with huge sets. Then, set unions and intersections can not be performed a satisfying way. Actually, this naive approach does not really fill any of the requirements and thus can not be considered as a possible candidate.

6.3.2 Binary representation (bitsets)

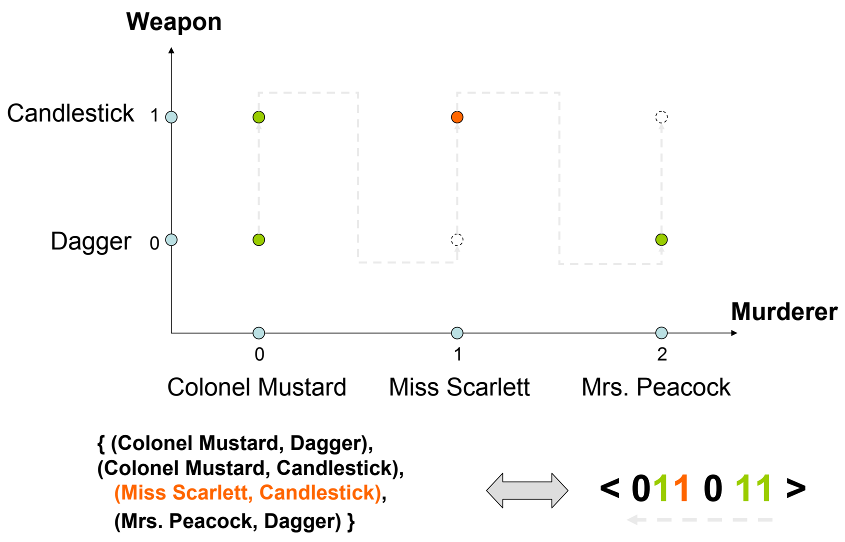

Figure 6.8: bitset representation

The bitset representation was already described in (Haenni and Lehmann, 2001), (Lehmann, 2001) and (Xu and Kennes, 1994). The underlying idea is a global ordering of the variables involved. A focal element, as a set of configurations, is represented by a bitset. Each bit of the bitset corresponds to a configuration: if the bit is set to 1, the configuration is present in the focal element.

Consequently, all bitsets on a fixed frame of discernment have the same size, S bits, given by the following formula:

$$S = \prod_{i=1}^{n} S_i \tag{6.1}$$

where n is the number of variables and S_i is the number of realizations of the variable i minus 1 (maximal realization index).

Consider a configuration $c = (r_1, ..., r_n)$, where r_k is a realization of the variable k. In a bitset representing a focal element defined on variables $\{1, ..., n\}$, c's index is directly given by:

$$I_c = \sum_{i=1}^{n} r_i \prod_{j=i+1}^{n} S_j \tag{6.2}$$

Note that all indexes begin at 0 and that bitset indexing is little endian.

Figure 6.8 gives an example of this representation. $Miss\ Scarlett$ has the index $I_{MissScarlett} = 1$ in the $Murderer$ realization set. $Candlestick$ has the index $I_{Candlestick} = 1$ in the $Weapon$ realization set. $Murderer$ is considered as the first variable. The index of the configuration $(MissScarlett, Candlestick)$ is given by:

$$I_{(MissScarlett,Candlestick)} = I_{MissScarlett} \times S_{Weapon} + I_{Candlestick} = 1 \times 1 + 1 = 2 \tag{6.3}$$

Benefits

Basic operations on bitsets are very efficient and can be performed by really simple operators:

- intersections are performed thanks to logical and;
- unions are performed thanks to logical inclusive or;
- projections and extensions are performed thanks to bit extractions (bit masks) and bit blocks shifts.

Drawbacks

As it was already explained, the size of the focal elements is constant and independent from the number of configurations. This size can grow heavily with variable number (and corresponding realizations number) (formula 6.1). For example, a focal element on a domain composed of 30 binary variables would be stored with a bitset of 2^{30} bits (128 Mb). Besides, one can note that this size does not depend on the number of configurations in the focal element. Consider a focal element defined on the same domain of 30 binary variables. Even if this focal element contains a single configuration, its size would be 128 Mb.

Anyway, (Haenni and Lehmann, 2001) explains that this size constraint is not too restrictive in practice, under the limit of 12 binary variables (i.e. bitsets of 4 Kb). This boundary is actually rarely exceeded.

6.3.3 Alternative representations: logical normal forms

Logical representations are very interesting alternatives to bitsets. Considering variables as propositions, the basic idea is to represent a focal element by a logical formula. Note that dealing with binary variables, propositional logic is sufficient for handling these representations.

Disjunctive normal forms

$$FE = \bigcup_i \bigcap_j S^i_j with S^i_j \in \Theta_{x_j} \tag{6.4}$$

Focal elements are represented by unions of set intersections, like explained in Equation 6.4. Figure 6.9 gives an example of a focal element under a disjunctive normal form (same focal element than figure 6.8):

$$FE = (\{Mustard, Scarlett\} \bigcap \{Candlestick\}) \bigcup (\{Mustard, Peacock\} \bigcap \{Dagger\})$$

Figure 6.9: Example of a disjunctive normal form

Conjunctive normal forms

$$FE = \bigcap_i \bigcup_j S^i_j with S^i_j \in \Theta_{x_j} \tag{6.5}$$

Focal elements are represented by intersections of set unions, like explained in Equation 6.5. Figure 6.10 gives an example of a focal element under a conjunctive normal form (same focal element than figure 6.8):

$$FE = (\{Mustard\} \bigcup \{Dagger, Candlestick\}) \bigcap (\{Scarlett\} \bigcup \{Candlestick\}) \bigcap (\{Peacock\} \bigcup \{Dagger\})$$

Figure 6.10: Example of a conjunctive normal form

Normal form implementation

Normal forms can be implemented efficiently by n-ary trees. Figures 6.11 and 6.12, based on examples 6.9 and 6.10 respectively, illustrate this implementation.

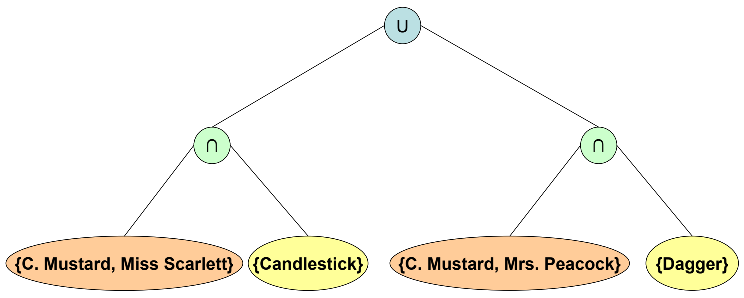

Figure 6.11: N-ary tree for disjunctive form

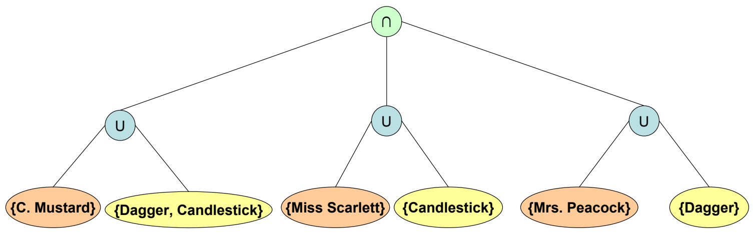

Figure 6.12: N-ary tree for conjunctive form

Benefits

All logical mechanisms profit these logical forms. The main benefit is the short representation produced, slightly smaller than bitset one. A study of the size needed by these representations can be found in (Lehmann, 2001). This size quality becomes really interesting when dealing with huge frames of discernment.

Drawbacks

Unfortunately, some operations, like equality test and set intersection, remain difficult (Haenni and Lehmann, 2001), (Lehmann, 2001).

6.3.4 Conclusion on focal element representation

This section presented the most usual representations for focal elements. Lists of configurations do not seem acceptable for performance and memory size reasons. The two advanced data structures proposed, bitsets and normal forms, have different but interesting properties. Set operations are really efficient with bitsets, but the size of the structures involved can become too important. Normal forms provide a short representation that is essential for huge focal elements, but the operations on these forms are less efficient. Even if normal forms are the only structure acceptable for huge focal elements, bitsets are in practice almost always relevant.

The bitset representation is certainly the most efficient representation for focal elements. Bitsets are thus adopted in our engine, `eVidenZ` (see Chapter 10). In the following parts of this report, focal elements are assumed to be represented by bitsets. Sections 7.3 and 7.2 propose efficient implementations of extension and projection based on this representation. The next section introduces different representations of potentials that also rely on bitsets.

6.4 Representing potentials

Potentials are global sources of information, the most high-level structures manipulated in the D-S framework. They couple focal elements and their associated belief masses. (see 3.2.1). Basically, a potential can be seen as a collection of pairs (F_i, m_i) with F_i a focal element and m_i a belief mass.

As for focal elements (see 6.3), the encoding of the potentials heavily influences the computation efficiency. The operations performed on potentials are combination, fusion and marginalization. Computing combination, lots of intersections are generated and lots of them are usually equal (Lehmann, 2001). Then, focal element matching and regrouping procedures are good clues for measuring the efficiency of a representation. We can sum up the requirements as follows:

- fast focal element matching and regrouping;
- limited memory size.

In the following subsections, we consider that focal elements are implemented thanks to the binary representation exposed in 6.3.2. We will reuse the example proposed at the beginning of this chapter and consider potentials defined on the variable $Murderer = \{Colonel\ Mustard, Miss\ Scarlett, Mrs.\ Peacock\}$.

6.4.1 List representation

Potentials can naturally be implemented by lists of pairs (F_i, m_i). This representation is unfortunately inefficient: the research procedure is performed in time $O(n)$ and regrouping in time $O(n^2)$. A slightly better representation is to consider sorted lists. This leads directly to the balanced trees representation.

6.4.2 Balanced tree representation

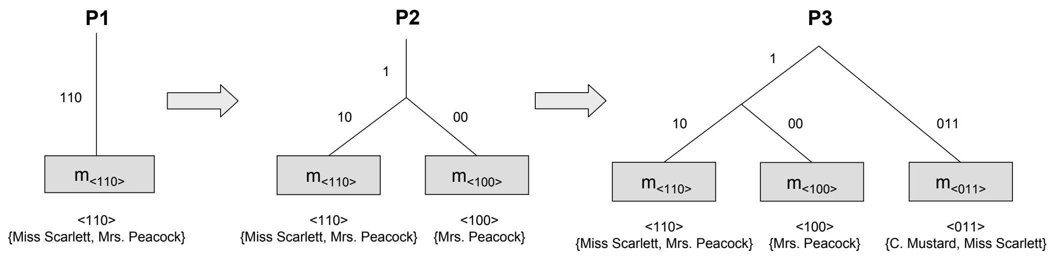

Figure 6.13: balanced tree representation

As described in (Lehmann, 2001), if the binary representation is used for the focal elements, then the branching of the tree is determined by the bitstrings of the focal sets. Every branch of the binary tree is labeled by a bitstring. The concatenation of all bitstrings on the path from the root to a leaf is the binary representation of the focal set. The leaf stores the corresponding mass. Figure 6.13 gives an example of the progressive construction of a potential with a balanced tree.

Using balanced trees, the research procedure can be performed in time $O(log(n))$ and regrouping in time $O(n \times log(n))$.

Benefits

In (Lehmann, 2001) and (Haenni and Lehmann, 2001), Lehmann and Haenni explain that balanced trees are very efficient for regrouping, combination and marginalization. They are one of the most relevant representation for huge potentials.

Figure 6.4.2 presents the computation times needed for combining 2 potentials P_1 and P_2 of 1000 focal elements with 1000 configurations, whether potentials are implemented by lists or balanced trees. This test was implemented thanks to `eVidenZ` (see Chapter 10) and performed on a 2.4 Ghz Xeon with 1 Gb Ram.

Potential representation	$P_1 \otimes P_2$ **computation time**
balanced trees (AVL)	6 seconds
lists	14 hours 22 minutes

Figure 6.14: Comparison between AVL and list representations

Drawbacks

Maintaining a balanced tree has an important cost, especially when dealing with extensions, which imply lots of modifications and insertions. In (Lehmann, 2001), Lehmann recommends using this representation only for local computations in huge potentials.

6.4.3 Hash table representation

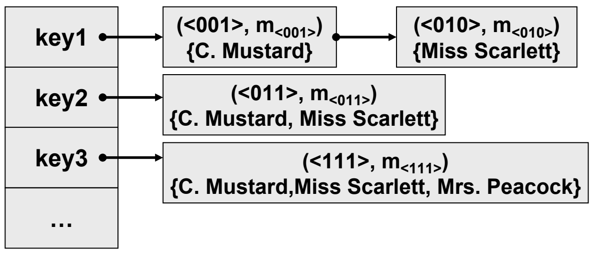
key1
key2
key3
...
(<001>, m<001>) {C. Mustard}
(<010>, m<010>) {Miss Scarlett}
(<011>, m<011>) {C. Mustard, Miss Scarlett}
(<111>, m<111>) {C. Mustard,Miss Scarlett, Mrs. Peacock}

Figure 6.15: hash table representation

The key associated to each focal element is computed using the corresponding bitset value. The table stores the masses associated to these keys. Figure 6.15 gives an example of a potential stored in a hash table.

Using hash tables, the research procedure can theoretically be performed in time $O(1)$ and regrouping in time O(n^2/s), where s is the size of the table.

Benefits

In average, research in hash tables is very efficient. In (Haenni and Lehmann, 2001) and (Lehmann, 2001), theoretical and empirical studies show that hash tables are in average 30% more efficient than balanced trees for reduced potentials.

Drawbacks

Hash tables are efficient at the condition of being large enough. Adding elements in a too small table causes many collisions that make the research much less efficient. One solution is then to enlarge the table, but the cost of recopying a table is heavy. Thus, hash tables are not really adapted to huge potentials.

6.4.4 Conclusion on potential representation

This section proposed two interesting data structures for representing potentials: balanced trees and hash tables. If balanced trees are theoretically the most adapted representation, hash tables are often more efficient in practice. In order to avoid the limitation of the hash table size, some researchers, like Lehmann in (Lehmann, 2001), proposed hybrid solutions. The idea is then to swap from hash table to balanced tree when the latter become favorable, i.e. when there occurs too many collisions in the hash table.

Chapter 7

Efficient D-S core mechanisms

In the previous chapter, we saw how the structures involved in D-S computations can be implemented efficiently. We will now focus on the core algorithms that manipulate these structures. We will explain how the theoretical algorithms, projection, extension, marginalization, combination and fusion can be improved or at least implemented efficiently.

7.1 Prerequisites

This chapter will first give efficient implementations for projection and extension algorithms. These implementations are taken from (Haenni and Lehmann, 2001) and are based on the binary representation of focal elements (see 6.3.2), so their efficiency is ensured by intensive bit string operations.

For better comprehension, let introduce some notations:

- $F^{\downarrow C}$: projection of focal element F on the domain C;
- $F^{\uparrow C}$: extension of focal element F on the domain C;
- $logior(B_1, B_2)$: bitwise logical inclusive or;
- $lsl(B, n)$: logical shift to the left (n bits, fill right-most bits with 0's);
- $lsr(B, n)$: logical shift to the right (n bits);
- $extract(B, n, pos)$: extracts n bits from B, starting at position pos;
- $deposit(B, n, pos, B')$: replaces n bits at position pos of B by the first n bits of B'.

7.2 Projection implementation

We describe here the projection implementation proposed in (Haenni and Lehmann, 2001). As explained in Subsection 6.3.2, we assume a global ordering of the variables. The focal element F to project is defined on a variable domain D. The main idea is to consider the projection on the domain C as an iterative process eliminating sequentially the variables in $D \backslash C$. This is valid because of the transitivity of the projection.

$$F \Rightarrow F^{\downarrow D-\{x_{i_1}\}} \Rightarrow F^{\downarrow D-\{x_{i_1},x_{i_2}\}} \Rightarrow ... \Rightarrow F^{\downarrow C} \qquad (7.1)$$

7.2.1 Simple projection

The procedure $simple\_projection(F, x_i)$ projects the focal element F, defined on D, on the domain $D - \{x_i\}$. To perform this variable elimination efficiently, the algorithm presented in Algorithm 1 is based on bit extraction and block shifting. Considering the variable x_i to be eliminated, the initial domain D is viewed as follows:

$$D = \{\underbrace{x_k, x_{k+1}, ..., x_{i-1}}_{L_{D(x_i)}}, x_i, \underbrace{x_{i+1} ... x_n}_{R_{D(x_i)}}\}$$

We will note $u_D(x_i) = |\Theta_{L_{D(x_i)}}|$, the number of configurations of F on the left sub-domain. The same way, $v_D(x_i) = |\Theta_{R_{D(x_i)}}|$ corresponds to the number of configurations of F on the right sub-domain.

Algorithm 1 simple_projection(F, x_i)

```
1: $R := \{\}$
2: $u := u_D(x_i)$; $v := v_D(x_i)$;
3: for $k$ from 1 to $|\Theta_{x_i}| - 1$ do
4:     $F := logior(F, lsr(F, v))$;
5: end for
6: for $k$ from 0 to $u - 1$ do
7:     $E := extract(F, v, k \times |\Theta_{x_i}| \times v)$;
8:     $R := deposit(R, v, k \times v, E)$;
9: end for
10: return$R$;
```

Complexity in the worst case: $O(n^2)$ where n is the size of the bitset.

The first loop, on lines 3 to 4, is a conditioning of the bitset F. Successively, F's bits are positioned on the right thanks to $lsr(F, v)$ and *logical inclusive or*, $logior(F, lsr(F, v))$. At the end of this loop, significant bit blocks are regrouped on the right side of the bitset. These particular bits present the configurations on the variables $D - \{x_i\}$ in the result focal element. This operation avoids searching for fragments of configurations in the following part of the algorithm.

The second loop, starting at line 6, is dedicated to extracting these significant bits. The bits are then deposited in the result bitset R. We finally return this bitset $R = F^{\downarrow D-\{x_i\}}$.

This algorithm is particularly efficient if $u_{D(x_i)}$ and $|\Theta_{x_i}|$ are relatively small, i.e. if the variable x_i to be eliminated has a small index in D and a small set of configurations.

7.2.2 Projection

The $projection$ procedure is described in Algorithm 2. It projects the focal element F on the domain C. It is actually a sequential call to $simple\_projection$ to eliminate successively the variables in $D \backslash C$.

Algorithm 2 projection(F, C)

```
1: for each $x_i \in D \backslash C$ do
2:     $F := simple\_projection(F, x_i)$;
3: end for
4: return$F$;
```

Complexity: $O(n)$ where n is the number of variables to eliminate.

The previous paragraph showed that the efficiency of the $simple\_projection$ procedure depends on the rank of the variable x_i to be eliminated. In (Haenni and Lehmann, 2001), Haenni and Lehmann propose to select iteratively the variable with the smallest index in D.

7.3 Extension implementation

We describe here the extension implementation proposed in (Haenni and Lehmann, 2001). It is basically the projection reversed operation (see Section 7.2). The focal element F to extend is defined on a variable domain C. The main idea is to consider the extension on the domain D as an iterative process adding sequentially the variables in $D \backslash C$.

$$F \Rightarrow F^{\uparrow C \bigcup \{x_{i_1}\}} \Rightarrow F^{\uparrow C \bigcup \{x_{i_1}, x_{i_2}\}} \Rightarrow ... \Rightarrow F^{\uparrow D} \tag{7.2}$$

7.3.1 Simple extension

The procedure $simple\_extension(F, x_i)$ extends the focal element F, defined on C, on the domain $C \bigcup \{x_i\}$. This procedure is presented in Algorithm 3. Considering the variable x_i to be added, the final domain D is viewed as follows:

$$D = \{\underbrace{x_k, x_{k+1}, ..., x_{i-1}}_{L_{D(x_i)}}, x_i, \underbrace{x_{i+1} ... x_n}_{R_{D(x_i)}}\}$$

We will note $u_D(x_i) = |\Theta_{L_{D(x_i)}}|$, the number of configurations of F on the left sub-domain. The same way, $v_D(x_i) = |\Theta_{R_{D(x_i)}}|$ corresponds to the number of configurations of F on the right sub-domain.

Algorithm 3 simple_extension(B, x_i)

```
1: $R := \{\}$
2: $u := u_D(x_i)$; $v := v_D(x_i)$;
3: for $k$ from 0 to $u - 1$ do
4:     $E := extract(B, v, k \times v)$;
5:     $R := deposit(R, v, k \times |\Theta_{x_i}| \times v, E)$;
6: end for
7: for $k$ from 1 to $|\Theta_{x_i}| - 1$ do
8:     $R := logior(R, lsl(R, v))$;
9: end for
10: return$R$;
```

Complexity in the worst case: $O(n^2)$ where n is the size of the bitset.

In the first loop starting at line 3, bit blocks of B are extracted and deposited in the result bitset R. At this temporary state, R contains the old configurations of B (configurations on C), ready to be extended on $C \bigcup \{x_i\}$. Because these bit blocks were deposited at known positions, the extension on the new variable will be performed efficiently.

In the second loop starting at line 7, the deposited bit blocks are extended thanks to *logical shifts* and *logical or* on R. Semantically, all the configurations are extended on the new variable x_i and placed at the right position. We finally obtain the bitset $R = F^{\uparrow C \bigcup \{x_i\}}$.

The same way than for the $simple\_projection$ procedure, this algorithm is particularly efficient if the variable x_i to be added has a small index in D and a small set of configurations.

7.3.2 Extension

The $extension$ procedure is described in Algorithm 4. It extends the focal element F on the domain D. It is actually a sequential call to $simple\_extension$ to add successively the variables in $D \backslash C$.

Algorithm 4 extension(B, D)

```
1: for each $x_i \in D \backslash C$ do
2:     $B := simple\_extension(B, x_i);$
3: end for
4: return$B;$
```

Complexity: $O(n)$ where n is the number of variables to add.

The previous paragraph showed that the efficiency of the $simple\_extension$ procedure depends on the rank of the variable x_i to be added. In (Haenni and Lehmann, 2001), Haenni and Lehmann propose to select iteratively the variable with the highest index in D.

7.4 Combination implementation

Sections 3.2.2 and 3.3.3 already introduced the theoretical definition of the combination mechanism. We will now propose a simple algorithm that can be directly implemented.

Before presenting the combination algorithm, we need to define regrouping and normalization procedures already introduced in Sections 3.3.2 and 3.2.2 respectively.

7.4.1 Regrouping

As already exposed in the section 6.4, an efficient $regrouping$ procedure is primordial in order to regroup the equal sets and sum up their mass values during combination. The final efficiency is heavily influenced by the representation of potentials, as described in the section 6.4. Algorithm 6 describes the basic $regrouping$ procedure. The key step is the $update(P, \{(F, m)\})$ procedure (line 3) that looks in the potential P for an existing focal element equal to F. If a such a focal element, $F' = F$, exists, then the belief mass m' associated to F' is replaced by $m' + m$. If P contains no focal element equal to F, the couple (F, m) is added to P. Pseudo-algorithm 5 sums up $update$ procedure.

Algorithm 5 update($P, \{(F, m)\}$)

```
1: if there exists $(F', m') \in P$ such that $F' = F$ then
2:     $\{F', m'\} := \{F', m + m'\}$;
3: else {}
4:     $P := P \bigcup \{(F, m)\}$;
5: end if
```

Complexity depends on the representation of potentials (see Section 6.4).

Algorithm 6 regrouping(P)

```
1: $R := \{\}$
2: for each $(F, m) \in P$ do
3:     $update(R, \{(F, m)\})$;
4: end for
5: return$R$;
```

Complexity depends on the representation of potentials. Example with balanced trees (see 6.13): O($n \times log(n)$) where n is the number of P's focal elements.

7.4.2 Normalization

The procedure $normalization(P)$ normalizes the potential P according to the closed world assumption (see Section 1.4). It redistributes the conflict mass associated to the empty focal element after empty intersections during combination. The pseudo-algorithm 7 can be described as follows:

- first, the procedure looks for the empty focal element;
- if the empty focal element is present and has a non-null belief mass:
  - ▷ the mass of every non-null focal element is updated: it is divided by $\frac{m}{1-m_\emptyset}$;
  - ▷ the empty focal element is deleted.

Algorithm 7 normalization(P)

```
1: if there exists $(\emptyset, m_\emptyset) \in P$ such that $m_\emptyset \neq 0$ then
2:     for each $(F, m) \in P$ such that $F \neq \emptyset$ do
3:         $(F, m) := (F, \frac{m}{1-m_\emptyset})$;
4:     end for
5:     $P := P - \{(\emptyset, m_\emptyset)\}$;
6: end if
```

Complexity depends on the representation of potentials. Example with balanced trees (see 6.13): O($n \times log(n)$) where n is the number of P's focal elements.

7.4.3 Combination

The algorithm for combining two potentials, P_1 and P_2, as described in (Haenni and Lehmann, 2001), is composed of four steps:

1. P_1 and P_2 focal elements are extended to $D = D_1 \bigcup D_2$;
2. extended focal elements of P_1 and extended focal elements of P_2 are intersected. Their respective belief masses are multiplied and associated to the intersections;
3. equal intersections are regrouped and their respective masses are summed up;
4. the resulting potential is normalized.

In the implementation proposed in Algorithm 8, steps 2 and 3 are regrouped thanks to the $update$ procedure at line 7.

Algorithm 8 combination(P_1, P_2)

```
1: for each $(F, m) \in P_1 \bigcup P_2$ do
2:     $F := F^{\uparrow D}$;
3: end for
4: $R := \{\}$;
5: for each $(F_1, m_1) \in P_1$ do
6:     for each $(F_2, m_2) \in P_2$ do
7:         $update(R, \{(F_1 \bigcap F_2, m_1 \times m_2)\})$;
8:     end for
9: end for
10: $R = normalization(R)$
11: return$R$;
```

Complexity depends on the representation of potentials. Example with balanced trees: O($n \times m \times log(n \times m)$) where n and m are numbers of P_1's and P_2's focal elements respectively.

7.5 Marginalization implementation

The marginalization of a potential P on a domain C is basically the projection of P's focal elements on C. In the algorithm proposed (Algorithm 9), note that regrouping is also done incrementally at line 3.

Algorithm 9 marginalization(P, C)

```
1: for each $(F, m) \in P$ do
2:     $F = F^{\downarrow C}$;
3:     $update(P, \{(F, m)\})$;
4: end for
5: return$P$;
```

Complexity depends on the representation of potentials. Example with balanced trees: O($n \times log(n)$) where n is the number of P's focal elements.

7.6 Fusion implementation

The fusion of two potentials P_1 and P_2 on a domain C is equivalent to the combination of P_1 and P_2 followed by a marginalization on C. In the algorithm proposed (Algorithm 10), regrouping is once again done incrementally.

Algorithm 10 fusion(P_1, P_2, C)

```
1: for each $(F, m) \in P_1 \bigcup P_2$ do
2:     $F := F^{\uparrow D}$;
3: end for
4: $R := \{\}$;
5: for each $(F_1, m_1) \in P_1$ do
6:     for each $(F_2, m_2) \in P_2$ do
7:         $update(R, \{((F_1 \bigcap F_2)^{\downarrow C}, m_1 \times m_2)\})$;
8:     end for
9: end for
10: $R = normalization(R)$
11: return$R$;
```

Complexity: depends on the representation of potentials. Example with balanced trees: O($n \times m \times log(n \times m)$) where n and m are numbers of P_1's and P_2's focal elements respectively.

7.7 Conclusion on the algorithms proposed

Both projection and extension algorithms we described rely on the bitset representation of the focal elements presented in the section 6.3.2. In fact, these algorithms do not really consider focal elements as sets. They do not consider each configuration independently and rather deal with block indexes. Their efficiency is thus based on bit string operations like *logical and*, *logical or* and *bit block shifts*. Such algorithms are valuable candidates for D-S core mechanisms, where computation speed is essential.

The combination, marginalization and fusion algorithms we proposed are simple but efficient algorithms that are all very close to their theoretical definitions. They introduce considerations like *regrouping* and *normalization*. They finally propose basic improvements like incremental regrouping. For the $combination$ and the $fusion$ mechanisms, a significant improvement can be obtained by memoizing projections and intersections thanks to hash tables citephl2001a, (Xu and Kennes, 1994). Indeed, dealing with huge systems composed of lots of focal elements, identical projections and intersections are usually computed several time. Caching return values has proved to divide computation time by two to three (Haenni and Lehmann, 2001).

Alternative algorithms for marginalization and fusion were introduced in (Haenni and Lehmann, 2001). Haenni and Lehmann propose to replace projection and extension algorithms by so-called *quasi-projection* and *quasi-extension*. The idea is to avoid bit extractions and deposits by performing only the pretreatment operations on the bitset during the sequential elimination or adding. Only the final bitset has to be effectively projected or extended. Haenni and Lehmann expose promising performance results. Nevertheless, our personal experiments were disappointing: using these techniques, we did not note any gain, rather a small performance loss.

Chapter 8

Advanced optimizations

In the two previous sections, we saw relevant structures and algorithms to implement D-S theoretical bases. These techniques are sensibly better than naive approaches, but they do not really increase D-S feasibility. They just do not worsen the computational complexity. To really make D-S formalism feasible, researchers have developed parallel heuristics. This section is dedicated to an overview of these methods. We will introduce approximations, fast Moebius transform and hypertree optimizations.

8.1 Approximation methods

As it was already explained, combining pieces of evidence with D-S theory can become quickly infeasible. To overcome these computational difficulties, various approximation algorithms have naturally been suggested. These methods necessarily induce biases in the theoretical bases. Besides, some of them result in the loss of important properties. The choice of an approximation method must be made as a tradeoff between runtime performance and accuracy. Another point to take into account is the time necessary to compute the approximated value: some approximation methods need heavy computations to be performed and so balance the other benefits.

This section will not make an exhaustive overview of these approximation methods. Empirical and theoretical studies already exist. Our aim is rather to introduce the general principles of the usual approximations and to give pointers to relevant papers.

The number of focal elements in the potentials heavily influences the computational complexity of combining pieces of evidence. As a consequence, most of the approximation algorithms tend to reduce this number. The main principle is to remove focal elements and to redistribute the corresponding numerical values. To do so, the methods proposed define various criteria to value the influence and the relevance of each focal element.

8.1.1 Bayesian approximation

This method, introduced in (Voorbraak, 1989), reduces the belief potential to a discrete probability distribution, i.e. focal elements are reduced to singletons. This approximation method has the property of limiting the focal elements number. Potentials can have a maximum of Θ focal elements, where Θ is the cardinal of the frame of discernment.

This method is the only one with improving approximations after several combinations. To understand this property, one must realize that combination mechanisms generate set intersections so that resulting focal elements get progressively closer to singletons. After some combinations, we can reasonably approximate a potential by the Bayesian method.

Bayesian approximation can thus become a really accurate approximation, with important simplifications of the potentials, but with relatively heavy intrinsic computation cost. Moreover, these good results have an important drawback: by limiting focal elements to singletons, the Bayesian approximation implies the incapability of dealing with partial ignorance.

8.1.2 Summarization, k-l-x and D1 approximations

These methods were introduced in (Lowrance et al., 1986), (Tessem, 1993) and (Bauer, 1996) respectively. Their principle is to incorporate only the highest valued focal elements into the approximation.

These methods propose different tradeoffs among the accuracy, the time needed for computing the approximated potential and the final number of focal elements. The D1 approximation is particularly interesting because it lets the user the choice of the final number of focal elements. This can lead to good tradeoffs between runtime performance (i.e. number of focal elements) and approximation quality (errors). It is probably the most adaptive method so far.

8.1.3 Probabilistic methods

Other methods, dealing with *Monte-Carlo algorithms*, were suggested in (Wilson, 1991) and (Kreinovich et al., 1994). These algorithms are based on the principle that initial belief masses are usually given by experts or other information sources we have reasonable doubt about. There is a non-null probability p_0 that the results obtained are totally irrelevant, whatever the way we compute them. This doubt can not be excluded totally and $1 - p_0$ represents the reliability of our information source.

By exploiting this imperfect reliability, Kreinovich proposes in (Kreinovich et al., 1994) several polynomial *Monte-Carlo* algorithms to compute D-S-like combinations. These computations are made with a chosen precision, so that the final reliability of the probabilistic D-S-like computation is close to p_0. One should note that we mentioned "D-S-like" computations because these probabilistic approaches use special combination rules, introduced in (Yager, 1987) and (Smets, 1998). These rules, simpler than Dempster's one, are mixed with *Monte-Carlo* approaches to make this formalism feasible (Kreinovich et al., 1994).

Performance obtained is interesting and, according to (Kreinovich et al., 1994), these approximations are not penalizing for huge systems. Anyway, without such methods, these computations simply could not be performed. If one can ideologically accept the alteration of the theoretical bases, these approaches are the rare ones to make really big combinations feasible.

8.1.4 Existing studies on approximation methods

It is difficult to make a fair comparison of the various approximation algorithms. One has first to define an error measure. The runtime gain and the complexity of computing the approximation itself also have to be taken into account. Several studies discuss the appropriateness of usual approximations. The empirical study presented in (Tessem, 1993) focuses on the quality of the approximation by considering the numerical deviation induced. Another study, in (Bauer, 1996), considers both quantitative and qualitative aspects. The validity and appropriateness of each approximation method is discussed through these two points of view. In (Lee and Zhu, 1995), Lee and Zhu give a more high-level overview of approximation approaches by presenting the principal trends and their respective creators.

8.2 Fast Moebius transformation

This approach to speed up Dempster's combinations relies on two important facts. First, theoretical studies showed that combinations can be performed thanks to a generalized Fourier transformation. Second, efficient algorithms for performing this transformation were proposed.

8.2.1 A generalized Fourier transform to compute combinations

In (Shafer, 1976), Shafer proved that there is a representation of belief masses in which Dempster's rule of combination is turned into a pointwise product. This representation is the commonality function introduced in Section 3.2.1. The transformation $m \rightarrow Com$ is called the Moebius transformation (Kennes and Smets, 1990). Thoma proved in (Thoma, 1991) that the Moebius transformation is in fact a generalized Fourier transformation.

8.2.2 Fast algorithms to compute the transformations

The second fundamental point is that there exists a fast Moebius transformation that can be viewed as a fast Fourier transformation generalization (Thoma, 1991), (Kennes and Smets, 1990). The algorithms to compute Moebius transformation and the inverse transformation an optimal way are proposed in (Kennes and Smets, 1990) and (Xu and Kennes, 1994).

8.2.3 Fast Moebius scheme for combination

In order to compute the combination of two potentials P_1 and P_2 (i.e. the combination of the associated belief function m_1 and m_2), one can use the following scheme:

- compute the commonality functions Com_1 and Com_2 using the fast Moebius algorithm;
- multiply Com_1 and Com_2 pointwise;
- compute the belief function corresponding to $m_1 \otimes m_2$ using the fast inverse Moebius transformation.

This scheme is summed up in Figure 8.1.

$$\begin{tikzcd}
(m_1, m_2) \arrow[r] \arrow[d] & (Com_1, Com_2) \arrow[d] \\
m_1 \otimes m_2 & Com_1 \cdot Com_2 \arrow[l]
\end{tikzcd}$$

Figure 8.1: Computing potential combinations with fast Moebius transformation

This method appears to speed up computations a significant way. For more details and results, please refer to theoretical and empirical studies published in (Kennes and Smets, 1990) and (Thoma, 1991).

8.3 Hypertree optimizations

Hypertrees, a.k.a clique trees (Lauritzen and Spiegelhalter, 1988) or binary join trees (Shenoy, 1997), are advanced algorithmic structures that are used to model belief transfers in huge D-S systems. Hypertrees are directly linked to another representation called Markov trees. Using one of these two structures, one can build a local message-passing scheme that propagates the belief functions among the network. Advanced propagation-based architectures rely on hypertrees or Markov trees and are successfully applied to D-S formalism (see Section 9). Hypertrees or Markov trees are mainly used to optimize the combination sequences thanks to graph and tree manipulations.

8.3.1 Hypergraph, hypertrees and Markov trees

Hypergraph A graph structure G is a pair $G = (V, E)$ where $V = \{v_1, ..., v_n\}$ is a set of vertices and $E = \{E_1, ..., E_k\}$ a set of edges $E_i \subseteq V$. If edges are pairs of vertices, then the graph is a simple graph. Considering edges as arbitrary sets of vertices, the graph is called a *hypergraph*, and edges are *hyperedges*. In the evidential context, V corresponds to the global variable domain. Hyperedges are subsets of D corresponding to the local domains of the belief potentials involved.

Hypertrees Hypertrees are acyclic hypergraphs. An interesting property is that it is always possible to find a covering hypertree $T = (V, E')$ for a hypergraph $H = (V, E)$ (Shafer et al., 1987). T covers H if, for every hyperedge $E_i \in E$, one can find a corresponding hyperedge $E'_i \in E'$ so that $E_i \subseteq E'_i$.

Markov trees Markov trees can be viewed as the hypertree symmetric representation. Indeed, the construction of a Markov tree from a hypertree is immediate (Xu and Kennes, 1994). The hyperedges of the hypertree are basically the nodes of the Markov tree. The main property of this structure is that when a variable x belongs to two distincts nodes, then every node on the path between these two nodes contains x.

Hypergraph, hypertree and Markov tree example: The hypergraph H on the left side of Figure 8.2 corresponds to the set of potentials $S = \{P_1, ..., P_5\}$ where P_1, ..., P_5 are defined on domains $\{a, b\}$, $\{b, c\}$, $\{b, d\}$, $\{c, d, e\}$ and $\{e\}$ respectively. In the middle of Figure 8.2, we propose a covering hypertree T associated to H. Cycles in H are eliminated by combining P_2 and P_3, initially defined on $\{b, c\}$ and $\{b, d\}$ respectively. $P_2 \otimes P_3$ is then defined on $\{b, c, d\}$, and cycles disappear. The Markov tree M on the right side of Figure 8.2 is associated to the hypertree T.

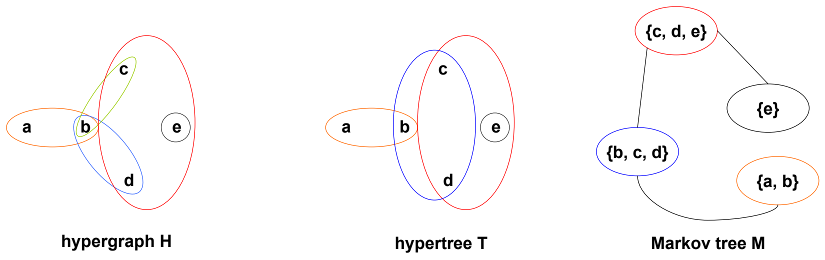

Figure 8.2: Hypergraph and associated covering hypertree and Markov Tree

8.3.2 Hypertrees and combination sequences

D-S systems were proved to be directly modeled by hypergraphs (Shafer et al., 1987). Consider potentials P_1, ..., P_n defined on domains d_1, ..., d_n respectively. Let H be the hypergraph defined by these domains. Then, any complete combination sequence of P_1, ..., P_n determines a covering hypertree for H (Shafer et al., 1987). For a given hypergraph, several covering hypertrees can be found, i.e. there exist several combination sequences. Indeed, an important property is that Dempster's rule of combination is commutative:

$$P_1 \otimes P_2 \otimes ... \otimes P_n = P_n \otimes P_1 \otimes ... \otimes P_2$$

Combining several potentials, combination order does not matter. Nevertheless, some sequences can be practically infeasible, while others can be computed easily.

The research of the optimal sequence corresponds to the research of the optimal covering hypertree (Shafer et al., 1987). Unfortunately, finding the optimal combination sequence was proved to be a NP-complete problem (Arnborg et al., 1987). Heuristic operations, based on hypertree manipulations, were suggested in (Shenoy, 1997) and (Mellouli, 1987) in order to approximate the best combination sequence. In (Lehmann, 2001), Lehmann provides an interesting overview of these heuristic methods.

This approach becomes really interesting when dealing with important systems, where hundreds of potentials have to be combined (Shenoy, 1997), (Xu and Kennes, 1994). For more theoretical details about this scheme, see also (Saffiotti and Umkehrer, 1991).

Conclusion on implementation aspects

This part of the report focused on structural and algorithmic considerations for efficient D-S computations. We first presented relevant structures for representing focal elements and potentials and introduced a new principle for considering belief functions. We explained how focal elements can be implemented with bitsets and why balanced trees or hash tables are relevant for potentials. We described the *delayed mass valuation* principle that tends to limit system reconstructions by building the system without valuating the belief masses. We saw why this principle can not be implemented directly because of strong practical limitations. Second, we described efficient implementations of D-S core algorithms, mainly based on the bitset representation of focal elements. All these structural and algorithmic considerations are unfortunately not sufficient for making D-S formalism theoretically feasible. More advanced methods were developed for decreasing D-S complexity. We introduced some of these methods in the third chapter of this part. We presented approximation methods that tend to reduce the explosion of the number of focal elements. We also explained an interesting scheme for computating combinations, the fast Moebius transformation, which is in fact a generalized fast Fourier transformation. We finally made a brief introduction to advanced heuristics based on hypertree propagation schemes for optimizing combination sequences.

At this point, we studied D-S fundamentals and applicability and we summed up about all the techniques for being able to apply this framework in practice. As a concrete application of these considerations, the next part of the report introduces `eVidenZ`, our D-S engine.

Part III

Introducing `eVidenZ`

Towards `eVidenZ`

This part of the report introduces `eVidenZ`, a D-S engine developed by the LRDE (Epita Research and Development laboratory). This free C++ implementation is currently a research prototype, but it already provides all the structures and basic algorithms for designing efficient D-S engines.

Before presenting `eVidenZ`, we will make an overview of the existing engines based on D-S formalism and explain why we need a free, efficient and general-purpose implementation. We will then introduce our engine, `eVidenZ`. We will detail its features and design and will illustrate its possibilities through a simple use case.

In conclusion, we will develop the future improvements and extensions of our engine.

Chapter 9

Existing architectures and engines

D-S formalism has been successfully applied to a large range of fields (see Chapter 5). All these applications have more or less research purposes even if some of them provide suitable solutions that begin being adopted. In the domain of reasoning under uncertainty, some commercial or free programs already implement evidential frameworks. This chapter is dedicated to the description of the most common architectures for D-S engines. It will also make an overview of the existing implementations of D-S fundamentals and conclude on the need for a new, free, efficient and general-purpose implementation.

9.1 Existing architectures

D-S formalism is usually associated to the family of network-based, graphical frameworks. As for bayesian networks, some researchers have designed message-passing schemes for propagating belief functions among so-called *belief networks*. Let introduce the most famous architectures:

- the Shenoy-Shafer architecture (Shenoy and Shafer, 1990)
- the Lauritzen-Spiegelhalter architecture (Lauritzen and Spiegelhalter, 1988)
- the HUGIN architecture (Jensen et al., 1990)
- the fast-division architecture (Bissig et al., 1997)

Basically, each of these architectures has a Markov tree as underlying computational structure (see 8.2).

The Shenoy-Shafer architecture is a very general architecture used in the framework of valuation networks. The D-S belief function theory fits perfectly into this framework and therefore the Shenoy-Shafer architecture can be used for belief functions.

Lauritzen-Spiegelhalter and HUGIN architectures are very popular in the field of Bayesian networks. However, their application to D-S theory is possible, but not very common because of strong structural restrictions (Shenoy and Shafer, 1990).

The fast-division architecture proposed in (Bissig et al., 1997) is very similar to Lauritzen-Spiegelhalter and HUGIN architectures, but is basically much more adapted to D-S computations.

These architectures provide very general frameworks for implementing D-S engines. They do not focus on local computations (combinations, fusions), but on the belief propagation (i.e. on combination sequences). They come with advanced propagation algorithms and then represent standard frameworks for applying hypertree-based heuristics introduced in Section 8.2. A lot of experimental prototypes and some general engines are based on such architectures (see next subsection). These implementations have obtained very interesting performance results. However, these high-level architectures exceeds our current preoccupations. They are generally not necessary for solving usual problems like the one we are interested in (see Appendix B). Thus, we preferred focusing on local computations in a first time.

9.2 Existing engines

Some general-purpose, network-reasoning engines provide a support for D-S computations. Let introduce the most popular ones:

- Free software
  - ▷ Belief
    Belief is a free Common Lisp implementation of the Dempster fusion and propagation algorithm for graphical belief function models. It also implements the Lauritzen and Spiegelhalter algorithm for graphical probabilistic models. It includes code for manipulating graphical belief models such as Bayesian networks using both belief functions and probabilities as basic representations of uncertainty. It uses the Shenoy and Shafer architecture, so one of its unique features is that it supports both probability distributions and belief functions. It also has limited support for second order models (probability distributions on parameters).
    - ∗ Web site: `www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/reasonng/probabl/belief/`
  - ▷ Pulcinella (Saffiotti and Umkehrer, 1991)
    Pulcinella is written in CommonLisp, and appears as a library of Lisp functions for creating, modifying and evaluating valuation systems. Pulcinella provides primitives to build and evaluate uncertainty models according to several uncertainty calculi, including probability theory, possibility theory, and D-S theory of belief functions; and the possibility theory by Zadeh, Dubois and Prade. Pulcinella is based on the Shenoy-Shafer architecture introduced in the previous subsection.
    - ∗ Web site: `iridia.ulb.ac.be/pulcinella/Welcome.html`
- Commercial software
  - ▷ Netica
    Netica is a software library proposed by Norsys Software Corp. Netica allows developers to implement belief or bayesian networks and to view them in a graphical interface.
    - ∗ Web site: `http://www.norsys.com`

Some researchers have also published toolboxes for D-S computations:

- T. Denoeux (Matlab)

  T. Denoeux proposes some Matlab implementations of D-S-based classifiers and approximations algorithms (see Section 8.1).
  - ▷ Website: `http://www.hds.utc.fr/~tdenoeux/software.htm`

- P. Smets (Matlab)

  P. Smets provide Matlab implementation of Fast Moebius Transformation algorithms (see Section 8.2).

  ▷ Website: `http://iridia.ulb.ac.be/~psmets/`

Finally, lots of research projects resulted in the implementation of *ad hoc* engines. One of the most famous of these engines is the Pathfinder system. This system was started during the 1980s by the Stanford Medical Computer Science program as a 'diagnostic expert system' for diagnosing lymph-node diseases. After consultation between Pathfinder's developers and expert physicians, the system was capable of dealing with more than 60 diseases and 100 symptoms. During trials on actual patients, the Pathfinder achieved 89% accuracy, and more recent versions outperform human experts. Please refer to (Russell and Norvig, 1995), page 547, for more information.

9.3 On the existing engines and the need for a new one

On the one hand, some existing programs provide entire, advanced development platforms for D-S formalism. These general tools are constraining, since they impose a GUI and a development language. They were designed for dealing with several kinds of reasoning networks and then do not especially focus on making D-S principles easier to implement. In spite of the powerful tools they provide, they are somehow too complex and not necessary for the applications we are currently interested in.

On the other hand, lots of papers introduce experimental prototypes of D-S engines. These prototypes would be interesting because they generally focus on local computation algorithms and optimization heuristics. Unfortunately, these engines are rarely publicly available and are usually *ad hoc* implementations hardly adaptable to other contexts.

If some researchers propose free toolboxes, especially for Matlab, there is in fact scarcely no complete, free and general D-S library for common development languages like C++. For our research purposes, we thus preferred implementing such a general library to developing one more *ad hoc* engine. This implementation, `eVidenZ`, is presented in the next Chapter.

Chapter 10

eVidenZ, a general-purpose D-S engine

eVidenZ is a free, general-purpose C++ implementation of D-S local computations. It can be considered as a library base for implementing specific D-S engines. As a library, it provides all the structures and algorithms for implementing D-S engines. Our implementation strictly respects the theoretical definitions of these structures and algorithms, so that eVidenZ is potentially adaptable to most of the possible final applications.

10.1 eVidenZ features

10.1.1 Available structures

- variables;

  eVidenZ is generic towards the type of variables' realizations. eVidenZ uses an unique, internal ordering for indexing the realizations (see Section 10.2).

- focal elements;

  Focal elements in eVidenZ are implemented thanks to bitsets (see Subsection 6.3.2). We chose the dynamic bitsets of *Boost* (Gurtovoy and Abrahams, 2002). Static bitsets like the *STL* ones (Stepanov and Lee, 1994) could not fit our current constraints: the size of the bitset is determined by the number of variables and the number of realizations by variables (see Subsection 6.3.2). Since we do not want these values to be fixed statically, we have to use less efficient, dynamic bitsets like Boost's ones.

- potentials;

  Potentials are implemented thanks to *STL maps* (Stepanov and Lee, 1994). These *maps* are actually balanced trees (see Subsection 6.13). eVidenZ does not propose hash tables yet.

- belief masses.

  There exist actually two versions of eVidenZ. The first one is based on the usual representation of masses as floating values and the second one on the *delayed mass valuation* principle (see Section 6.2). Our current efforts will soon result in mixing these two approaches (see Chapter 11.2).

10.1.2 Available algorithms

- credal system construction;
  - ▷ projection and extension;
    eVidenZ implements the algorithms proposed in Sections 7.2 and 7.3. These efficient algorithms, based on bit string operations, were first proposed in (Haenni and Lehmann, 2001).
  - ▷ combination, marginalization and fusion.
    eVidenZ proposes an implementation of the algorithms exposed in Sections 7.4, 7.5 and 7.6, with efficient regrouping and normalization support.
- pignistic operations;
  - ▷ belief, commonality, plausibility, ignorance, doubt computation.
    eVidenZ provides naturally the procedures to compute all these values. For instance, one can easily interrogate a potential to get the belief on a given focal element.

eVidenZ does not provide any approximation method (see Section 8.1), fast Moebius transform (Section 8.2) nor hypertree optimization (Section 8.2) so far. At the pignistic level, eVidenZ only proposes the atomic interrogation procedures: belief, commonality, plausibility, ignorance and doubt computation. We did not include any decision algorithm (see Section 3.4) yet.

See Chapter 11.2 for information about future improvements of eVidenZ.

10.1.3 Additional features

eVidenZ also provides convenient features like checking different properties:

- checking that a given configuration is well defined, i.e. if the values are grouped in the right number and are valid variable realizations;
- checking that the sum of belief masses is equal to 1.

These simple verifications ensure the sanity of the system the developer is building w.r.t D-S theoretical bases and avoid losing a lot of development time. These checks can naturally be ignored when compiling the final, running engine.

10.2 eVidenZ design

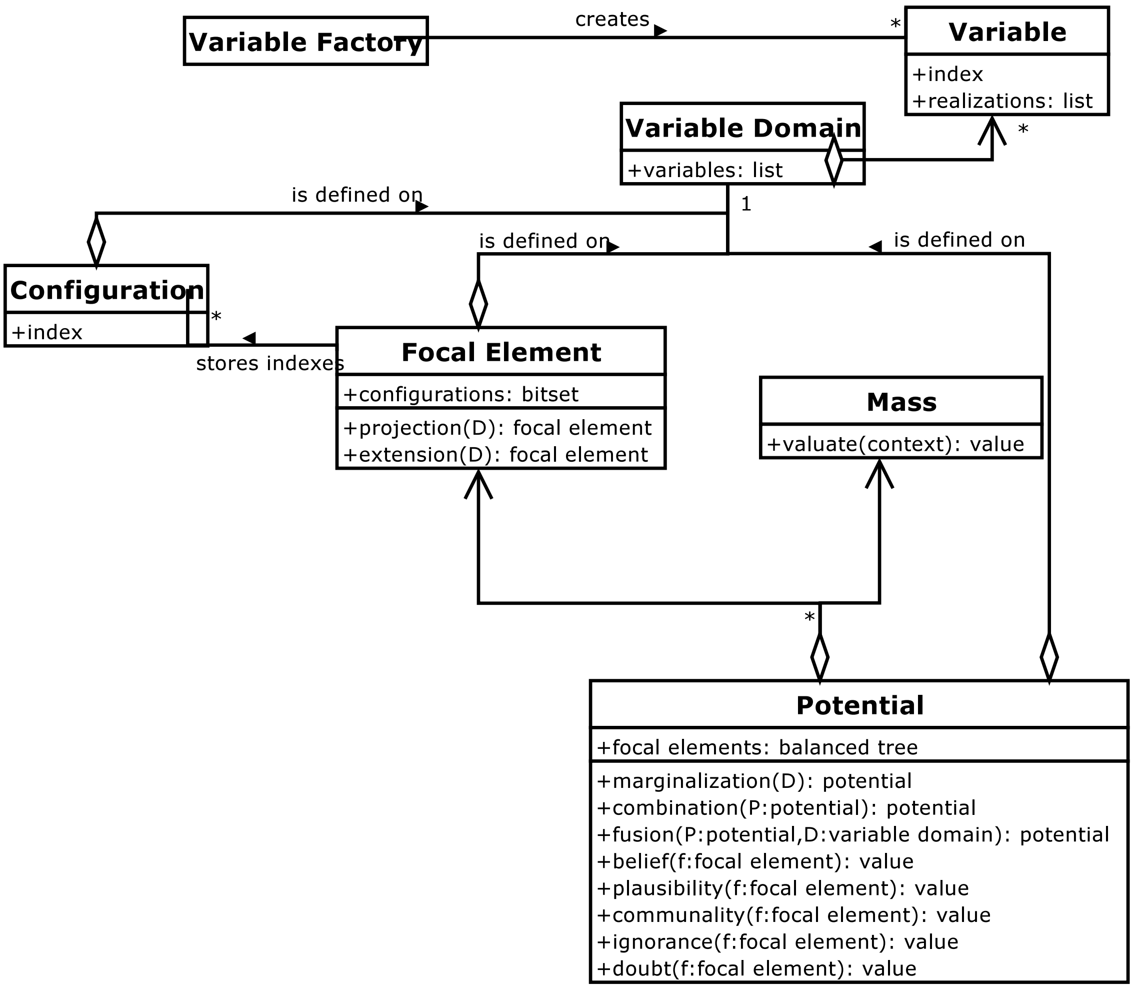
Variable Factory
creates
*
Variable
+index
+realizations: list
Variable Domain
+variables: list
*
is defined on
1
is defined on
is defined on
Configuration
+index
*
stores indexes
Focal Element
+configurations: bitset
+projection(D): focal element
+extension(D): focal element
Mass
+valuate(context): value
*
Potential
+focal elements: balanced tree
+marginalization(D): potential
+combination(P:potential): potential
+fusion(P:potential,D:variable domain): potential
+belief(f:focal element): value
+plausibility(f:focal element): value
+communality(f:focal element): value
+ignorance(f:focal element): value
+doubt(f:focal element): value

Figure 10.1: eVidenZ design

As described in the previous sections, eVidenZ is based on efficient algorithms that imply special data representations, like bitsets for focal elements. eVidenZ is then designed for making all this internal machinery totally transparent for the final user. Client code manipulates only comprehensive representations that are very close to their theoretical definitions. eVidenZ is for example generic w.r.t. the representation of the realizations of the variables, so that the final user can define his system and display his results with any convenient type.

Figure 10.1 describes the design of eVidenZ in pseudo-UML. Since eVidenZ is implemented in C++, its design is object-oriented and composed of the core components exposed in Chapter 6.

Variable Factory We already mentioned in Subsection 6.3.2 that a global ordering of variables is needed to manage efficient research operations on configurations and thus focal elements. Then, the *Variable Factory* is an object dedicated to the creation and the management of the variables. It is instantiatied once for a given problem and follows the principle of the Factory design pattern described in (Gamma et al., 1995). $Variable$ objects can be created only through a $Variable\ Factory$, so that the $Variable\ Factory$ maintains, for instance, an exhaustive indexing of the variables.

Variable A $Variable$ object is created by a $Variable\ Factory$ and obtains an index. $Variable$ purpose is mainly to keep information about its realizations. These realizations can be from any initial type. The $Variable$ performs an ordering of these realizations, so that the other objects manipulate only realization indexes.

Variable Domain A $Variable\ Domain$ is basically a list of $Variable$ objects. It represents the domain on which configurations, focal elements and thus potentials are defined. This is a central object, since focal elements query their domain for obtaining, for example, the index of a given configuration. The $Variable\ Domain$ makes the link between the initial representation of the variables and the internal system that is entirely based on indexes. It delegates queries from $Focal\ Element$s to $Variable$s, like displaying the real form of a configuration.

Configuration A configuration is theoretically a tuple of variable values, defined on a variable domain. Actually, in our internal system, a $Configuration$ object is entirely defined by an index on the associated $Variable\ Domain$. This index corresponds to a bit in a focal element bitset on the same variable domain (see Section 6.3.2). Only the $Variable\ Domain$ on which a $Configuration$ object relies is able to compute this index and inversely to rebuild the real representation of the configuration from this index.

Focal Element In `eVidenZ`, focal elements are implemented thanks to bitsets, as described in Section 6.3.2. Then, instead of storing $Configuration$ objects, a $Focal\ Element$ only stores a bitset defined on a $Variable\ Domain$. The indexes of the bits set to 1 correspond to the indexes of the associated $Configuration$ objects. $Focal\ Element$ objects also provide procedures to be projected or extended to a new domain.

Mass As it was already explained in Section 10.1.1, there exist two versions of `eVidenZ`. One is based on the *delayed valuation* principle and the other one on the classical representation of belief masses. In the first version, belief masses are functors (see Subsection 6.2.4), so that they are waiting for a *context* to be valuated. This context can be of any type and depends on the final system implemented. In the second version of `eVidenZ`, $Mass$ objects are simply floating values.

Potential The $Potential$ object holds the final information defined by pairs of $Focal\ Element$s and $Mass$es. These pairs are stored in a balanced tree as exposed in Subsection 6.13. Finally, the $Potential$ object provides all the member functions to be marginalized on a new domain and to be combined/fusion-ed with another potential. It also permits the developer to retrieve belief, commonality, plausibility, ignorance and doubt values on a given focal element for pignistic operations.

10.3 Use case

We propose the implementation of a simple example using eVidenZ. This example is a simplified "Clue" game, already presented in Section 6.1.

Consider two players during such a Clue game. The first one has strong belief that the murderer is "Colonel Mustard" and that the crime was perpetrated in the "Kitchen". The first player has not any information about the weapon and does not totally trusts his clues. Then, his belief is defined on the domain $D_1 = \{Murderer, Room\}$. A belief mass of 0.8 is associated to the focal element $f_{11} = \{(``Colonel\ Mustard", ``Kitchen")\}$ and a mass of 0.2 to the entire frame of discernment (f_{12}). The second player has strong belief that the murderer is "Colonel Mustard" and that the weapon is a "Candlestick". The second player has no information about the place and does not totally trusts his clues. Then, his belief is defined on the domain $D_2 = \{Murderer, Weapon\}$. A belief mass of 0.8 is associated to the focal element $f_{21} = \{(``Colonel\ Mustard", ``Candlestick")\}$ and a mass of 0.2 to the entire frame of discernment (f_{22}).

It is then interesting to see how D-S theory combines the knowledge of these two players on the global domain $D = \{Murderer, Room, Weapon\}$. In the following listings, we describe the implementation of the combination of these two players' beliefs using eVidenZ, in C++. The final potential is interrogated to obtain pignistic values on the focal element $f_{final} = \{(Colonel\ Mustard, Kitchen, Candlestick)\}$.

10.3.1 Prerequisites

```
#include <iostream>
#include <string>

#include <evidenz.hh>

typedef FocalElement<std::string> Focal_type;
typedef Configuration<std::string> Config_type;
typedef VarFactory<std::string> VarFactory_type;
typedef Var<std::string> Var_type;
typedef VarDomain<std::string> VarDomain_type;
typedef Potential<std::string, Event> Potential_type;
typedef std::list<std::string> List_type;

typedef float Mass;
```

Variable realizations will be represented by character strings. This example is implemented by the "usual" version of eVidenZ, so that masses are simply floating values. We also define some type aliases for better readability.

10.3.2 System definition

```
  // Global Variable Factory

  VarFactory_type vf;

  // Definition of the ''Murderer'' variable

  Var_type& Murderer = vf.new_var();
```

```
Murderer += "Colonel␣Mustard";
Murderer += "Miss␣Scarlett";
Murderer += "Mrs.␣Peacock";

// Definition of the ‘‘Room’’ variable

Var_type& Room = vf.new_var();
Room += "Dining␣room";
Room += "Kitchen";

// Definition of the ‘‘Weapon’’ variable

Var_type& Weapon = vf.new_var();
Weapon += "Dagger";
Weapon += "Candlestick";

// Global variable domain D = {Murderer, Room, Weapon}

VarDomain_type D(vf);
D += Murderer;
D += Room;
D += Weapon;
```

Here begins the real executive code. An unique $Variable\ Factory$ is created. Thanks to this $Variable\ Factory$, the three variables, $Murderer$, $Room$ and $Weapon$ and their respective realizations are defined. The global variable domain D is set to $D = \{Murderer, Room, Weapon\}$.

10.3.3 First player

```
// Variable domain

VarDomain_type D1(vf);
D1 += Murderer;
D1 += Room;


// First focal element

Focal_type f11(D1);

List_type l11;
l11.push_back("Colonel␣Mustard");
l11.push_back("Kitchen");
Config_type config11(D1, l11);

f11 += config11;


// Second focal element

Focal_type f12(D1);

f12.add_all_configs();
```

```
// First potential

Potential_type player1(D1);
Mass m1 = 0.8;

player1.add(f11, m1);
player1.add(f12, 1 - m1);

std::cout << ‘‘Checking player1’’ << std::endl;
player1.check();
```

This source code defines first player's belief, which is defined on the variable domain $D_1 = \{Murderer, Room\}$. The focal element f_{11} contains only one configuration, $config_{11} = (Colonel Mustard, Kitchen)$. Note that $config_1$ is defined by a standard list of realizations. The second focal element f_{12} includes the entire frame of discernment on the domain D_1 thanks the convenient $add_all_configs()$ function.

The potential $player1$ is created. The focal element f_{11} is associated to a mass $m1$ set to 0.8 and f_{12} is associated to the complement of $m1$, so that the total belief is equal to 1. Finally, the sanity of this potential is checked.

10.3.4 Second player

```
// Variable domain

VarDomain_type D2(vf);
D2 += Murderer;
D2 += Weapon;


// First focal element

Focal_type f21(D2);

List_type l21;
l21.push_back("Colonel␣Mustard");
l21.push_back("Candlestick");
Config_type config21(D2, l21);

fs21 += config21;


// Second focal element

Focal_type fs22(D2);

f22.add_all_configs();


// Second potential

Potential_type player2(D2);
Mass m2 = 0.8;

player2.add(f21, m2);
```

```
  player2.add(f22, 1 - m2);

  std::cout << ‘‘Checking player2’’ << std::endl;
  player2.check();
```

This source code defines second player's belief, which is defined on the variable domain $D_2 = \{Murderer, Weapon\}$. The focal element f_{11} contains only one configuration, $config_{21} = (Colonel\ Mustard, Candlestick)$. The second focal element f_{22} includes the entire frame of discernment on the domain D_2.

The potential $player2$ is created. The focal element f_{21} is associated to a mass $m2$ set to 0.8 and f_{22} is associated to the complement of $m2$. Finally, the sanity of this potential is checked.

10.3.5 Combination and interrogation

```
  // player1 & player2 combination

  Potential_type final = player1 + player2;


  // focal element for interrogation:
  // ffinal = { (Colonel Mustard, Kitchen, Candlestick) }

  Focal_type ffinal(D);

  List_type lfinal;
  lfinal.push_back("Colonel␣Mustard");
  lfinal.push_back("Kitchen");
  lfinal.push_back("Candlestick");
  Config_type configfinal(D, lfinal);

  ffinal += configfinal;

  std::cout << ‘‘Checking ffinal...’’ << std::endl;

  // System interrogation

  std::cout << "player1␣+␣player2:" << std::endl << final << std::endl;

  std::cout << "Focal␣element␣" << ffinal << ":␣" << std::endl
            << "bel␣=␣" << final.bel(ffinal) << std::endl
            << "pls␣=␣" << final.pls(ffinal) << std::endl
            << "dou␣=␣" << final.dou(ffinal) << std::endl
            << "com␣=␣" << final.com(ffinal) << std::endl
            << "ign␣=␣" << final.ign(ffinal) << std::endl
            << std::endl;
}
```

Potentials $player1$ and $player2$ are combined thanks to $operator+$. The potential obtained, $final$, is defined on $D = D_1 \bigcup D_2$. A focal element $ffinal = \{(ColonelMustard, Kitchen, Candlestick)\}$ is built to query the $final$ potential. Finally, $final$ is displayed and belief, plausibility, doubt, commonality and ignorance values on $ffinal$ are computed.

The following results are obtained:

```
Checking player1...
Sanity check OK

Checking player2...
Sanity check OK

Checking ffinal...
Sanity check OK

player1 + player2:

{ ( Colonel Mustard, Kitchen, Candlestick ) } (0.64)

{ ( Colonel Mustard, Dining room, Candlestick ),
  ( Colonel Mustard, Kitchen, Candlestick ) } (0.16)

{ ( Colonel Mustard, Kitchen, Dagger ),
  ( Colonel Mustard, Kitchen, Candlestick ) } (0.16)

{ ( Colonel Mustard, Dining room, Dagger ),
  ( Colonel Mustard, Diningroom, Candlestick ),
  ( Colonel Mustard, Kitchen, Dagger ),
  ( Colonel Mustard, Kitchen, Candlestick ),
  ( Miss Scarlett, Dining room, Dagger ),
  ( Miss Scarlett, Diningroom, Candlestick ),
  ( Miss Scarlett, Kitchen, Dagger ),
  ( Miss Scarlett, Kitchen, Candlestick ),
  ( Mrs. Peacock, Dining room, Dagger ),
  ( Mrs. Peacock, Dining room, Candlestick ),
  ( Mrs. Peacock, Kitchen, Dagger ),
  ( Mrs. Peacock, Kitchen, Candlestick ) } (0.04)

Focal element { ( Colonel Mustard, Kitchen, Candlestick ) }:
bel = 0.64
pls = 1
dou = 0
com = 1
ign = 0.36
```

Displaying a potential results in showing its focal elements and their associated masses.

10.4 Performance

Appendix A proposes a simple performance study based on `eVidenZ`. This study confirms the theoretical influence of different factors on D-S computations. It reveals unsurprising time and memory explosions that are in fact not too penalizing when dealing with real-scale applications. These benchmarks validates `eVidenZ` behavior since results match the theoretical complexities of `eVidenZ` algorithms.

Since existing implementations of D-S engines are mostly experimental implementations (see 9), performance comparisons are not totally relevant.

Haenni and Lehmann propose a simple benchmark of their experimental engine in (Haenni and Lehmann, 2001). Figure 10.4 sums up this benchmark. Haenni and Lehmann propose the combination of two randomly generated potentials. On a 400 Mhz PowerMac G3, they achieve to combine the two potentials in 96.5 seconds. With the same settings, using `eVidenZ` and a 2.4 Ghz

Xeon, we achieve the combination in about 5 seconds. These results are merely comparable since they rely on the same algorithms and underlying structures. This tends to confirm the relevance of our implementation, even if more reliable comparisons would be welcome.

	Haenni and Lehmann's implementation (Haenni and Lehmann, 2001)	**eVidenZ**
Implementation of focal elements	bitsets	bitsets
Implementation of potentials	balanced trees	balanced trees
Language used	Macintosh Common Lisp	C++
P_1's number of focal elements	1862	1862
P_2's number of focal elements	1135	1135
P_1 and P_2's number of binary variables	11	11
Test machine	400 Mhz Power Mac G3	2.4 Ghz Xeon
$P_1 \otimes P_2$ computation time	96.5 sec	5 sec

Figure 10.2: Comparison benchmark

10.5 Availability

The first version (0.1) of `eVidenZ` will be released soon and will be distributed under the GNU/GPL licence.

Please check `www.lrde.epita.fr` for availability.

Chapter 11

Conclusion and implementation perspectives

11.1 Summary

This part of the report was dedicated on the presentation of our implementation, `eVidenZ`. We first presented the most famous architectures and existing implementations. We saw that complex, advanced engines already exist. If such engines are interesting for studying network-propagation algorithms, they are actually oversized for the concrete applications we are interested in. Most of the other implementations are research prototypes not publicly available or *ad hoc* engines we can not reuse. We thus conclude on the need for a free, simple and general-purpose D-S base.

We then introduced our engine, `eVidenZ`, implemented in C++. `eVidenZ` fill nearly all the requirements for efficient D-S local computations. Its features are close to the D-S theoretical bases so that it is adaptable to every specific context. All the structures and algorithms described in the theoretical definitions of D-S framework can be found in `eVidenZ`. Then, researchers can manipulate these structures an intuitive and comprehensive way. Nevertheless, `eVidenZ` design hides an efficient, generic internal organization for managing heavy computations.

`eVidenZ` is currently a research prototype and then does not implement all the principles proposed in Part II. Anyway, `eVidenZ` will be applied to different research projects as exposed in B. It will be significantly improved in the meantime.

11.2 Future improvements of `eVidenZ`

11.2.1 Code generation

The main problem of the *delayed mass valuation* principle is the functor indirections it induces (see Section 6.2). We can not profit from the two-layer and generative nature of C++ since the algorithmic structures involved rely on dynamically computed values. Actually, we only need another evaluation layer to avoid these indirections.

We are currently studying code generation to remove indirections as much as possible. Rather than building directly the final system, the idea would be to generate code after the system definition. Such approaches raise important problems: in a naive implementation, the code generation would have the same complexity than unrolling all the indirections at runtime. This way, code generation simply could not be feasible. Some improvements are then necessary. An important property to note is that, in a mass evaluation tree, a lot of code is shared with other focal elements and potentials. This is not surprising since potentials are generated sequentially by combination and rely on their parent masses. We are currently evaluating the feasibility of generating code step by step, with memoizing. However, we still have serious apprehension about the generation time and about the size of the generated code. Thus, we are also studying the applicability of partial *delayed mass valuation* by mixing classical mass values and *delayed* masses.

11.2.2 Mixing classical mass values and *delayed* masses

We saw in Section 6.2 that both *delayed* and usual representations for belief masses are not totally satisfactory. `eVidenZ` disposes of both mass representations and is consequently split in two different versions. We plan to merge these versions to allow the developer to switch from one representation to another in the same D-S engine. The idea would be to apply *delayed mass valuation* principles partially. Such hybrid approaches could be related to the idea of *partial evaluation* (see (Anisko, 2002)): given a part of available mass values, some improvements, like approximations, would be partially applicable. The remaining belief masses would not be valuated and would enable the reasoning system to be reused on a certain class of problems sharing the already valuated masses.

Remember the example of the medical diagnosis system: considering a group of women, diseases like prostate cancer can be excluded. Then, following the usual approach, belief masses on prostate cancer are directly set to 0 and the focal elements are eliminated. The masses on other disease groups are not valuated, so that the system is built once for all this group of women.

The main problem will be to keep a reasonable number of indirections. We are currently studying heuristics to determine the critical limit above which indirections become problematic. These heuristics would enable us to switch efficiently from *delayed mass valuation* to common mass values during the credal construction. Mixing such schyzophrenic behavior with code generation, we have good hope for obtaining interesting performance.

11.2.3 Other future improvements

At the credal level, we plan to improve `eVidenZ` performance with mechanisms like memoizing and hash tables for potentials (see Subsection 6.4.3). The fast Moebius transform (see Subsection 8.2) and some approximation methods (see Subsection 8.1) will also be implemented soon.

At the pignistic level, decision algorithms, mainly based on Smets' bet probability (see SubSection 3.4.2), will be available in a near future. This way, `eVidenZ` will provide solutions for implementing entire D-S reasoning systems, from the construction of the system to the decision making.

In a more pragmatic point of view, `eVidenZ` usability will be increased by lightening the syntax. As you should have noticed in Section 10.3, `eVidenZ` syntax is currently too heavy. Simpler initializers could sensibly improve the readability of the code (Wang-Reboud, 2002).

General conclusion

This report had two main goals. First one was to make a global and analyzed bibliographic survey of both theoretical and implementation aspects of evidence theory. This task was quite delicate because of the great number of papers dealing with this theory, often being contradictory or erroneous. The second goal was to present our contributions: *delayed mass valuation* and `eVidenZ`, a Dempster-Shafer engine.

The first part of the report clarified the theoretical foundations of evidence theory. The Transferable Belief Model was presented, and a short comparison with other models pointed out the interesting properties of the evidence model. But the appealing features of this framework are lessened by the high complexity of the Dempster's combination rule, making some problems really difficult or even impossible to compute. Our practical observations on D-S feasibility are presented in a performance study proposed in Appendix A. This study globally confirms the impressive theoretical complexities of D-S computations. It also moderates this theoretical 'infeasibility' by showing that real-scale applications, even if relatively heavy, are reasonably possible. Thus, the implementation of evidence theory has to be carefully taken into account.

A wide panel of efficient structures, algorithms and advanced optimizations for making computations more feasible was presented in the second part. This study is mainly a global overview of existing methods, but we also presented our personal observations and criticisms. We explained for instance that factorizing D-S fusion using mass functors could be a really interesting principle. This would be especially efficient in image processing, where identical fusion processes, modulo the mass values, are computed for each pixel. We called this principle *delayed mass valuation*. Unfortunately, this innovative design implies strong implementation difficulties we are currently trying to solve.

In the third part of the report, our survey of existing engines concludes on the need for a new, general engine to perform local computations efficiently and conveniently. So we decided to develop our own library in C++, `eVidenZ`, introduced in this report. We described how `eVidenZ` is designed for maximum efficiency while staying very close to the theoretical definitions and providing natural, convenient features.

Our future work will now focus on two main tasks. First, we have to extend `eVidenZ` and make an in-depth study of the feasibility of code generation for the *delayed mass valuation*. Then, our engine will be applied to two research projects: cancerous cells analysis and satellite image segmentation. A simple introduction to these two projects is given in Appendix B.

Bibliography

Anisko, R. (2002). Towards automatic partial evaluation for the c++ language. Technical report, EPITA Research And Development Laboratory, Paris, France.

Appriou, A. (1991). Probabilités et incertitude en fusion de données multi-senseurs. *Revue Scientifique et Technique de la Défense*, 11:27–40.

Arnborg, S., Corneil, D. G., and Proskurowski, A. (1987). Complexity of finding embeddings in a k-tree. In *SIAM Journal of Algebraic and Discrete Methods*, volume 8, pages 277–284.

Bauer, M. (1996). Approximations for decision making in the Dempster-Shafer theory. In *Uncertainty in Artificial Intelligence*, pages 339–344, Saarbrücken.

Bissig, R., Kohlas, J., and Lehmann, N. (1997). Fast-division architecture for dempster-shafer belief functions. In Gabbay, D., Kruse, R., Nonnengart, A., and Ohlbach, H., editors, *Qualitative and Quantitative Practical Reasoning, First International Joint Conference on Qualitative and Quantitative Practical Reasoning; ECSQARU–FAPR'97*. Springer.

Bloch, I. (1996). Information Combination Operators for Data Fusion: A Comparative Review with Classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(1):52–67.

Coplien, J. O. (1992). *Advanced C++ Programming Styles and Idioms*. Software Patterns Series. Addison-Wesley, Reading, Massachusetts.

Dempster, A. P. (1967). Upper and lower probabilities induced by a multiple valued mapping. *Ann. Math. Statistics*, 38:325–339.

Dubois, D., Grabisch, M., Prade, H., and Smets, P. (2001). Using the transferable belief model and a qualitative possibility theory approach on an illustrative example: the assessment of the value of a candidate. *Intern. J. Intell. Systems*, forthcoming. forthcoming.

Dubois, D. and Prade, H. (1982). On several representations of an uncertainty body of evidence. In Gupta, M. M. and Sanchez, E., editors, *Fuzzy Information and Decision Processes*, pages 167–181. North-holland, New-York.

Dubois, D. and Prade, H. (1990). Aggregation of possibility measures. In *Multiperson Decision Making Models Using Fuzzy Sets and Possibility Theory*, pages 55–63. Kluwer Academic Publ.

Elouedi, Z., Mellouli, K., and Smets, P. (2000). Classification with belief decision trees. In *Proceedings of the 9th International Conference on Artificial Intelligence : Methodology, Systems, Architectures : AIMSA 2000, Springer Lecture Notes of Articial Intelligence*.

Elouedi, Z., Mellouli, K., and Smets, P. (2001). Belief decision trees: Theoretical foundations. *Approximate Reasoning*, 28:91–124.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements od Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, NY.

Genest, C. and Zidek, J. V. (1986). Combining probability distributions: A critique and an annotated bibliography. *Statistical Science*, 1:114–148.

Guan, J. W. and Bell, D. A. (1991). *Evidence Theory and its Applications*. North Holland, Amsterdam.

Gurtovoy, A. and Abrahams, D. (2002). The boost C++ metaprogramming library.

Haenni, R. and Lehmann, N. (2001). Implementing belief function computations. Technical Report 01-28, Department of Informatics, University of Fribourg.

Heckerman, D. (1986). Probabilistic interpretation for MYCIN's certainty factors. In Kanal, L. N. and Lemmer, J. F., editors, *Uncertainty in Artificial Intelligence*, pages 167–196. North Holland, Amsterdam.

Henrion, M., Shachter, R. D., Kanal, L. N., and Lemmer, J. F., editors (1990). *Uncertainty in Artificial Intelligence 5*. North Holland, Amsteram.

Jeffrey, R. C. (1988). Conditioning, kinematics, and exchangeability. In Skyrms, B. and Harper, W. L., editors, *Causation, Chance, and Credence*, pages 21–255. Reidel, Dordrecht.

Jensen, F., Lauritzen, S., and Olesen, K. (1990). Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quaterly*, 4:269–282.

Kennes, R. and Smets, P. (1990). Computational aspects of the Moebius transform. In Henrion et al. (1990), pages 01–416.

Kohlas, J. and Monney, P. A. (1994). Theory of evidence: A survey of its mathematical foundations, applications and computations. *ZOR-Mathematical Methods of Operatioanl Research*, 39:35–68.

Kohlas, J. and Monney, P. A. (1995). *A Mathematical Theory of Hints: An Approach to Dempster-Shafer Theory of Evidence*. Lecture Notes in Economics and Mathematical Systems No. 425. Springer-Verlag.

Kreinovich, V., Bernat, A., Borrett, W., and Villa, E. (1994). Monte-carlo methods make dempster-shafer formalism feasible. In Yager et al. (1994), pages 175–191.

Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local computation with probabilities and graphical structures and their application to expert systems. *J. Royal Statistical Society B*, 50:157–224.

Le Hégarat-Mascle, S., Bloch, I., and Vidal-Madjar, D. (1998). Introduction of neighborhood infomation in evidence theory and application to data fusion of radar and optical images with partial cloud cover. *Pattern Recognition*, 31(11):1811–1823.

Lee, E. S. and Zhu, Q. (1995). *Fuzzy and evidence reasoning*. Physica-Verlag Heidelberg, Studies in Fuzziness.

Lehmann, N. (2001). *Argumentation System and Belief Functions*. PhD thesis, Department of Informatics, University of Fribourg.

Levi, I. (1983). Consonance, dissonance end evidentiary mechanisms. In Gardenfors, P., Hansson, B., and Sahlin, N. E., editors, *Evidentiary Value: Philosophical, Judicial and Psychological Aspects of a Theory*, pages 27–43. C.W.K. Gleerups, Lund, Sweden.

Lowrance, J., Garvey, T., and Strat, T. (1986). A framework for evidential-reasoning systems. In *8th National Conference of the American Association for Artificial Intelligence*, pages 896–903.

LRDE (2003). Automatic detection of healthy or cancerous cells (adhoc).

Mellouli, K. (1987). *On the propagation of beliefs in network using the Dempster-Shafer theory of evidence*. PhD thesis, School of business, University of Kansas, Lawrence.

Orponen, P. (1990). Dempster's rule of combination is #p-complete. *Artificial Intelligence*, 44(1-2):245–253.

Parsons, S. and Hunter, A. (1998). A review of uncertainty handling formalisms. In *Applications of Uncertainty Formalisms*, pages 8–37.

Pearl, J. (1990a). Bayesian decision methods. In Shafer, G. and Pearl, J., editors, *Readings in Uncertain Reasoning*, pages 345–352. Kaufmann, San Mateo, CA.

Pearl, J. (1990b). Reasoning with belief functions: an analysis of compatibility. *Intern. J. Approx. Reasoning*, 4:363–390.

Provan, G. M. (1990). The application of dempster shafer theory to a logic-based visual recognition system. In Henrion, M., Shachter, R. D., Kanal, L. N., and Lemmer, J. F., editors, *Uncertainty in Artificial Intelligence 5*, pages 389–405. North-Holland, Amsterdam.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence A Modern Approach*. Prentice Hall, second edition.

Saffiotti, A. and Umkehrer, E. (1991). Pulcinella: a general tool for propagating uncertainty in valuation networks. pages 323–331.

Sentz, K. and Ferson, S. (2002). Combination rules in dempster-shafer theory. In *6th World Multiconference on Systemics, Cybernetics and Informatics*.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton Univ. Press. Princeton, NJ.

Shafer, G., Shenoy, P. P., and Mellouli, K. (1987). Propagating belief functions in qualitative markov trees. *Int. J. Approx. Reasoning*, 1:349–400.

Shenoy, P. (1997). Binary join trees for computing marginals in the shenoy-shafer architecture.

Shenoy, P. P. and Shafer, G. (1990). Axioms for probability and belief functions propagation. In Shachter, R. D., Levitt, T. S., Kanal, L. N., and Lemmer, J. F., editors, *Uncertainty in Artificial Intelligence 4*, pages 159–198. North Holland, Amsterdam.

Shortliffe, E. H. (1976). *Computer Based Medical Consultations: MYCIN*. American Elsevier, New York (1976.

Smets, P. (1990a). The combination of evidence in the transferable belief model. *IEEE Pattern analysis and Machine Intelligence*, 12:447–458.

Smets, P. (1990b). Constructing the pignistic probability function in a context of uncertainty. In Henrion et al. (1990), pages 29–40.

Smets, P. (1990c). The transferable belief model and possibility theory. Technical Report TR/IRIDIA/90-2, IRIDIA, Bruxelles.

Smets, P. (1991). Varieties of ignorance and the need for well-founded theories. *Information Sciences*, 57:135–144.

Smets, P. (1993). No Dutch book can be built against the TBM even though update is not obtained by Bayes rule of conditioning. In *SIS, Workshop on Probabilistic Expert Systems. R. Scozzafava (ed.), Roma, Italy*, pages 181–204.

Smets, P. (1994). What is Dempster-Shafer's model? In Yager et al. (1994), pages 5–34.

Smets, P. (1997). Imperfect information : Imprecision - uncertainty. In Motro, A. and Smets, P., editors, *Uncertainty Management in Information Systems: from Needs to Solutions*, pages 225–254. Kluwer, Boston.

Smets, P. (1998). Numerical representation of uncertainty. In Gabbay, D. M. and Smets, P., editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 3, pages 265–311. Kluwer, Doordrecht, The Netherlands.

Smets, P. (1999). Practical uses of belief functions. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden*.

Smets, P. (2001). Decision making in a context where uncertainty is represented by belief functions. Technical report, IRIDIA, Bruxelles.

Smets, P., Hsia, Y., Saffiotti, A., Kennes, R., Xu, H., and Umkehrer, E. (1991). The transferable belief model. In Kruse, R. and Siegel, P., editors, *Symbolic and Quantitative Approaches to Uncertainty*, pages 91–98. Springer-Verlag, Berlin.

Smets, P. and Kennes, R. (1994). The transferable belief model. *Artificial Intelligence*, 66:191–234.

Smithson, M. (1989). *Ignorance and Uncertainty: Emerging Paradigms*. Springer-Verlag, New York.

Stepanov, A. A. and Lee, M. (1994). The Standard Template Library. Technical Report X3J16/94-0095, WG21/N0482.

Teller, P. (1973). Conditionalization and observation. *Synthesis*, 26:218–258.

Tessem, B. (1993). Approximations for efficient computation in the theory of evidence. *Artificial Intelligence*, 61:315–329.

Thoma, H. M. (1991). Belief function computations. In Goodman, I. R., Gupta, M. M., Nguyen, H. T., and Rogers, G. S., editors, *Conditional Logic in Expert Systems*, pages 269–308. North-Holland, Amsterdam.

Vannoorenberghe, P., Colot, O., and DeBrucq, D. (1999). Color image segmentation using dempster-shafer's theory. In *IEEE International Conference on Image Processing, ICIP'99, October 25-28, Kober, Japan, 1999*.

Voorbraak, F. (1989). A computationally efficient approximation of Dempster-Shafer theory. *International Journal of Machine Studies*, 30:525–536.

Wang-Reboud, A. (2002). User-friendly and secure initializations in c++. Technical report, EPITA Research And Development Laboratory, Paris, France.

Wilson, N. (1991). A monte-carlo algorithm for dempster-shafer belief. Technical Report QMW-DCS-1991-535, Queen Mary College, Department of Computer Science.

Xu, H. and Kennes, R. (1994). Steps towards an efficient implementation of Dempster-Shafer theory. In Yager et al. (1994), pages 153–174.

Yager, R. R. (1987). On the dempster-shafer framework and new combination rules. *Information Sciences*, 41:93–137.

Yager, R. R. (1990). Decision making under Dempster-Shafer uncertainties. Technical Report MII-915, Iona College.

Yager, R. R., Kacprzyk, J., and Fedrizzi, M., editors (1994). *Advances in the Dempster-Shafer Theory of Evidence*. Wiley, New York.

Zadeh, L. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28.

Zadeh, L. A. (1965). Fuzzy sets. *Inform.Control.*, 8:338–353.

Part IV

Appendix

Appendix A

Performance study

The implementation of `eVidenZ` *led us to benchmark the influence of different factors on execution time, memory load and final number of focal elements. This simple study allows us to discuss the practical applicability of D-S formalism and to validate the behavior of our engine.*

A.1 Description of the test bed

Our test bed consists in the combination of two potentials P_1 and P_2 randomly generated. We studied the influence of different factors on the computation time, the memory load and the number of focal elements of $P_1 \otimes P_2$. To do so, we varied the following factors of P_1 and P_2:

- the number of variables (for both P_1's and P_2's domains);
  - ▷ 2 to 12 variables;
  - ▷ 2 realizations per variable;
  - ▷ 1000 focal elements per potential;
  - ▷ 100 configurations per focal element;
  - ▷ computation time results: Figure A.1;
  - ▷ memory load results: Figure A.2;
- the number of realizations per variable (for both P_1's and P_2's domains);
  - ▷ 2 variables;
  - ▷ 500 to 4000 realizations per variable;
  - ▷ 100 focal elements per potential;
  - ▷ 100 configurations per focal element;
  - ▷ computation time results: Figure A.3;
  - ▷ memory load results: Figure A.4;
  - ▷ number of final focal elements evolution: Figure A.5;
- the number of focal elements per potential (for both P_1 and P_2);

  - ▷ 2 variables;
  - ▷ 500 realizations per variable;
  - ▷ 100 to 700 focal elements per potential;
  - ▷ 100 configurations per focal element;

  - ▷ computation time results: Figure A.6;
  - ▷ memory load results: Figure A.7;
  - ▷ number of final focal elements evolution: Figure A.8;

- the number of configurations per focal element (for both P_1 and P_2);
  - ▷ 2 variables;
  - ▷ 1000 realizations per variable;
  - ▷ 200 focal elements per potential;
  - ▷ 100 to 250 configurations per focal element;

  - ▷ computation time results: Figure A.9;
  - ▷ memory load results: Figure A.10;
  - ▷ number of final focal elements evolution: Figure A.11.

Tests were performed on a 2.4Ghz Xeon with 1Go Ram and implemented with `eVidenZ` (see Chapter 10). Note that most of the curves are given with a very approximative approaching formula: measurable ranges are so narrow for all the factors that it is difficult to obtain significant trends.

A general observation is that the memory load is generally the most restrictive factor. The size of the focal elements grows so heavily that, very quickly, potentials can not be stored in memory. We can also notice that the memory load is highly correlated to the number of resulting focal elements, which are the heaviest structures.

A.2 Number of variables

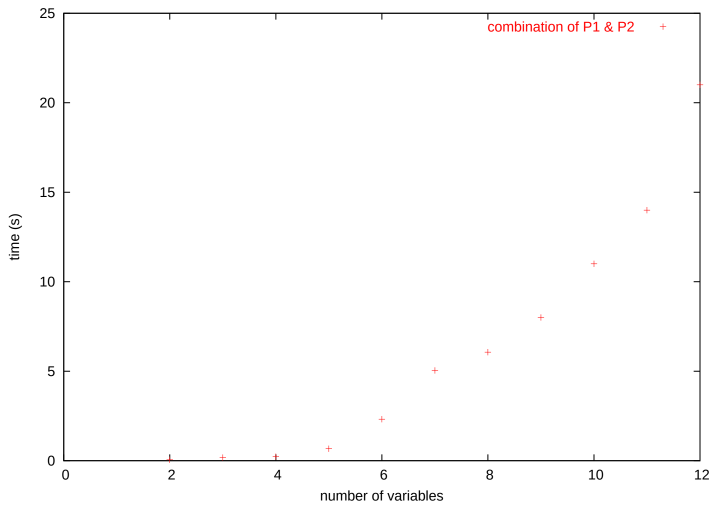

Figure A.1: Influence of the variable number on computation time

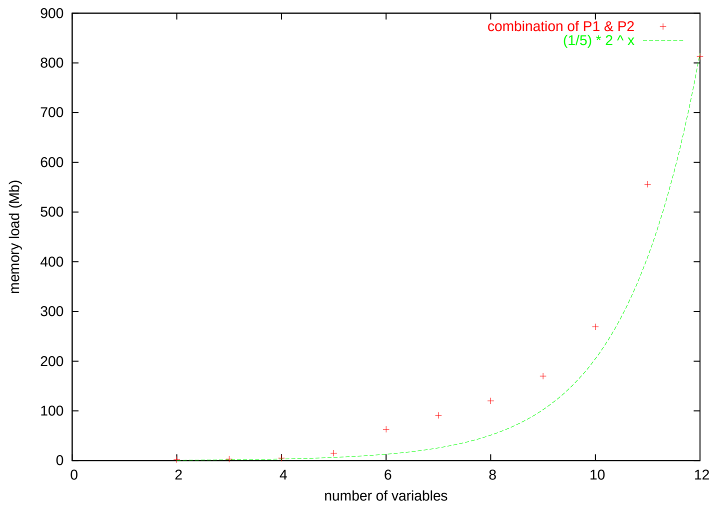

Figure A.2: Influence of the variable number on memory load

The number of variables especially influences the memory load. As it was explained in Section 6.3.2, the size of focal elements heavily depends on the number of variables when using the bitset representation. Remember that the size of the bitset is given by:

$$S = \prod_{i=1}^{n} S_i \tag{A.1}$$

Our results confirm the empirical limit of twelve binary variables given by Haenni and Lehmann in Haenni and Lehmann (2001). The memory load curve proposed in Figure A.2 shows a growth in $O(m^n)$, where n is the number of variables of both potentials' domains and m the cardinal of these variables.

A.3 Number of realizations per variable

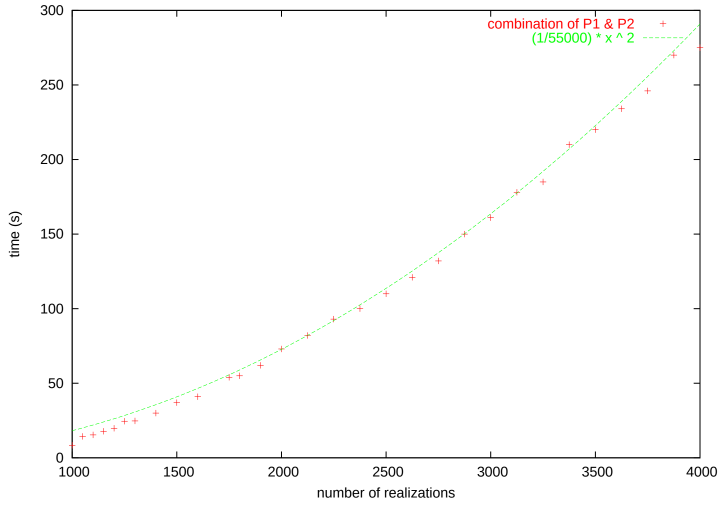

Figure A.3: Influence of the number of realizations per variable on computation time

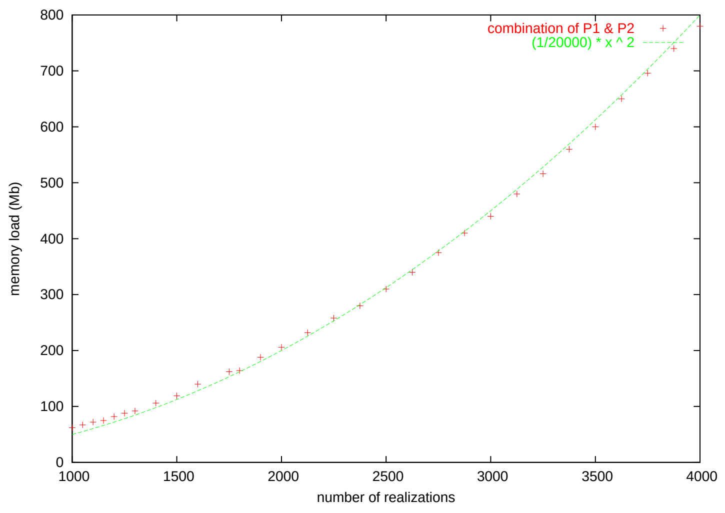

Figure A.4: Influence of the number of realizations per variable on memory load

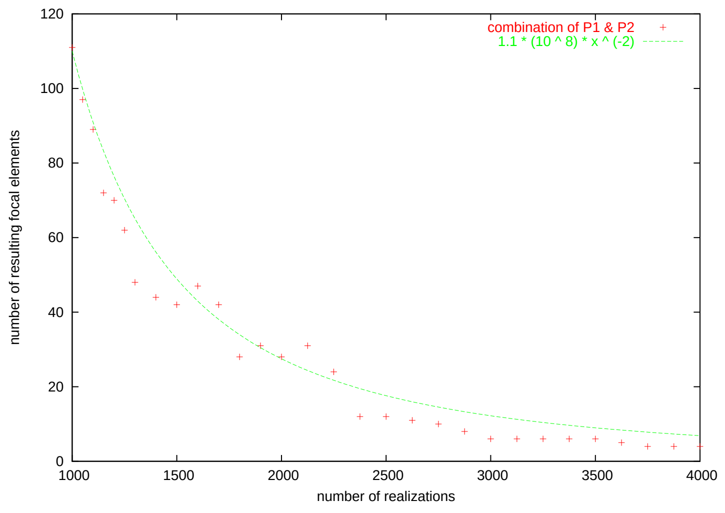

Figure A.5: Influence of the number of realizations per variable on the number of final focal elements

The influence of the number of realizations per variable is confirmed. With a domain of 2 variables for both potentials, the computation time and the memory load are globally squared with increasing the number of realizations (Figures A.3 and A.4). At the same time, with increasing the number of realizations per variable, the number of resulting focal elements is decreasing in $O(n^{-m})$, where n is the number of realizations per variable and m the number of variables (Figure A.5). The focal elements of the two potentials are randomly generated: they are chosen in a set of n^m possibilities (m variables of n realizations). Then, the decreasing number of final focal elements is not surprising since more intersections of focal elements are empty when increasing the number of realizations.

A.4 Number of focal elements per potential

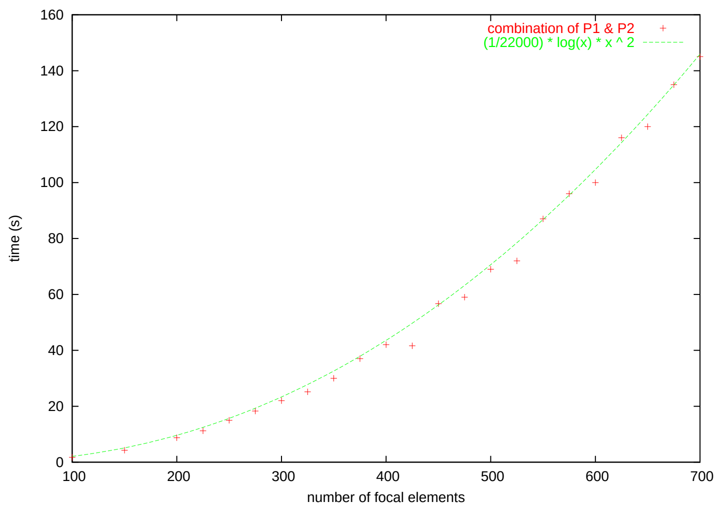

Figure A.6: Influence of the number of focal elements per potential on computation time

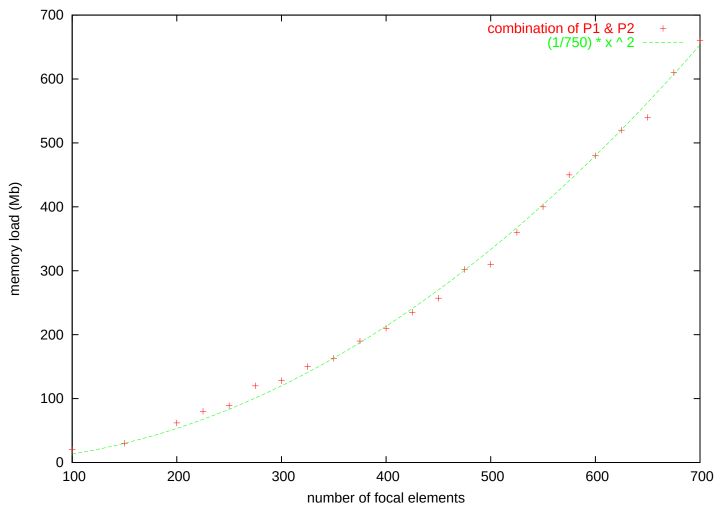

Figure A.7: Influence of the number of focal elements per potential on memory load

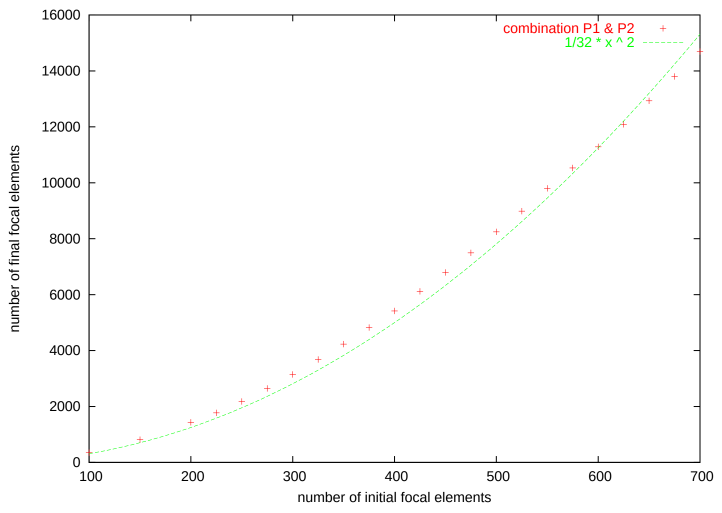

Figure A.8: Influence of the number of focal elements per potential on the number of final focal elements

The number of focal elements also induces an explosion of the memory and time factors. This is unavoidable since combination mechanisms are basically manipulations of focal elements. Figure A.6 shows a growth of the computation time in $O(n^2 \times log(n))$, where n is the number of initial focal elements of both potentials. This corresponds to the theoretical complexity of the combination algorithm proposed in Subsection 7.4.3, with potentials represented by balanced trees (see 6.13). The growth of the number of resulting focal elements is exposed in Figure A.8: it is a little bit slower than $O(n^2)$. The growth of the memory load follows basically the same trend (Figure A.7).

A.5 Number of configurations per focal element

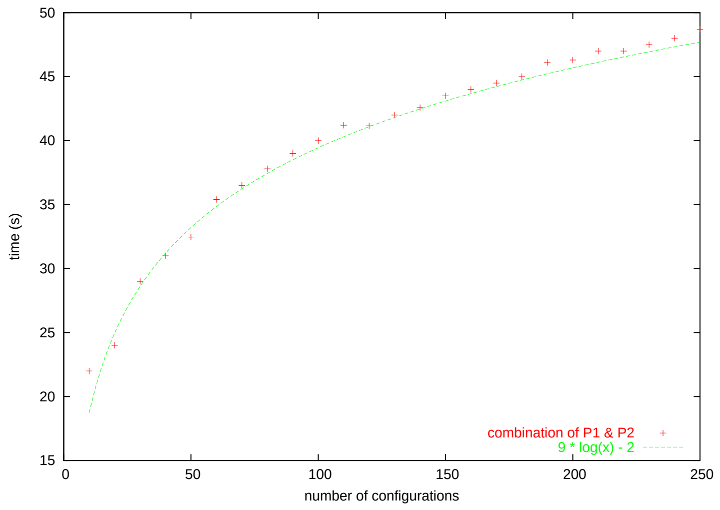

Figure A.9: Influence of the number of configurations on computation time

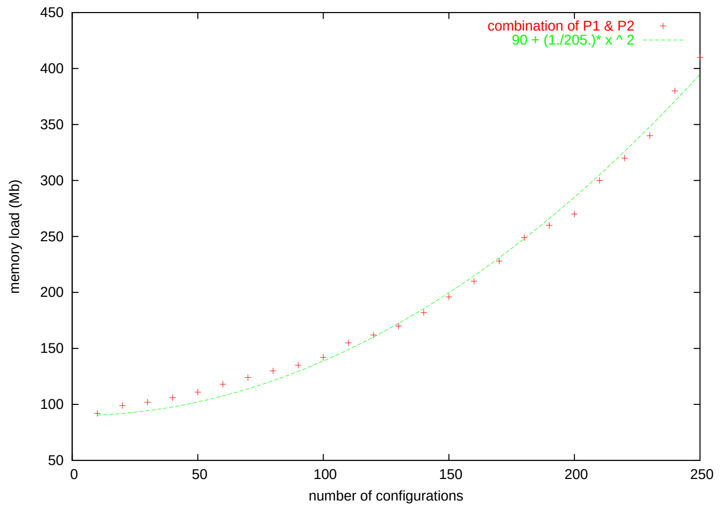

Figure A.10: Influence of the number of configurations on memory load

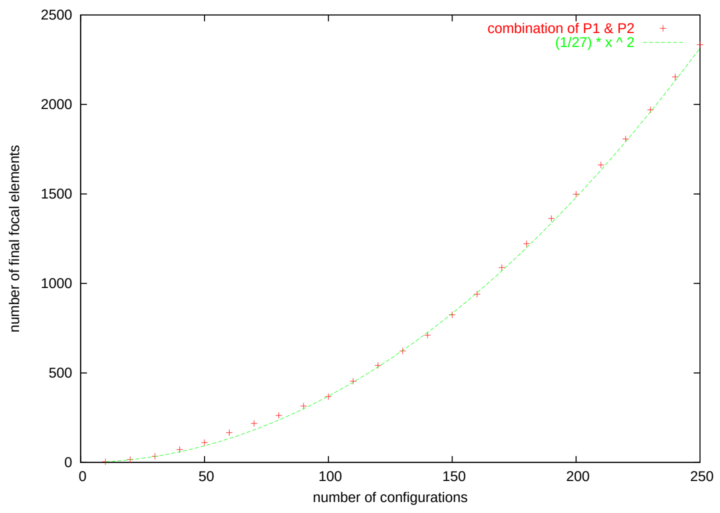

Figure A.11: Influence of the number of configurations on the number of final focal elements

Adding configurations in focal elements without changing the domain of variables does not change the size of the focal elements, if one uses the bitset representation (see 6.3.2). Nevertheless, it makes focal elements more complex, so that more intersections (i.e. more focal elements) are created during the combination. Basically, the number of resulting focal elements and the memory load grow in $O(n^2)$ with n the number of configurations per focal element (for P_1 and P_2). With a fixed number of initial focal elements, the variable time-consuming operation during the combination is the $update$ procedure: the resulting potential (i.e. the associated balanced tree) grows in $O(n^2)$ with n the number of configurations per focal element, so that the time needed for computing the $update$ procedure with a balanced tree grows in $O(log(n^2)) = O(log(n))$ (see Figure A.9).

A.6 On practical applicability of D-S formalism

This test bed showed significant explosions of the memory load and computation time w.r.t. all the factors we studied. These results globally confirm the theoretical complexities of D-S mechanisms. We will now sum up and discuss the real impact of these explosions.

A.6.1 Memory load and number of focal elements as limiting factors

The memory load is a critical point one has to take into account when performing D-S computations. For all the examples implemented thanks to `eVidenZ`, the memory load was the real limiting factor.

As it was already exposed, the memory load is highly correlated to the size of the focal elements. Using the bitset representation described in Subsection 6.3.2, the size of the focal elements is induced by the variable domain: the number of variables and the number of configurations per variable. Then, one can guess that problems defined on huge domains of variables, especially highly multi-variable domains, will cause important memory difficulties.

The second critical factor is the number of focal elements induced by the problem. With complex, numerous potentials, combination mechanisms can create such an explosion of resulting focal elements that the combination will not be practically computable because of memory problems. The number of focal elements after a combination relies on a lot of factors:

- "compatibility" between the two potentials that are combined: it is really common to combine conflicting potentials, so that the number of resulting focal elements is reduced;
- complexity (i.e. the number of configurations) of the initial focal elements: we saw that complex focal elements induce more intersections;
- regrouping factor: this factor can be really important, so that even if the two combined potentials are highly compatible, the number of resulting focal elements is reduced;

Combining all these factors, it is difficult to predict the evolution of the number of focal elements. This number can decrease or increase after some combinations, depending on the type of the problem.

A.6.2 Computation time as a limiting factor

This study confirms the time complexity of the combination mechanism w.r.t. the number of focal elements. Computation time becomes a really limiting factor when dealing with a lot of simple focal elements, i.e. when the problem implies lots of simple hypotheses. In practice, focal elements are usually defined on very simple domains, as we will explain it in the next subsection. Computation time can then become a real problem.

A.6.3 Real-scale applications

All the tests proposed in this chapter could seem alarming: they show memory and time explosions induced by increasing slowly some factors. This study confirms naturally the theoretical infeasibility of the D-S framework. However, one must realize that the tests performed are totally unreasonable when considering real-scale applications. As it was developed in the Chapter 5, the D-S formalism is usually applied to "simple" problems: the number of variables is generally very limited, problems are even rarely multi-variable. The number of configurations per variable, the number of initial focal elements are also much more reasonable than the values proposed in our tests.

Example of a real-scale application Consider, for example, the segmentation of satellite images presented in Appendix B.1. The seven layers induce seven potentials. There is only one variable, the $pixeltype$, which has basically less than hundred realizations (i.e. the number of segmentation classes): $\{water, building, trees, ...\}$. We consider 10 focal elements per potential of 10 configurations (i.e., every focal element brings information about 10 of the 100 segmentation classes). Combining the seven potentials takes about 0.004 seconds, for a memory load of about 1 Mb.

Real applications of the D-S theory are thus generally reasonable. Some propagation-based frameworks were developed to deal with more important systems (see Section 9), but one rarely needs such equipment. Even if D-S computations hide NP-complete problems, usual applications rarely exceed practical limitations. Then, by following the principles exposed in this report, these computations can be performed efficiently so that they become feasible in a reasonable time.

Appendix B

Research perspectives

These chapter give some explanations about our future research projects involving our D-S engine, `eVidenZ` (see Part III).

B.1 Satellite image segmentation

We dispose of satellite images composed of 7 channels/layers. Each layer is acquired on a given wave length, so that it provides a specific information. We have in fact different visions of the same scene. For example, the first layer could see water and building separately, where the second layer can not distinguish water and trees. For obtaining the final information, i.e. the nature of a given pixel, the underlying pieces of information of each layer must be merged.

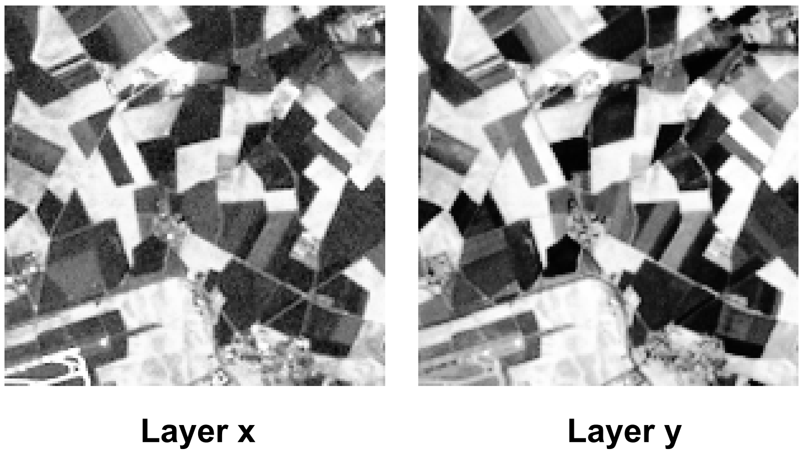

Figure B.1: Two layers of a satellite image

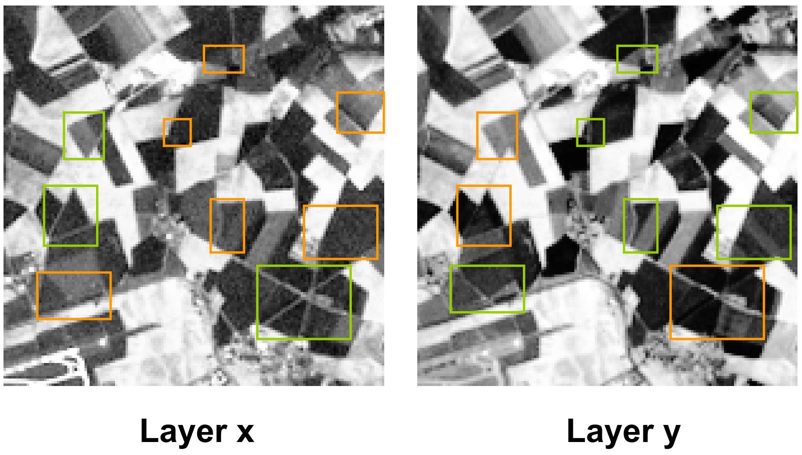

Figure B.2: Two layers of a satellite image (differences)

Figures B.1 and B.2 illustrate the type of problem we have to face. Some informations are clearly present on the layer X and do not appear on the layer X. The contrary is also true. Then, D-S theory seems relevant for modeling and merging these partial, sometimes fuzzy pieces of information. We basically obtain the following D-S modeling:

- 1 variable: $pixel\_type = \{forest, water, road, building, ...\}$;
- potentials = layers;
- pixel gray level (**context**) induces belief masses for pixel type sets;
  - ▷ combining layers leads to global belief for each pixel type.

Connected problems were introduced (and solved) in (Le Hégarat-Mascle et al., 1998) and (Vannoorenberghe et al., 1999). (See Sections 5.2 and 5.3).

B.2 Cancerous cell characterization

D-S theory will be applied to Adhoc[2] project, a cytology project by the LRDE (Epita Research and Development laboratory). Please refer to (LRDE, 2003).

We dispose of acquisitions of fine-needle aspiration done with the spot method. Figure B.3 presents one of these acquisitions.

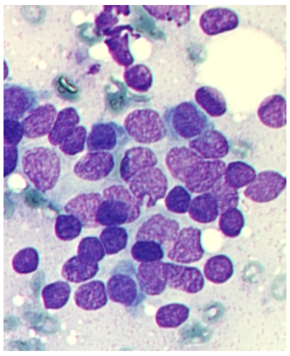

Figure B.3: Example of spot acquisition

Adhoc[2] project aims at analyzing these images in order to identify and characterize malignant cells. The goal is to reduce subjectivity due to omission or misinterpretation of important information carried by the cytological image and so provide a reproducible, justified diagnosis help. Experts apply currently subjective, fuzzy rules for analyzing the images. They basically study different cell characteristics:

- cell/nucleus size;
- cell/nucleus regularity;
- cell color...

Considering each cell characteristic individually, experts can express logical rules for explaining their belief. They could say, for example, that cancerous cells are generally bigger and less regular than normal ones. At the same time, they moderate these rules by explaining that a small, really irregular cell can be cancerous. Then, one can guess that every factor provides only partial information and that only merging these pieces of knowledge can lead to a possible decision.

We basically obtain the following D-S modeling:

- 1 variable: $cell\_type = \{normal, abnormal, not\_a\_cell, cancerous\}$;
- potentials = criters: size, regularity, color...
- cell characteristics induce belief masses for cell type sets;
  - ▷ combining criters leads to global belief for each cell type;

Our aim is currently to set curves like ones presented in Figures B.4 and B.5. In fact, these curves represent the belief functions for each potential/criterion. They associate belief masses to focal elements according to a given context (i.e. the criterion value). For instance, the 'size' curve proposed in Figure B.4 would give the belief that the cell studied is cancerous w.r.t. its size.

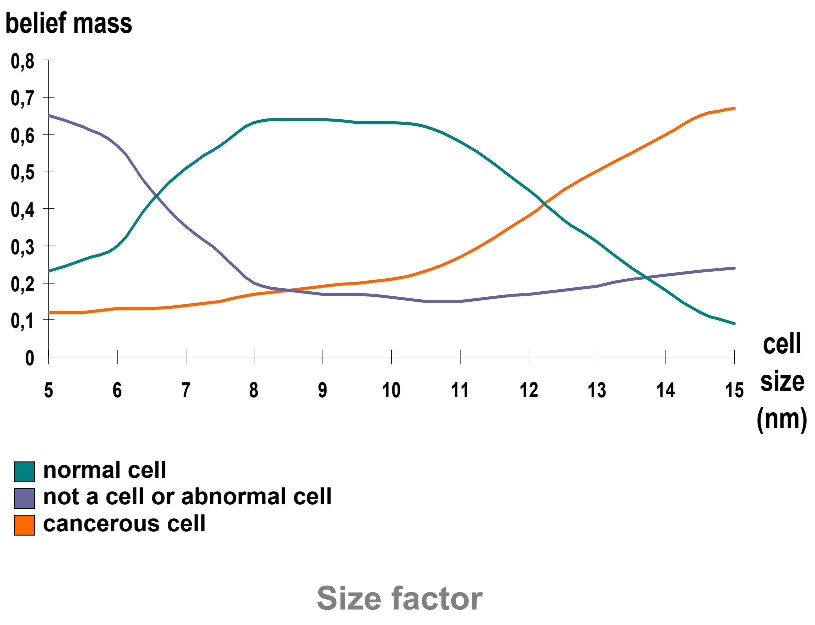

Figure B.4: Size factor and belief mass

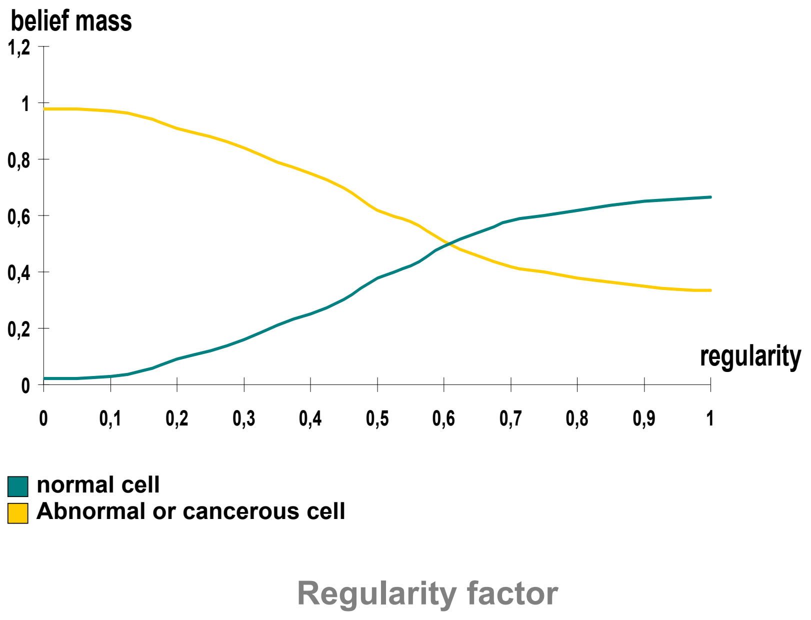

Figure B.5: Regularity factor and belief mass

We are currently evaluating the possibility of obtaining these curves thanks to human experts or by machine learning.

Anyway, D-S formalism seems particularly adapted to model the information fusion in this problem. D-S framework also provides justified decision and quantified ignorance, two really interesting properties with such medical diagnosis applications (see also section 5.4).