Arnaud Martin

ENSIETA E3I2-EA3876,
2 Rue François Verny, 29806 Brest Cedex 9, France.
Arnaud.Martin@ensieta.fr

Implementing general belief function framework with a practical codification for low complexity

Published in:
Florentin Smarandache & Jean Dezert (Editors)
Advances and Applications of DSmT for Information Fusion
(Collected works), Vol. III
American Research Press (ARP), Rehoboth, 2009
ISBN-10: 1-59973-073-1
ISBN-13: 978-1-59973-073-8
Chapter VII, pp. 217 - 273

Abstract: *In this chapter, we propose a new practical codification of the elements of the Venn diagram in order to easily manipulate the focal elements. In order to reduce the complexity, the eventual constraints must be integrated in the codification at the beginning. Hence, we only consider a reduced hyper-power set* D_r^Θ *that can be* 2^Θ *or* D^Θ*. We describe all the steps of a general belief function framework. The step of decision is studied in particular when we have to decide on intersections of the singletons of the discernment space no actual decision functions are easily to use. Hence, two approaches are proposed, an extension of previous one and an approach based on the specificity of the elements on which to decide. The principal goal of this chapter is to provide practical codes of a general belief function framework for the researchers and users needing the belief function theory.*

This work was carried out while the author was visiting DRDC (Defense Research and Development Canada) at Valcatier, Québec, Canada, and is partially supported by the DGA (*Délégation Générale pour l'Armement*) and by BMO (*Brest Métropole Océane*).

7.1 Introduction

Today the belief function theory initiated by [6, 26] is recognized to propose one of the more complete theories for human reasoning under uncertainty, and has been applied in many kinds of applications [32]. This theory is based on the use of functions defined on the power set 2^Θ (the set of all the subsets of Θ), where Θ is the set of considered elements (called *discernment space*), whereas the probabilities are defined only on Θ. A *mass function* or *basic belief assignment*, m is defined by the mapping of the power set 2^Θ onto $[0, 1]$ with:

$$\sum_{X \in 2^\Theta} m(X) = 1. \tag{7.1}$$

One element X of 2^Θ, such as $m(X) > 0$, is called *focal* element. The set of focal elements for m is noted $\mathcal{F}_m$. A mass function where Θ is a focal element, is called a *non-dogmatic* mass functions.

One of the main goals of this theory is the combination of information given by many experts. When this information can be written as a mass function, many combination rules can be used [23]. The first combination rule proposed by Dempster and Shafer is the normalized conjunctive combination rule given for two basic belief assignments m_1 and m_2 and for all $X \in 2^\Theta$, $X \neq \emptyset$ by:

$$m_{\text{DS}}(X) = \frac{1}{1-k} \sum_{A \cap B = X} m_1(A)m_2(B), \tag{7.2}$$

where $k = \sum_{A \cap B = \emptyset} m_1(A)m_2(B)$ is the inconsistency of the combination (generally called conflict).

However the high computational complexity, especially compared to the probability theory, remains a problem for more industrial uses. Of course, higher the cardinality of Θ is, higher the complexity becomes [38]. The combination rule of Dempster and Shafer is $\#P$-complete [25]. Moreover, when combining with this combination rule, non-dogmatic mass functions, the number of focal elements can not decrease.

Hence, we can distinguish two kinds of approaches to reduce the complexity of the belief function framework. First we can try to find optimal algorithms in order to code the belief functions and the combination rules based on Möbius transform [18, 33] or based on local computations [28] or to adapt the algorithms to particulars mass functions [3, 27]. Second we can try to reduce the number of focal elements by approximating the mass functions [4, 9, 16, 17, 36, 37], that could be particularly important for dynamic fusion.

In practical applications the mass functions contain at first only few focal elements [1, 7]. Hence it seems interesting to only work with the focal elements and not with the entire space 2^Θ. That is not the case in all general developed algorithms [18, 33].

Now if we consider the extension of the belief function theory proposed by [10], the mass function is defined on the extension of the power set into the hyper-power set D^Θ (that is the set of all the disjunctions and conjunctions of the elements of

Θ). This extension can be seen as a generalization of the classical approach (and it is also called DSmT for Dezert and Smarandache Theory [29, 30]). This extension is justified in some applications such as in [20, 21]. Try to generate D^Θ is not easy and becomes untractable for more than 6 elements in Θ [11].

In [12], a first proposition has been proposed to order elements of hyper-power set for matrix calculus such as [18, 33] made in 2^Θ. But as we said herein, in real applications it is better to only manipulate the focal elements. Hence, some authors propose algorithms considering only the focal elements [9, 15, 22]. In the previous volume [15, 30] have proposed MATLAB$^{\text{TM}}$ codes for DSmT hybrid rule. These codes are a preliminary work, but first it is really not optimized for MATLAB$^{\text{TM}}$ and second have been developed for a dynamic fusion.

MATLAB$^{\text{TM}}$ is certainly not the best program language to reduce the speed of processing, however most of people using belief functions do it with MATLAB$^{\text{TM}}$.

In this chapter, we propose a codification of the focal elements based on a codification of Θ in order to program easily in MATLAB$^{\text{TM}}$ a general belief function framework working for belief functions defined on 2^Θ but also on D^Θ.

Hence, in the following section we recall a short background of belief function theory. In section 7.3 we introduce our practical codification for a general belief function framework. In this section, we describe all the steps to fuse basic belief assignments in the order of necessity: the codification of Θ, the addition of the constraints, the codification of focal elements, the step of combination, the step of decision, if necessary the generation of a new power set: the *reduced hyper-power set* D_r^Θ and for the display, the decoding. We particularly investigate the step of the decision for the DSmT. In section 7.5 we give the major part of the MATLAB$^{\text{TM}}$ codes of this framework.

7.2 Short background on theory of belief functions

In the DSmT, the mass functions m are defined by the mapping of the hyper-power set D^Θ onto $[0, 1]$ with:

$$\sum_{X \in D^\Theta} m(X) = 1. \tag{7.3}$$

In the more general model, we can add constraints on some elements of D^Θ, that means that some elements can never be focal elements. Hence, if we add the constraints that all the intersections of elements of Θ are impossible (*i.e.* empty) we recover 2^Θ. So, the constraints given by the application can drastically reduce the number of possible focal elements and so the complexity of the framework. On the contrary of the suggestion given by the flowchart on the cover of the book [29] and the proposed codes in [15], we think that the constraints must be integrated directly in the codification of the focal elements of the mass functions as we shown in section 7.3. Hereunder, the hyper-power set D^Θ taking into account the constraints is called the *reduced hyper-power set* and noted D_r^Θ. Hence, D_r^Θ can be D^Θ, 2^Θ, have a cardinality between these two power sets or inferior to these two power sets. So the normality condition is given by:

$$\sum_{X \in D_r^\Theta} m(X) = 1, \tag{7.4}$$

where we consider less terms in the sum than in the equation (7.3).

Once the mass functions coming from numerous sources are defined, many combination rules are possible (see [5, 20, 23, 31, 35] for recent reviews of the combination rules). Most of the combination rules are based on the conjunctive combination rule, given for mass functions defined on 2^Θ by:

$$m_c(X) = \sum_{Y_1 \cap \ldots \cap Y_s = X} \prod_{j=1}^{s} m_j(Y_j), \tag{7.5}$$

where $Y_j \in 2^\Theta$ is the response of the source j, and $m_j(Y_j)$ the corresponding basic belief assignment. This rule is commutative, associative, not idempotent, and the major problem which the majority of the rules try to resolve is the increase of the belief on the empty set with the number of sources and the cardinality of Θ [19]. Now, in D^Θ without any constraint, there is no empty set, and the conjunctive rule given by the equation (7.5) for all $X \in D^\Theta$ with $Y_j \in D_r^\Theta$ can be used. If we have some constraints, we must transfer the belief $m_c(\emptyset)$ on other elements of the reduced hyper-power set. There is no optimal combination rule, and we cannot achieve this optimality for general applications.

The last step in a general framework for information fusion system is the decision step. The decision is also a difficult task because no measures are able to provide the best decision in all the cases. Generally, we consider the maximum of one of the three functions: credibility, plausibility, and pignistic probability. Note that other decision functions have been proposed [13].

In the context of the DSmT the corresponding generalized functions have been proposed [14, 29]. The generalized credibility Bel is defined by:

$$\mathrm{Bel}(X) = \sum_{Y \in D_r^\Theta, Y \subseteq X, Y \not\equiv \emptyset} m(Y) \tag{7.6}$$

The generalized plausibility Pl is defined by:

$$\mathrm{Pl}(X) = \sum_{Y \in D_r^\Theta, X \cap Y \not\equiv \emptyset} m(Y) \tag{7.7}$$

The generalized pignistic probability is given for all $X \in D_r^\Theta$, with $X \neq \emptyset$ is defined by:

$$\mathrm{GPT}(X) = \sum_{Y \in D_r^\Theta, Y \not\equiv \emptyset} \frac{\mathcal{C}_\mathcal{M}(X \cap Y)}{C_M(Y)} m(Y), \tag{7.8}$$

where $\mathcal{C}_\mathcal{M}(X)$ is the DSm cardinality corresponding to the number of parts of X in the Venn diagram of the problem [14, 29] associated with a model $\mathcal{M}$. Generally in

2^Θ, the maximum of these functions is taken on the elements in Θ. In this case, with the goal to reduce the complexity we only have to calculate these functions on the singletons. However, first, there exist methods providing decision on 2^Θ such as in [2] and that can be interesting in some application [24], and secondly, the singletons are not the more precise elements on D_r^Θ. Hence, to calculate these functions on the entire reduced hyper-power set could be necessary, but the complexity could not be inferior to the complexity of D_r^Θ and that can be a real problem if there are a few constraints.

7.3 A general belief function framework

We introduce here a practical codification in order to consider all the previous remarks to reduce the complexity:

- only manipulate focal elements,
- add constraints on the focal elements before combination, and so work on D_r^Θ,
- a codification easy for union and intersection operations with programs such as MATLAB$^{\text{TM}}$.

We first give the simple idea of the practical codification for enumerating the distinct parts of the Venn diagram and therefore a codification of the discernment space Θ. Then we explain how simply add the constraints on the distinct elements of Θ and how to do the codification of the focal elements. The subsections 7.3.4 and 7.3.5 show how to combine and decide with this practical codification, giving a particular reflexion on the decision in DSmT. The subsection 7.3.6 presents the generation of D_r^Θ and the subsection 7.3.7 the decoding.

7.3.1 A practical codification

The simple idea of the practical codification is based on the affectation of an integer number in $[1; 2^n - 1]$ to each distinct part of the Venn diagram that contains $2^n - 1$ distinct parts with $n = |\Theta|$. The figures 7.1 and 7.2 illustrate the codification for respectively $\Theta = \{\theta_1, \theta_2, \theta_3\}$ and $\Theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$ with the code given in section 7.5. Of course other repartitions of these integers are possible.

Hence, for example the element θ_1 is given by the concatenation of 1, 2, 3 and 5 for $|\Theta| = 3$ and by the concatenation of 1, 2, 3, 4, 6, 7, 9 and 12 for $|\Theta| = 4$. We will note respectively $\theta_1 = [1\ 2\ 3\ 5]$ and $\theta_1 = [1\ 2\ 3\ 4\ 6\ 7\ 9\ 12]$ for $|\Theta| = 3$ and for $|\Theta| = 4$, with increasing order of the integers. Hence, Θ is given respectively for $|\Theta| = 3$ and $|\Theta| = 4$ by:

$$\Theta = \{[1\ 2\ 3\ 5], [1\ 2\ 4\ 6], [1\ 3\ 4\ 7]\}$$

and

$$\Theta = \{[1\ 2\ 3\ 4\ 6\ 7\ 9\ 12], [1\ 2\ 3\ 5\ 6\ 8\ 10\ 13], [1\ 2\ 4\ 5\ 7\ 8\ 11\ 14], [1\ 3\ 4\ 5\ 9\ 10\ 11\ 15]\}.$$

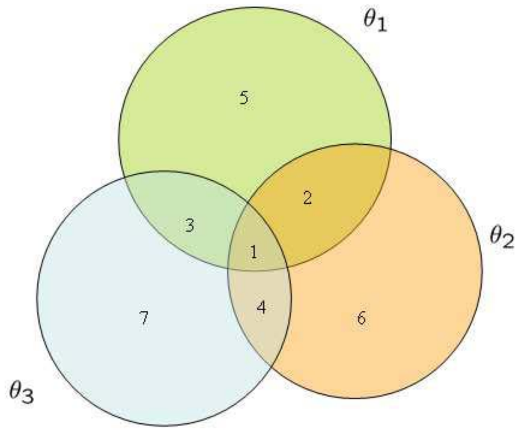


Figure 7.1: Codification for $\Theta = \{\theta_1, \theta_2, \theta_3\}$.

The number of integers for the codification of one element $\theta_i \in \Theta$ is given by:

$$1 + \sum_{i=1}^{n-1} C_{n-1}^{i}, \tag{7.9}$$

with $n = |\Theta|$ and C_n^p the number of p-uplets with n numbers. The number 1 will be always by convention be the intersection of all the elements of Θ. The codification of $\theta_1 \cap \theta_3$ is given by [1 3] for $|\Theta| = 3$ and [1 2 4 7] for $|\Theta| = 4$. And the codification of $\theta_1 \cup \theta_3$ is given by [1 2 3 4 5 7] for $|\Theta| = 3$ and [1 2 3 4 6 7 9 12] for $|\Theta| = 4$.

In order to reduce the complexity, especially using more hardware language than MATLAB$^{\text{TM}}$, we could use binary numbers instead of the integer numbers.

The Smarandache's codification [11], was introduced for the enumeration of distinct parts of a Venn diagram. If $|\Theta| = n$, $< i >$ denotes the part of θ_i with no covering with other θ_j, $i \neq j$. $< ij >$ denotes the part of $\theta_i \cap \theta_j$ with no covering with other parts of the Venn diagram. So if $n = 2$, $\theta_1 \cap \theta_2 = \{< 12 >\}$ and if $n = 3$, $\theta_1 \cap \theta_2 = \{< 12 >, < 123 >\}$, see the figure 7.3 for an illustration for $n = 3$. The authors note a problem for $n \geq 10$, but if we introduce space in the codification we can conserve integers instead of other symbols and we write $< 1\ 2\ 3 >$ instead of $< 123 >$.

Contrary to the Smarandache's codification, the proposed codification gives only one integer number to each part of the Venn diagram. This codification is more complex for the reader then the Smarandache's codification. Indeed, the reader can understand directly the Smarandache's codification thanks to the meaning of the numbers knowing the n: each disjoint part of the Venn diagram is seen as an inter-

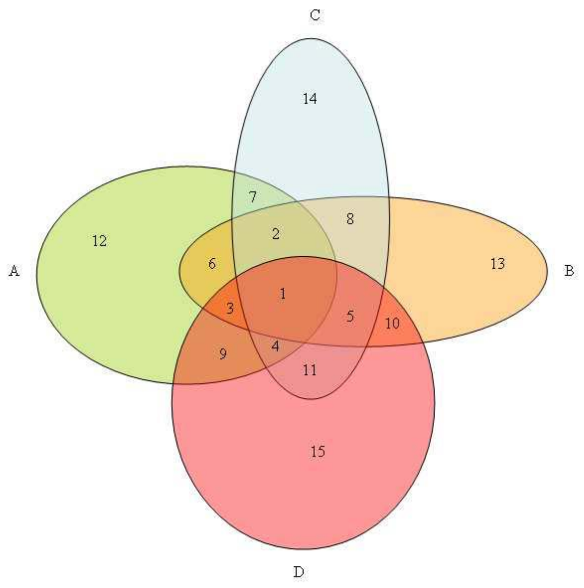


Figure 7.2: Codification for $\Theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$.

section of the elements of Θ. More exactly, this is a part of the intersections. For example, $\theta_1 \cap \theta_2$ is given with the Smarandache's codification by $\{< 12 >\}$ if $n = 2$ and by $\{< 12 >, < 123 >\}$ if $n = 3$. With the practical codification the same element has also different codification according to the number n. For the previous example $\theta_1 \cap \theta_2$ is given by [1] if $n = 2$, and by [1 2] if $n = 3$.

The proposed codification is more practical for computing union and intersection operations and the DSm cardinality, because only one integer represents one of the distinct parts of the Venn diagram. With Smarandache's codification computing union and intersection operations and the DSm cardinality could be very similar than with the practical codification, but adding a routine in order to treat the code of one part of the Venn diagram.

Hence, we propose to use the proposed codification to compute union, intersection and DSm cardinality, and the Smarandache's codification, easier to read, to present the results in order to save eventually a scan of D_r^Θ.

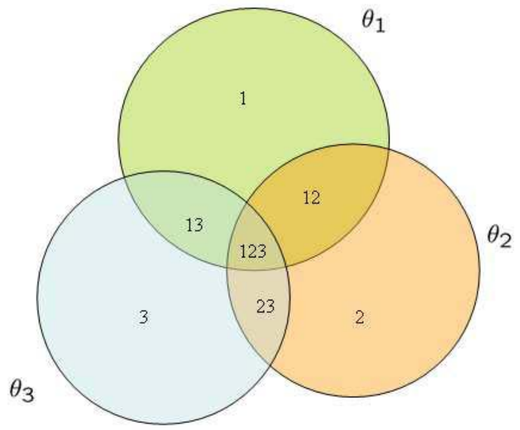


Figure 7.3: Smarandache's codification for $\Theta = \{\theta_1, \theta_2, \theta_3\}$.

7.3.2 Adding constraints

With this codification, adding constraints is very simple and can reduce rapidly the number of integers. For example assume that in a given application we know $\theta_1 \cap \theta_3 \equiv \emptyset$ (*i.e.* $\theta_1 \cap \theta_3 \notin D_r^\Theta$), that means that the integers [1 3] for $|\Theta| = 3$ and [1 2 4 7] for $|\Theta| = 4$ do not exist. Hence, the codification of Θ with the reduced discernment space, noted Θ_r, is given respectively for $|\Theta| = 3$ and $|\Theta| = 4$ by:

$$\Theta_r = \{[2\ 5], [2\ 4\ 6], [4\ 7]\}$$

and

$$\Theta_r = \{[3\ 6\ 9\ 12], [3\ 5\ 6\ 8\ 10\ 13], [5\ 8\ 11\ 14], [3\ 5\ 9\ 10\ 11\ 15]\}.$$

Generally we have $|\Theta| = |\Theta_r|$, but it is not necessary if a constraint gives $\theta_i \equiv \emptyset$, with $\theta_i \in \Theta$. This can happen in dynamic fusion, if one element of the discernment space can disappear.

Thereby, the introduction of the simple constraint $\theta_1 \cap \theta_3 \equiv \emptyset$ in Θ, includes all the other constraints that follow from it such as the intersection of all the elements of Θ is empty. In [15] all the constraints must be given by the user.

7.3.3 Codification of the focal elements

In D_r^Θ, the codification of the focal elements is given from the reduced discernment space Θ_r. The codification of an union of two elements of Θ is given by the concatenation of the codification of the two elements using Θ_r. The codification of an intersection of two elements of Θ is given by the common numbers of the codification of the two elements using Θ_r. In the same way, the codification of an union of two focal elements is given by the concatenation of the codification of the two focal elements and the codification of an intersection of two focal elements is given by the common numbers of the codification of the two focal elements. In fact, for union and intersection operations we only consider one element as the set of the numbers given in its codification.

Hence, with the previous example (we assume $\theta_1 \cap \theta_3 \equiv \emptyset$, with $|\Theta| = 3$ or $|\Theta| = 4$), if the following elements $\theta_1 \cap \theta_2$, $\theta_1 \cup \theta_2$ and $(\theta_1 \cap \theta_2) \cup \theta_3$ are some focal elements, there are coded for $|\Theta| = 3$ by:

$$\theta_1 \cap \theta_2 = [2],$$

$$\theta_1 \cup \theta_2 = [2\ 4\ 5\ 6],$$
$$(\theta_1 \cap \theta_2) \cup \theta_3 = [2\ 4\ 7],$$

and for $|\Theta| = 4$ by:

$$\theta_1 \cap \theta_2 = [3\ 6],$$
$$\theta_1 \cup \theta_2 = [3\ 5\ 6\ 8\ 9\ 10\ 12\ 13],$$
$$(\theta_1 \cap \theta_2) \cup \theta_3 = [3\ 5\ 6\ 8\ 11\ 14].$$

The DSm cardinality $\mathcal{C}_{\mathcal{M}}(X)$ of one focal element X is simply given by the number of integers in the codification of X. The DSm cardinality of one singleton is given by the equation (7.9), only if there is no constraint on the singleton, and is inferior otherwise.

The previous example with the focal element $(\theta_1 \cap \theta_2) \cup \theta_3$ illustrates well the easiness to deal with the brackets in one expression. The codification of the focal elements can be made with any brackets.

7.3.4 Combination

In order to manage only the focal elements and their associated basic belief assignment, we can use a list structure [9, 15, 22]. The intersection and union operations between two focal elements coming from two mass functions are made as described before. If the intersection between two focal elements is empty the associated codification is [ ]. Hence the conjunctive combination rule algorithm can be done by algorithm 1. The disjunctive combination rule algorithm is exactly the same by changing $\cap$ in $\cup$.

Once again, the interest of the codification is for the intersection and union operations. Hence in MATLAB$^{\text{TM}}$, we do not need to redefine these operations as in [15].

For more complicated combination rules such as PCR6, we have generally to conserve the intermediate calculus in order to transfer the partial conflict. Algorithms for these rules have been proposed in [22], and MATLAB$^{\text{TM}}$ codes are given in section 7.5.

Algorithm 1: Conjunctive rule

```
Data: $n$ experts $ex$: $ex[1]\dots ex[n]$, $ex[i].focal$, $ex[i].bba$
Result: Fusion of $ex$ by conjunctive rule: $conj$
$extmp \leftarrow ex[1]$;
for $e = 2$ to $n$ do
    $comb \leftarrow \emptyset$;
    foreach $foc1$ in $extmp.focal$ do
        foreach $foc2$ in $ex[e].focal$ do
            $tmp \leftarrow extmp.focal(foc1) \cap ex[e].focal(foc2)$;
            $comb.focal \leftarrow tmp$;
            $comb.bba \leftarrow extmp.bba(foc1) \times ex[e].bba(foc2)$;
    Concatenate same focal in $comb$;
    $extmp \leftarrow comb$;
$conj \leftarrow extmp$;
```

7.3.5 Decision

As we wrote before, we can decide with one of the functions given by the equations (7.6), (7.7), or (7.8). These functions are increasing functions. Hence generally in 2^Θ, the decision is taken on the elements in Θ by the maximum of these functions. In this case, with the goal to reduce the complexity, we only have to calculate these functions on the singletons. However, first, we can provide a decision on any element of 2^Θ such as in [2] that can be interesting in some applications [24], and second, the singletons are not the more precise or interesting elements on D_r^Θ. The figures 7.4 and 7.5 show the DSm cardinality $\mathcal{C}_\mathcal{M}(X)$, $\forall X \in D^\Theta$ with respectively $|\Theta| = 3$ and $|\Theta| = 4$. The specificity of the singletons (given by the DSm cardinality) appears at a central position in the set of the specificities of the elements in D^Θ.

Hence, to calculate these decision functions on all the reduced hyper-power set could be necessary, but the complexity could not be inferior to the complexity of D_r^Θ and that can be a real problem. The more reasonable approach is to consider either only the focal elements or a subset of D_r^Θ on which we calculate decision functions.

7.3.5.1 Extended weighted approach

Generally in 2^Θ, the decisions are only made on the singletons [8, 34], and only a few approaches propose a decision on 2^Θ. In order to provide decision on any elements of D_r^Θ, we can first extend the principle of the proposed approach in [2] on D_r^Θ. This approach is based on the weighting of the plausibility with a Bayesian mass function taking into account the cardinality of the elements of 2^Θ.

In a general case, if there is no constraint, the plausibility is not interesting because all elements contain the intersection of all the singletons of Θ. According to

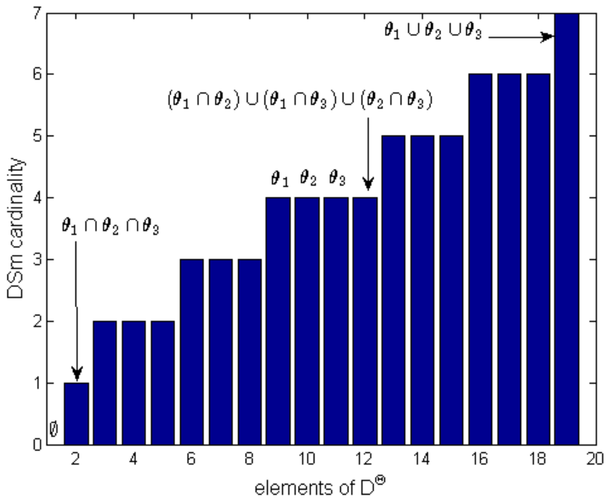


Figure 7.4: DSm cardinality $\mathcal{C}_\mathcal{M}(X)$, $\forall X \in D^\Theta$ with $|\Theta| = 3$.

the constraints the plausibility could be used.

Hence, we generalize here the weighted approach to D_r^Θ for every decision function f_d (plausibility, credibility, pignistic probability, ...). We note f_{wd} the weighted decision function given for all $X \in D_r^\Theta$ by:

$$f_{wd}(X) = m_d(X) f_d(X), \tag{7.10}$$

where m_d is a basic belief assignment given by:

$$m_d(X) = K_d \lambda_X \left(\frac{1}{\mathcal{C}_\mathcal{M}(X)^s} \right), \tag{7.11}$$

s is a parameter in $[0, 1]$ allowing a decision from the intersection of all the singletons ($s = 1$) (instead of the singletons in 2^Θ) until the total indecision Θ ($s = 0$). λ_X allows the integration of the lack of knowledge on one of the elements X in D_r^Θ. The constant K_d is the normalization factor giving by the condition of the equation (7.4). Thus we decide the element A:

$$A = \arg\max_{X \in D_r^\Theta} f_{wd}(X), \tag{7.12}$$

If we only want to decide on whichever focal element of D_r^Θ, we only consider $X \in \mathcal{F}_m$ and we decide:

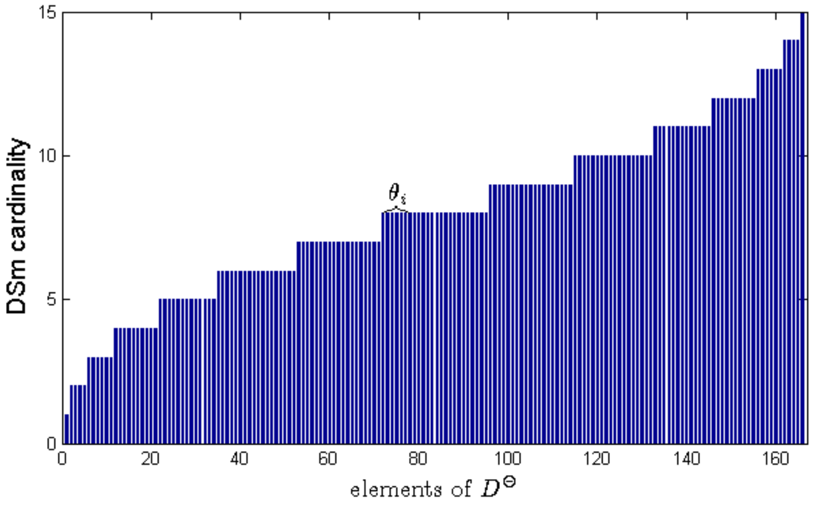


Figure 7.5: DSm cardinality $C_M(X)$, $\forall X \in D^\Theta$ with $|\Theta| = 4$.

$$A = \arg\max_{X \in \mathcal{F}_m} f_{wd}(X), \tag{7.13}$$

with f_{wd} given by the equation (7.10) and:

$$m_d(X) = K_d \lambda_X \left(\frac{1}{\mathcal{C}_\mathcal{M}(X)^s} \right), \, \forall X \in \mathcal{F}_m, \tag{7.14}$$

s and K_d are the two parameters defined above.

7.3.5.2 Decision according to the specificity

The cardinality $\mathcal{C}_\mathcal{M}(X)$ can be seen as a specificity measure of X. The figures 7.4 and 7.5 show that for a given specificity there is different kind of elements such as singletons, unions of intersections or intersections of unions. The figure 7.6 shows well the central role of the singletons (the DSm cardinality of the singletons for $|\Theta|$=5 is 16), but also that there are many other elements (619) with exactly the same cardinality. Hence, it could be interesting to precise the specificity of the elements on which we want to decide. This is the role of s in the Appriou approach. Here we propose to directly give the wanted specificity or an interval of the wanted specificity

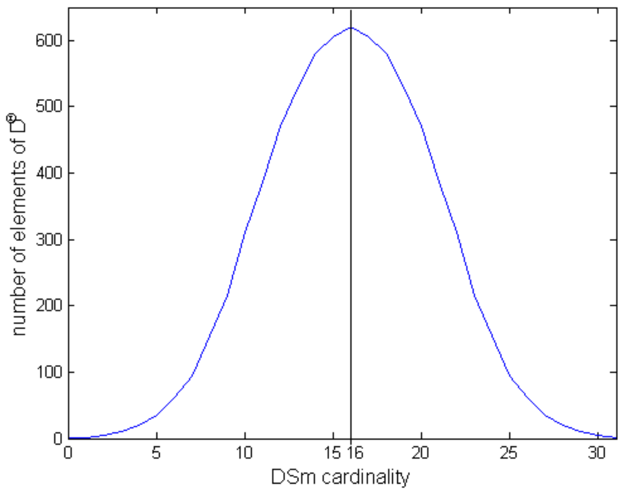


Figure 7.6: Number of elements of D^Θ for $|\Theta| = 5$, with the same DSm cardinality.

in order to build the subset of D_r^Θ on which we calculate decision functions. Thus we decide the element A:

$$A = \underset{X\in\mathcal{S} f_d(X),}{\arg\max} \tag{7.15}$$

where f_d is the chosen decision function (credibility, plausibility, pignistic probability, ...) and

$$\mathcal{S} = \left\{X \in D_r^\Theta ; min_S \le \mathcal{C}_\mathcal{M}(X) \le max_S\right\}, \tag{7.16}$$

with min_S and max_S respectively the minimum and maximum of the specificity of the wanted elements. If $min_S \neq max_S$, if have to chose a pondered decision function for f_d such as f_{wd} given by the equation (7.10).

However, in order to find all $X \in \mathcal{S}$ we must scan D_r^Θ. To avoid to scan all D_r^Θ, we have to find the cardinality of $\mathcal{S}$, but we can only calculate an upper bound of the cardinality, unfortunately never reached. Let us define the number of elements of the Venn diagram n_V. This number is given by:

$$n_V = \mathcal{C}_\mathcal{M}\left(\bigcup_{i=1}^{n} \theta_i\right), \tag{7.17}$$

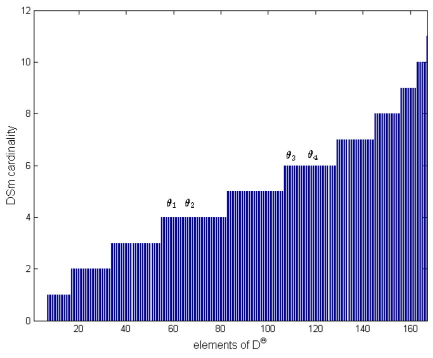
where n is the cardinality of Θ_r and $\theta_i \in \Theta_r$. Recall that the DSm cardinality is simply given by the number of integers of the codification. The upper bound of the cardinality of $\mathcal{S}$ is given by:

$$|\mathcal{S}| < \sum_{s=min_S}^{max_S} C_{n_V}^{s}, \tag{7.18}$$

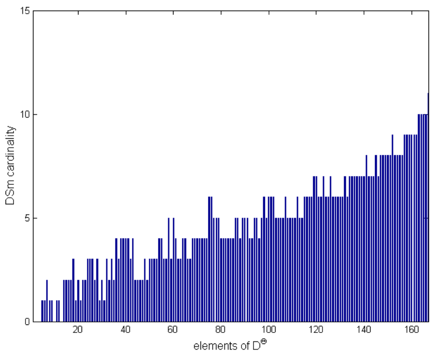
where $C_{n_V}^{s}$ is the number of combinations of s elements among n_V. Note that it also works if $min_S = 0$ for the empty set.

7.3.6 Generation of D_r^Θ

The generation of D_r^Θ could have the same complexity than the generation of D^Θ if there is no constraint given by the user. Today, the complexity of the generation of D^Θ is the complexity of the proposed code in [11]. Assume for example, the simple constraint $\theta_1 \cap \theta_2 \equiv \emptyset$. First, the figures 7.7(a) and 7.7(b) show the DSm cardinality for the elements of D_r^Θ with $|\Theta| = 4$ and the previous given constraint. On the left part of the figure, the elements are ordered by increasing DSm cardinality and on the right part of the figure with the same order as the figure 7.5. We can observe that the cardinality of the elements have naturally decreased and the number of non empty elements also. This is more interesting if the cardinality of Θ is higher. Figure 7.8 presents for a given positive DSm cardinality, the number of elements of D_r^Θ for $|\Theta| = 5$ and with the same constraint $\theta_1 \cap \theta_2 \equiv \emptyset$. Compared to figure 7.6, the total number of non empty elements (the integral of the curve) is considerably lower.



(a) Elements are ordered by increasing DSm cardinality.



(b) Elements are ordered with the same order than the figure 7.5.

Figure 7.7: DSm cardinality $\mathcal{C}_\mathcal{M}(X)$, $\forall X \in D_r^\Theta$ with $|\Theta| = 4$ and $\theta_1 \cap \theta_2 \equiv \emptyset$.

Thus, we have to generate D_r^Θ and not D^Θ. The generation of D^Θ (see [11] for more details) is based on the generation of monotone boolean functions. A monotone boolean function f_{mb} is a mapping of $(x_1, ..., x_b) \in \{0,1\}^b$ to a single binary output such as $\forall x, x' \in \{0,1\}^b$, with $x \preccurlyeq x'$ then $f_{mb}(x) \leq f_{mb}(x')$. Hence, a monotone boolean function is defined by the values of the 2^b elements $(x_1, ..., x_b)$, and there is $|D^b|$ different monotone boolean functions. All the values of all these monotone boolean functions can be represented by a $|D^b| \times 2^b$ matrix. If we multiply this matrix by the vector of all the possible intersections of the singletons in Θ with $|\Theta| = b$ (there are 2^b intersections) given an union of intersections, we obtain all the elements of D^Θ. We can also use the basis of all the unions of Θ (and obtain the intersections of unions), but with our codification the unions are coded with more integer numbers. So, the intersection basis is preferable.

Moreover, if we have some constraints (such as $\theta_1 \cap \theta_2 \equiv \emptyset$), some elements of the intersection basis can be empty. So we only need to generate a $|D^b| \times n_b$ matrix where n_b is the number of non empty intersections of elements in Θ_r. For example, with the constraint given in example for $|\Theta| = 3$, the basis is given by: $\emptyset$, θ_1, θ_2, θ_3, $\theta_1 \cap \theta_3$, $\theta_2 \cap \theta_3$, and there are no $\theta_1 \cap \theta_2$ and $\theta_1 \cap \theta_2 \cap \theta_3$.

Hence, the generation of D_r^Θ can run very fast if the basis is small, *i.e.* if there are some constraints. The MATLAB$^{\text{TM}}$ code is given in section 7.5.

7.3.7 Decoding

Once the decision on one element A of D_r^Θ is taken, we have to transmit this decision to the human operator. Hence we must to decode the element A (given by the integer numbers of the codification) in terms of unions and intersections of elements of Θ. If we know that A is in a subset of elements of D_r^Θ given by the operator, we only have to scan this subset. Now, if the decision A comes from the focal elements (*a priori* unknown) or from all the elements of D_r^Θ we must scan all D_r^Θ with possibly high complexity. What we propose here is to consider the elements of D_r^Θ ordering with first the elements most encountered in applications. Hence, we first scan the elements of 2^Θ and in the same time the intersection basis that we must build for the generation of D_r^Θ. Then, only if the element is not found we generate D_r^Θ and stop the generation when found (see the section 7.5 for more details).

Smarandache's codification is an alternative to the decoding because the user can directly understand it. Hence we can represent the focal element as an union of the distinct part of the Venn diagram. Smarandache's codification allows a clear understanding of the different parts of the Venn diagram unlike the proposed codification. This representation of the results (for the combination or the decision) does not need the generation of D_r^Θ. However, if we need to generate D_r^Θ according to the strategy of decision, the decoding will give a better display without more generation of D_r^Θ.

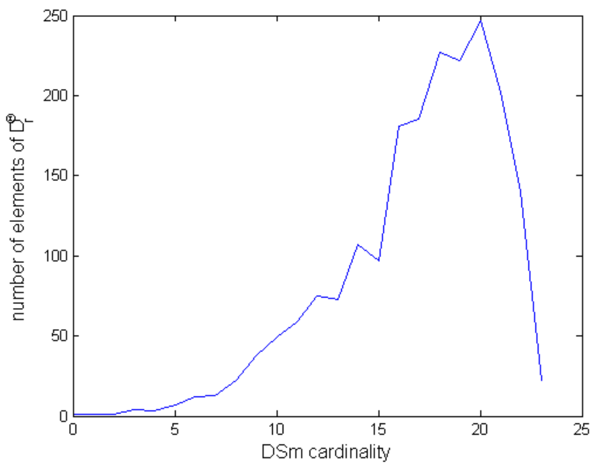


Figure 7.8: Number of elements of D_r^Θ for $|\Theta| = 5$ and $\theta_1 \cap \theta_2 \equiv \emptyset$, with the same positive DSm cardinality.

7.4 Concluding remarks

This chapter presents a general belief function framework based on a practical codification of the focal elements. First the codification of the elements of the Venn diagram gives a codification of Θ. Then, the eventual constraints are integrated giving a reduced discernment space Θ_r. From the space Θ_r, we obtain the codification of the focal elements. Hence, we manipulate elements of a reduced hyper-power set D_r^Θ and not the complete hyper-power set D^Θ, reducing the complexity according to the kind of given constraints.

With the practical codification, the step of combination is easily made using union and intersection functions.

The step of decision was particularly studied, because of the difficulties to decide on D^Θ or D_r^Θ. An extension of the approach given in [2] in order to give the possibility to decide on the unions in 2^Θ was proposed. Another approach based on the specificity was proposed in order to simply choose the elements on which to decide according to their specificity.

The principal goal of this chapter is to provide practical codes of a general belief function framework for the researchers and users needing the belief function theory. However, for sake of clarity, all the MATLAB$^{\text{TM}}$ codes are not in the listing, but can be provided on demand to the author. The proposed codes are not optimized either for MATLAB$^{\text{TM}}$, or in general and can still have bugs. All suggestions in order to improve them are welcome.

7.5 MATLAB™ codes

We give and explain here some MATLAB™ codes of the general belief function framework[1]. Note that the proposed codes are not optimized either for MATLAB™, or in general.

First the human operator has to describe the problem (see function 1) giving the cardinality of Θ, the list of the focal elements and the corresponding bba for each experts, the eventual constraints (‘ ’ if there is no constraint), the list of elements on which he wants to obtain a decision and the parameters corresponding to the choice of combination rule, the choice of decision criterion the mode of fusion (static or dynamic) and the display. When this is done, he just has to call the fuse function 2.

Function 1. - *Command configuration*

```
% description of the problem
CardTheta=4; % cardinality of Theta
% list of experts with focal elements and associated bba
    expert(1).focal={'1' '1u3' '3' '1u2u3'};
    expert(1).bba=[0.5421 0.2953 0.0924 0.0702];

    expert(2).focal={'1' '2' '1u3' '1u2u3'};
    expert(2).bba=[0.2022 0.6891 0.0084 0.1003];

    expert(3).focal={'1' '3n4' '1u2u3'};
    expert(3).bba=[0.2022 0.6891 0.1087];

constraint={'1n2' '1n3' '2n3'}; % set of empty elements
elemDec={'F'}; % set of decision elements

%-------------------------------------------------------------
% parameters
criterionComb=1; % combination criterion
criterionDec=0; % decision criterion
mode='static'; % mode of fusion
display=3; % kind of display

%-------------------------------------------------------------
% fusion

fuse(expert,constraint,CardTheta,criterionComb,criterionDec,...
mode,elemDec,display)
```

[1]Copyright © 2009 Arnaud Martin. May be used free of charge for non commercial products. Selling without prior written consent is prohibited. Obtain permission before redistributing.

The first step of the fuse function 2 is the coding. The cardinality of Θ gives the codification of the singletons of Θ, thanks to the function 3, then we add the constraints to Θ with the function 4 and obtain Θ_r. With Θ_r, the function 6 calling the function 5 codes the focal elements of the experts given by the human operator. The combination is made by the function 7 in static mode. For dynamic fusion, we just consider one expert with the previous combination. In this case the order of the experts given by the user can have an important signification. The decision step is made with the function 11. The last step concerns the display and the hard problem of the decoding. Thus, 4 choices are possible: no display, the results of the combination only, the results of decision only and both results. These displays could take a long time according to the parameters given by the human operator. Hence, the results of the combination could have the complexity of the generation of D_r^Θ and must be avoided if the user does not need it. The complexity of the decision results could also be high if the user does not give the exact set of elements on which to decide, or only the singletons with 'S' or on 2^Θ with '2T'. In other cases, with luck, the execution time can be short thanks to the function 18.

Function 2. - *Fuse function*

```
function fuse(expert,constraint,n,criterionComb,criterionDec,...
...mode,elemDec,display)

% To fuse experts' opinions
%
% fuse(expert,constraint,n,criterionComb,criterionDec,mode,...
% ...elemDec,display)
%
% Inputs:
% expertC = contains the structure of the list of coded focal
% elements and corresponding bba for all the experts
% constraint = the empty elements
% elemDec = list of elements on which we can decide
% n = size of the discernment space
% criterionComb = is the combination criterion
%         criterionComb=1 Smets criterion
%         criterionComb=2 Dempster-Shafer criterion (normalized)
%         criterionComb=3 Yager criterion
%         criterionComb=4 disjunctive combination criterion
%         criterionComb=5 Florea criterion
%         criterionComb=6 PCR6
%         criterionComb=7 Mean of the bbas
%         criterionComb=8 Dubois criterion (normalized and
%                                  disjunctive combination)
%         criterionComb=9 Dubois and Prade criterion
% (mixt combination)
%         criterionComb=10 Mixt Combination
% (Martin and Osswald criterion)
%         criterionComb=11 DPCR (Martin and Osswald criterion)
%         criterionComb=12 MDPCR (Martin and Osswald criterion)
%         criterionComb=13 Zhang's rule
%
%
% criterionDec = is the combination criterion
%           criterionDec=0 maximum of the bba
%           criterionDec=1 maximum of the pignistic probability
%           criterionDec=2 maximum of the credibility
%           criterionDec=3 maximum of the credibility with reject
%           criterionDec=4 maximum of the plausibility
%           criterionDec=5 Appriou criterion
%           criterionDec=6 DSmP criterion
%
% mode = 'static' or 'dynamic'
% elemDec = list of elements on which we can decide,
%     or A for all, S for singletons only, F for focal elements
%     only, SF for singleton plus focal elements, Cm for given
%     specificity, 2T for only 2^Theta (DST case)
% display = kind of display
%       display = 0 for no display,
%       display = 1 for combination display,
%       display = 2 for decision display,
%       display = 3 for both displays,
%       display = 4 for both displays with Smarandache
%                       codification
%
% Output:
% res = contains the structure of the list of focal elements and
% corresponding bbas for the combinated experts
%
% Copyright (c) 2008 Arnaud Martin

% Coding
[Theta,Scod]=codingTheta(n);
ThetaRed=addConstraint(constraint,Theta);

expertCod=codingExpert(expert,ThetaRed);

%--------
switch nargin
    case 1:5
```

```
        mode='static';
        elemDec=ThetaRed;
        display=4;
    case 6
        elemDec=ThetaRed;
        display=4;
    case 7
        elemDec=string2code(elemDec);
        display=4;
end
%--------
if (display==1) || (display==2) || (display==3)
    [DThetar,D_n]=generationDThetar(ThetaRed);
else
    switch elemDec{1}
        case {'A'}
            [DThetar,D_n]=generationDThetar(ThetaRed);
        otherwise
            DThetar.s={[]};
            DThetar.c={[]};
    end
end
%--------

% Combination
if strcmp(mode, 'static')
    [expertComb]=combination(expertCod,ThetaRed,criterionComb);
else % dynamic case
    nbexp=size(expertCod,2);
    expertTmp(1)=expertCod(1);
    for exp=2:nbexp
        expertTmp(2)=expertCod(exp);
        expertTmp(1)=combination(expertTmp,ThetaRed,...
                    ...criterionComb);
    end
    expertComb=expertTmp(1);
end

% Decision
[decFocElem]=decision(expertComb,ThetaRed,DThetar.c,...
...criterionDec,elemDec);

% Display
switch display
```

```
case 0
    'no display'
case 1
    % Result of the combination
    sFocal=size(expertComb.focal,2);
    focalRec=decodingExpert(expertComb,ThetaRed,DThetar);
    focal=code2string(focalRec)
    for i=1:sFocal
        disp ( [ focal{i},'=',num2str(expertComb.bba(i)) ] )
    end
case 2
    % Result of the decision
    if isstruct(decFocElem)
        focalDec=decodingFocal(decFocElem.focal,elemDec,...
          ...ThetaRed);
        disp(['decision:',code2string(focalDec)])
    else
        if decFocElem==0
            disp(['decision: rejected'])
        else
            if decFocElem==-1
                disp(['decision: cannot be taken'])
            end
        end
    end
case 3
    % Result of the combination
    sFocal=size(expertComb.focal,2);
    expertDec=decodingExpert(expertComb,ThetaRed,DThetar);
    focal=code2string(expertDec.focal)
    for i=1:sFocal
        disp ( [ focal{i},'=',num2str(expertDec.bba(i)) ] )
    end
    % Result of the decision
    if isstruct(decFocElem)
        focalDec=decodingFocal(decFocElem.focal,elemDec,...
                 ...ThetaRed,DThetar);
        disp(['decision:',code2string(focalDec)])
    else
        if decFocElem==0
            disp(['decision: rejected'])
        else
            if decFocElem==-1
                disp(['decision: cannot be taken'])
                end
            end
        end
    case 4
        % Results with Smarandache codification display
        % Result of the combination
        sFocal=size(expertComb.focal,2);
        expertDec=cod2ScodExpert(expertComb,Scod);
        for i=1:sFocal
            disp ([expertDec.focal{i},'=',...
                   ...num2str(expertDec.bba(i))])
        end
        % Result of the decision
         if isstruct(decFocElem)
            focalDec=cod2ScodFocal(decFocElem.focal,Scod);
            disp(['decision:',focalDec])
        else
            if decFocElem==0
                disp(['decision: rejected'])
            else
                if decFocElem==-1
                    disp(['decision: cannot be taken'])
                end
            end
        end

    otherwise
        'Accident in fuse: choice of display is uncorrect'
end
```

7.5.1 Codification

The codification is based on the function 3. The order of the integer numbers could be different, here the choice is made to number the intersection of all the elements with 1 and the smallest integer among the $|\Theta| = n$ bigger integers for the first singleton. At the same time this function gives the correspondence between the integer numbers of the practical codification and Smarandache's codification. This function 3 is based on the MATLAB$^{\text{TM}}$ function *nchoosek(tab,k)* given the array of all the combination of k elements of the vector *tab*. If the length of *tab* is n, this function return an array of C_n^k rows and k columns.

Function 3. - *codingTheta function*

```
function [Theta,Scod]=codingTheta(n)
```

```
% Code Theta for DSmT framework
%
% [Theta,Scod]=codingTheta(n)
%
% Input:
% n = cardinality of Theta
%
% Outputs:
% Theta = the list of coded elements in Theta
% Scod = the bijection function between the integer of
%   the coded elements in Theta and the Smarandache codification
%
% Copyright (c) 2008 Arnaud Martin

i=2^n-1;
tabInd=[];
for j=n:-1:1
    tabInd=[tabInd j];
    Theta{j}=[i];

    Scod{i}=[j];
    i=i-1;
end

i=i+1;
for card=2:n
    tabPerm=nchoosek(tabInd,card);
    for j=1:n
        [l,c]=find(tabPerm==j);
        tabi=i.*ones(1,size(l,1));
        Theta{j}=[sort(tabi-l') Theta{j}];
        for nb=1:size(l,1)
            Scod{i-l(nb)}=[Scod{i-l(nb)} j];
        end
    end
    i=i-size(tabPerm,1);
end
```

The addition of the constraints is made in two steps: first the codification of the elements in the list *constraint* is made with the function 5, then the integer numbers in the codification of the constraints are suppressed from the codification of Θ. The function *string2code* is just the translation of the brackets and union and intersection operators in negative numbers (-3 for '(', -4 for ')', -1 for '$\cup$' and -2 for '$\cap$') in order to manipulate faster integers than strings. This simple function is not provided here.

Function 4. - *addConstraint function*

```
function [ThetaR]=addConstraint(constraint,Theta)

% Code ThetaR the reduced form of Theta
% taking into account the constraints given by the user
%
% [ThetaR]=addConstraint(constraint,Theta)
%
% Inputs:
% constraint = the list of element considered as constraint
%             or '2T' to work on 2^Theta
% Theta = the description of Theta after coding
%
% Output:
% ThetaR = the description of coded Theta after reduction
% taking into account the constraints
%
% Copyright (c) 2008 Arnaud Martin

if strcmp(constraint{1}, '2T')
    n=size(Theta,2);
    nbCons=1;
    for i=1:n
        for j=i+1:n
            constraint(nbCons)={[i -2 j]};
            nbCons=nbCons+1;
        end
    end
else
    constraint=string2code(constraint);
end

constraintC=codingFocal(constraint,Theta);

sConstraint=size(constraintC,2);
unionCons=[];
for i=1:sConstraint
    unionCons=union(unionCons,constraintC{i});
end

sTheta=size(Theta,2);
```

```
for i=1:sTheta
    ThetaR{i}=setdiff(Theta{i},unionCons);
end
```

The function 5 simply transforms the list of focal elements given by the user with the codification of Θ to obtain the list of constraints and with Θ_r for the focal elements of each expert. The function 6 prepares the coding of focal elements and returns the list of the experts with the coded focal elements.

Function 5. - *codingFocal function*

```
function [focalC]=codingFocal(focal,Theta)

% Code the focal element for DSmT framework
%
% [focalC]=codingFocal(focal,Theta)
%
% Inputs:
% focal = the list of focal element for one expert
% Theta = the description of Theta after coding
%
% Output:
% focalC = the list of coded focal element for one expert
%
% Copyright (c) 2008 Arnaud Martin

 nbfoc=size(focal,2);
 if nbfoc
     for foc=1:nbfoc
         elemC=treat(focal{foc},Theta);
         focalC{foc}=elemC;
     end
 else
     focalC={[]};
 end
end

%%
function [elemE]=eval(oper,a,b)

    if oper==-2
        elemE=intersect(a,b);
    else
        elemE=union(a,b);
    end
```

```
end

%%
function [elemC,cmp]=treat(focal,Theta)

    nbelem=size(focal,2);
    PelemC=0;
    oper=0;
    e=1;
    if nbelem
        while e <= nbelem
           elem=focal(e);
           switch elem
               case -1
                   oper=-1;
               case -2
                   oper=-2;
               case -3
                  [elemC,nbe]=treat(focal(e+1:end),Theta);
                  e=e+nbe;

                   if oper~=0 & ~isequal(PelemC,0)
                       elemC=eval(oper,PelemC,elemC);
                       oper=0;
                   end
                   PelemC=elemC;

               case -4
                   cmp=e;
                   e=nbelem;
               otherwise
                   elemC=Theta{elem};
                   if oper~=0 & ~isequal(PelemC,0)
                       elemC=eval(oper,PelemC,elemC);
                       oper=0;
                   end
                   PelemC=elemC;
           end
           e=e+1;
        end
    else
        elemC=[];
    end
end
```

Function 6. - *codingExpert function*

```
function [expertC]=codingExpert(expert,Theta)

% Code the focal element for DSmT framework
%
% [expertC]=codingExpert(expert,Theta)
%
% Inputs:
% expert = structure containing the list of focal elements for
%         each expert and the bba corresponding
% Theta = the description of Theta after coding
%
% Output:
% expertC = structure containing the list of coded focal element
%         for each expert and the bba corresponding
%
% Copyright (c) 2008 Arnaud Martin

    nbExp=size(expert,2);
    for exp=1:nbExp
       focal=string2code(expert(exp).focal);
       expertC(exp).focal=codingFocal(focal,Theta);
       expertC(exp).bba=expert(exp).bba;
    end
end
```

7.5.2 Combination

The function 7 proposes many combination rules. Most of them are based on the function 8, but for some combination rules we need to keep more information, so we use the function 9 for the conjunctive combination. *E.g.* in the function 10 note the simplicity of the code for the PCR6 combination rule. The codes for other combination rules are not given here for the sake of clarity.

Function 7. - *combination function*

```
function [res]=combination(expertC,ThetaR,criterion)

% Give the combination of many experts
%
% [res]=combination(expert,constraint,n,criterion)
%
% Inputs:
```

```
% expertC = contains the structure of the list of focal elements
%           and corresponding bba for all the experts
% ThetaR = the coded and reduced discernment space
% criterion = is the combination criterion
%    criterion=1 Smets criterion (conjunctive rule in open world)
%    criterion=2 Dempster-Shafer criterion (normalized)
%                (conjunctive rule in closed world)
%    criterion=3 Yager criterion
%    criterion=4 disjunctive combination criterion
%    criterion=5 Florea criterion
%    criterion=6 PCR6
%    criterion=7 Mean of the bbas
%    criterion=8 Dubois criterion
%                (normalized and disjunctive combination)
%    criterion=9 Dubois and Prade criterion (mixt combination)
%    criterion=10 Mixt Combination (Martin and Osswald criterion)
%    criterion=11 DPCR (Martin and Osswald criterion)
%    criterion=12 MDPCR (Martin and Osswald criterion)
%    criterion=13 Zhang's rule
%
% Output:
% res = contains the structure of the list of focal elements and
%       corresponding bbas for the combinated experts
%
% Copyright (c) 2008 Arnaud Martin

switch criterion
    case 1
        %Smets criterion
        res=conjunctive(expertC);
    case 2
        %Dempster-Shafer criterion (normalized)
        expConj=conjunctive(expertC);
        ind=findeqcell(expConj.focal,[]);
        if ~isempty(ind)
            k=expConj.bba(ind);
            expConj.bba=expConj.bba/(1-k);
            expConj.bba(ind)=0;
        end
        res=expConj;
    case 3
        %Yager criterion
        expConj=conjunctive(expertC);
        ind=findeqcell(expConj.focal,[]);
        if ~isempty(ind)
            k=expConj.bba(ind);
            eTheta=ThetaR{1};
            for i=2:n
                eTheta=[union(eTheta,ThetaR{i})];
            end
            indTheta=findeqcell(expConj.focal,eTheta);
            if ~isempty(indTheta)
                expConj.bba(indTheta)=expConj.bba(indTheta)+k;
                expConj.bba(ind)=0;
            else
                sFocal=size(expConj.focal,2);
                expConj.focal(sFocal+1)={eTheta};
                expConj.bba(sFocal+1)=k;
                expConj.bba(ind)=0;
            end
        end
        res=expConj;
    case 4
        %disjounctive criterion
        [res]=disjunctive(expertC);
    case 5
        % Florea criterion
        expConj=conjunctive(expertC);
        expDis=disjunctive(expertC);

        ind=findeqcell(expConj.focal,[]);
        if ~isempty(ind)
            k=expConj.bba(ind);
            alpha=k/(1-k+k*k);
            beta=(1-k)/(1-k+k*k);

            expFlo=expConj;
            expFlo.bba=beta.*expFlo.bba;
            expFlo.bba(ind)=0;
            nbFocConj=size(expConj.focal,2);
            nbFocDis=size(expDis.focal,2);

            expFlo.focal(nbFocConj+1:nbFocConj+nbFocDis)=...
            ...expDis.focal;
            expFlo.bba(nbFocConj+1:nbFocConj+nbFocDis)=...
            ...alpha.*expDis.bba;

            expFlo=reduceExpert(expFlo);
        else
            expFlo=expConj;
        end
        res=expFlo;
    case 6
        % PCR6
        [res]=PCR6(expertC);
    case 7
        % Means of the bba
        [res]=meanbba(expertC);
    case 8
        % Dubois criterion (normalized & disjunctive combination)
        expDis=disjunctive(expertC);

        ind=findeqcell(expDis.focal,[]);
        if ~isempty(ind)
            k=expDis.bba(ind);
            expDis.bba=expDis.bba/(1-k);
            expDis.bba(ind)=0;
        end
        res=expDis;
    case 9
        % Dubois and Prade criterion (mixt combination)
        [res]=DP(expertC);
    case 10
        % Martin and Ossawald criterion (mixt combination)
        [res]=Mix(expertC);
    case 11
        % DPCR (Martin and Osswald criterion)
        [res]=DPCR(expertC);
    case 12
        % MDPCR (Martin and Osswald criterion)
        [res]=MDPCR(expertC);
    case 13
        % Zhang's rule
        [res]=Zhang(expert)

    otherwise
        'Accident: in combination choose of criterion: uncorrect'
end
```

Function 8. - *conjunctive function*

```
function [res]=conjunctive(expert)

% Conjunctive Rule
%
% [res]=conjunctive(expert)
%
% Inputs:
% expert = contains the structures of the list of focal elements
% and corresponding bba for all the experts
%
% Output:
% res = is the resulting expert (structure of the list of focal
%       element and corresponding bba)
%
% Copyright (c) 2008 Arnaud Martin

nbexpert=size(expert,2);
for i=1:nbexpert
    nbfocal(i)=size(expert(i).focal,2);
    nbbba(i)=size(expert(i).bba,2);
    if nbfocal(i)~=nbbba(i)
        'Accident: in conj: the numbers of bba and focal...
                ... element are different'
    end
end

interm=expert(1);
for exp=2:nbexpert
    nbfocalInterm=size(interm.focal,2);
    i=1;
    comb.focal={};
    comb.bba=[];
    for foc1=1:nbfocalInterm
        for foc2=1:nbfocal(exp)
            tmp=intersect(interm.focal{foc1},...
             ...expert(exp).focal{foc2});
            if isempty(tmp)
                tmp=[];
            end
            comb.focal(i)={tmp};
            comb.bba(i)=interm.bba(foc1)*expert(exp).bba(foc2);
            i=i+1;
        end
    end
    interm=reduceExpert(comb);
end
res=interm;
```

Function 9. - *globalConjunctive function*

```
function [res,tabInd]=globalConjunctive(expert)

% Conjunctive Rule conserving all the focal elements
% during the combination
%
% [res,tabInd]=globalConjunctive(expert)
%
% Input:
% expert = contains the structures of the list of focal elements
% and corresponding bba for all the experts
%
% outputs:
% res = is the resulting expert (structure of the list of focal
%       element and corresponding bba)
% tabInd = table of the indices given the combination
%
% Copyright (c) 2008 Arnaud Martin

nbexpert=size(expert,2);
for i=1:nbexpert
    nbfocal(i)=size(expert(i).focal,2);
    nbbba(i)=size(expert(i).bba,2);
    if nbfocal(i)~=nbbba(i)
        'Accident: in conj: the numbers of bba and focal...
         ... element are different'
    end
end
interm=expert(1);
tabIndPrev=[1:1:nbfocal(1)];
for exp=2:nbexpert
    nbfocalInterm=size(interm.focal,2);
    i=1;
    comb.focal={};
    comb.bba=[];
    tabInd=[];
    for foc1=1:nbfocalInterm
        for foc2=1:nbfocal(exp)
```

```
            tmp=intersect(interm.focal{foc1},...
            ...expert(exp).focal{foc2});
            tabInd=[tabInd [tabIndPrev(:,foc1);foc2]];
            if isempty(tmp)
                tmp=[];
            end
            comb.focal(i)={tmp};
            comb.bba(i)=interm.bba(foc1)*expert(exp).bba(foc2);
            i=i+1;
        end
    end
    tabIndPrev=tabInd;
    interm=comb;
end
res=interm;
```

Function 10. - *PCR6 function*

```
function [res]=PCR6(expert)

% PCR6 combination rule
%
% [res]=PCR6(expert)
%
% Input:
% expert = contains the structures of the list of focal elements
% and corresponding bba for all the experts
%
% Output:
% res = is the resulting expert (structure of the list of focal
%      element and corresponding bba)
%
% Reference: A. Martin and C. Osswald, ''A new generalization
%  of the proportional conflict redistribution rule stable in
%  terms of decision,'' Applications and Advances of DSmT for
%  Information Fusion, Book 2, American Research Press Rehoboth,
%   F. Smarandache and J. Dezert, pp. 69-88 2006.
%
% Copyright (c) 2008 Arnaud Martin

[expertConj,tabInd]=globalConjunctive(expert);

ind=findeqcell(expertConj.focal,[]);
```

```
nbexp=size(tabInd,1);
if ~isempty(ind)
    expertConj.bba(ind)=0;
    sInd=size(ind,2);
    for i=1:sInd
        P=1;
        S=0;
        for exp=1:nbexp
            bbaexp=expert(exp).bba(tabInd(exp,ind(i)));
            P=P*bbaexp;
            S=S+bbaexp;
        end
        for exp=1:nbexp
            expertConj.focal(end+1)=...
            ...expert(exp).focal(tabInd(exp,ind(i)));
            expertConj.bba(end+1)=...
            ...expert(exp).bba(tabInd(exp,ind(i)))*P/S;
        end
    end
end
res=reduceExpert(expertConj);
```

7.5.3 Decision

The function 11 gives the decision on the expert focal element list for the corresponding bba with one of the chosen criterion and on the elements given by the user for the decision. Note that the choices ‘A’ and ‘Cm’ for the variable *elemDec* could take a long time because it needs the generation of D_r^Θ. This function can call one of the decision functions 13, 14, 15, 16. If any decision is possible on the chosen elements given by *elemDec*, the function returns -1. In case of rejected element, the function returns 0.

Function 11. - *decision function*

```
function [decFocElem]=decision(expert,Theta,criterion,elemDec)

% Give the decision for one expert
%
% [decFocElem]=decision(expert,Theta,criterion)
%
% Inputs:
% expert = contains the structure of the list of focal elements
%          and corresponding bba for all the experts
% Theta = list of coded (and reduced with constraint) of the
%         elements of the discernement space
% criterion = is the combination criterion
%    criterion=0 maximum of the bba
%    criterion=1 maximum of the pignistic probability
%    criterion=2 maximum of the credibility
%    criterion=3 maximum of the credibility with reject
%    criterion=4 maximum of the plausibility
%    criterion=5 DSmP criterion
%    criterion=6 Appriou criterion
%    criterion=7 Credibility on DTheta criterion
%    criterion=8  pignistic on DTheta criterion
% elemDec = list of elements on which we can decide,
%    or A for all, S for singletons only, F for focal elements
%    only, SF for singleton plus focal elements, Cm for given
%    specificity, 2T for only 2^Theta (DST case)
%
% Output:
% decFocElem = the retained focal element, 0 in case of reject, -1
%              if the decision cannot be taken on elemDec
%
% Copyright (c) 2008 Arnaud Martin


type=1;
switch elemDec{1}
    case 'S'
        type=0;
        elemDecC=Theta;
        expertDec=expert;
    case 'F'
        elemDecC=expert.focal;
        expertDec=expert;
    case 'SF'
        expertDec=expert;
        n=size(Theta,2);
        for i=1:n
            expertDec.focal{end+1}=Theta{i};
            expertDec.bba(end+1)=0;
        end
        expertDec=reduceExpert(expertDec);
        elemDecC=expertDec.focal;
    case 'Cm'
        sElem=size(elemDec,2);
        switch sElem
```

```
            case 2
                minSpe=str2num(elemDec{2});
                maxSpe=minSpe;
            case 3
                minSpe=str2num(elemDec{2});
                maxSpe=str2num(elemDec{3});
            otherwise
                'Accident in decision: with the option Cm for ...
             ...elemDec give the specifity of decision element ...
             ...(eventually the minimum and the maximum of the ...
             ...desired specificity'
                pause
        end

        elemDecC=findFocal(Theta,minSpe,maxSpe);
        expertDec.focal=elemDecC;

        expertDec.bba=zeros(1,size(elemDecC,2));
        for foc=1:size(expert.focal,2)
            ind=findeqcell(elemDecC,expert.focal{foc});
            if ~isempty(ind)
                expertDec.bba(ind)=expert.bba(foc);
            else
                expertDec.bba(ind)=0;
            end
        end

    case '2T'
        type=0;
        natoms=size(Theta,2);
        expertDec.focal(1)={[]};
        indFoc=findeqcell(expert.focal,{[]});
        if isempty(indFoc)
            expertDec.bba(1)=0;
        else
            expertDec.bba(1)=expert.bba(indFoc);
        end
        step =2;
        for i=1:natoms
            expertDec.focal(step)=codingFocal({[i]},Theta);

            indFoc=findeqcell(expert.focal,...
             ...expertDec.focal{step});
            if isempty(indFoc)
```

```
                expertDec.bba(step)=0;
            else
                expertDec.bba(step)=expert.bba(indFoc);
            end

            step=step+1;
            indatom=step;
            for step2=2:indatom-2
                expertDec.focal(step)=...
                ...{[union(expertDec.focal{step2},...
                 ...expertDec.focal{indatom-1})]};

                indFoc=findeqcell(expert.focal,...
                ...expertDec.focal{step});
                if isempty(indFoc)
                    expertDec.bba(step)=0;
                else
                    expertDec.bba(step)=expert.bba(indFoc);
                end

                step=step+1;
            end
        end
        elemDecC=expertDec.focal;
    case 'A'
        elemDecC=generationDThetar(Theta);
        elemDecC=reduceFocal(elemDecC);
        expertDec.focal=elemDecC;
        expertDec.bba=zeros(1,size(elemDecC,2));
        for foc=1:size(expert.focal,2)
           expertDec.bba(findeqcell(elemDecC,...
           ...expert.focal{foc}))=expert.bba(foc);
        end
    otherwise
        type=0;
        elemDec=string2code(elemDec);
        elemDecC=codingFocal(elemDec,Theta);
        expertDec=expert;
        nbElemDec=size(elemDecC,2);
        for foc=1:nbElemDec
            if ~isElem(elemDecC{foc}, expertDec.focal)
                expertDec.focal{end+1}=elemDecC{foc};
                expertDec.bba(end+1)=0;
            end
        end
end

%-----------------------------------------------------------
nbFocal=size(expertDec.focal,2);
switch criterion

    case 0
        % maximum of the bba
        nbFocal=size(expertDec.focal,2);
        nbElem=0;
        for foc=1:nbFocal
            ind=findeqcell(elemDecC,expertDec.focal{foc});
            if ~isempty(ind)
                bba(ind)=expertDec.bba(foc);
            end
        end
        [bbaMax,indMax]=max(bba);
        if bbaMax~=0
            decFocElem.bba=bbaMax;
            decFocElem.focal={elemDecC{indMax}};
        else
            decFocElem=-1;
        end
    case 1
        % maximum of the pignistic probability
        [BetP]=pignistic(expertDec);
        decFocElem=MaxFoc(BetP,elemDecC,type);
    case 2
        % maximum of the credibility
        [Bel]=credibility(expertDec);
        decFocElem=MaxFoc(Bel,elemDecC,type);
    case 3
        % maximum of the credibility with reject
        [Bel]=credibility(expertDec);

        TabSing=[];
        focTheta=[];
        for i=1:size(Theta,2)
            focTheta=union(focTheta,Theta{i});
        end

        for foc=1:nbFocal
            if isElem(Bel.focal{foc}, elemDecC)
```

```
                TabSing=[TabSing [foc ;  Bel.Bel(foc)]];
            end

        end
        [BelMax,indMax]=max(TabSing(2,:));
        if BelMax~=0
            focMax=Bel.focal{TabSing(1,indMax)};

            focComplementary=setdiff(focTheta,focMax);
            if isempty(focComplementary)
                focComplementary=[];
            end

            ind=findeqcell(Bel.focal,focComplementary);
            if BelMax < Bel.Bel(ind)
            % if ind is empty this is always false
                decFocElem=0; % That means that we reject
            else
               if isempty(ind)
                   decFocElem=0; % That means that we reject
               else
                decFocElem.focal={Bel.focal{TabSing(1,indMax)}};
                decFocElem.Bel=BelMax;
               end
            end
        else
            decFocElem=-1; % That means that we reject
        end
    case 4
        % maximum of the plausibility
        [Pl]=plausibility(expertDec);
        decFocElem=MaxFoc(Pl,elemDecC,type);
    case 5
        % DSmP criterion
        epsilon=0.00001; % 0 can allows problem

        [DSmP]=DSmPep(expertDec,epsilon);
        decFocElem=MaxFoc(DSmP,elemDecC,type);
    case 6
        % Appriou criterion
        [Pl]=plausibility(expertDec);
        lambda=1;
        r=0.5;
        bm=BayesianMass(expertDec,lambda,r);
        Newbba=Pl.Pl.*bm.bba;
        % normalization
        Newbba=Newbba/sum(Newbba);
        funcDec.focal=Pl.focal;
        funcDec.bba=Newbba;
        decFocElem=MaxFoc(funcDec,elemDecC,type);
    case 7
        % Credibility on DTheta criterion
        [Bel]=credibility(expertDec);
        lambda=1;
        r=0.5;
        bm=BayesianMass(expertDec,lambda,r);
        Newbba=Bel.Bel.*bm.bba;
        % normalization
        Newbba=Newbba/sum(Newbba);
        funcDec.focal=Bel.focal;
        funcDec.bba=Newbba;
        decFocElem=MaxFoc(funcDec,elemDecC,type);
    case 8
        % pignistic on DTheta criterion
        [BetP]=pignistic(expertDec);
        lambda=1;
        r=0.5;
        bm=BayesianMass(expertDec,lambda,r);
        Newbba=BetP.BetP.*bm.bba;
        % normalization
        Newbba=Newbba/sum(Newbba);
        funcDec.focal=BetP.focal;
        funcDec.bba=Newbba;
        decFocElem=MaxFoc(funcDec,elemDecC,type);
    otherwise
        'Accident: in decision choose of criterion: uncorrect'
end
end


%%
function [bool]=isElem(focal, listFocal)

% The g oal of this function is to return a boolean on the test
% focal in listFocal
%
% [bool]=isElem(focal, listFocal)
%
```

```
% Inputs:
% focal = one focal element (matrix)
% listFocal = the list of elements in Theta (all different)
%
% Output:
% bool = boolean: true if focal is in listFocal, elsewhere false
%
% Copyright (c) 2008 Arnaud Martin

n=size(listFocal,2);
bool=false;
for i=1:n
    if isequal(listFocal{i},focal)
        bool=true;
        break;
    end
end
end

%%
function [decFocElem]=MaxFoc(funcDec,elemDecC,type)

fieldN=fieldnames(funcDec);

switch fieldN{2}
    case 'BetP'
        funcDec.bba=funcDec.BetP;
    case 'Bel'
        funcDec.bba=funcDec.Bel;
    case 'Pl'
        funcDec.bba=funcDec.Pl;
    case 'DSmP'
        funcDec.bba=funcDec.DSmP;

end

if type
    [funcMax,indMax]=max(funcDec.bba);
    FocMax={funcDec.focal{indMax}};
else
    nbFocal=size(funcDec.focal,2);
    TabSing=[];
    for foc=1:nbFocal
        if isElem(funcDec.focal{foc}, elemDecC)
```

```
            TabSing=[TabSing [foc ;  funcDec.bba(foc)]];
        end
    end
    [funcMax,indMax]=max(TabSing(2,:));
    FocMax={funcDec.focal{TabSing(1,indMax)}};
end

if funcMax~=0
    decFocElem.focal=FocMax;
    switch fieldN{2}
        case 'BetP'
            decFocElem.BetP=funcMax;
        case 'Bel'
            decFocElem.Bel=funcMax;
        case 'Pl'
            decFocElem.Pl=funcMax;
        case 'DSmP'
            decFocElem.DSmP=funcMax;
    end
else
    decFocElem=-1;
end

end
```

Function 12. - *findFocal function*

```
function [elemDecC]=findFocal(Theta,minSpe,maxSpe)

% Find the element of DTheta with the minium of specifity minSpe
% and the maximum maxSpe
%
% [elemDecC]=findFocal(Theta,minSpe,maxSpe)
%
% Input:
% Theta = list of coded (and eventually reduced with constraint)
%         of the elements of the discernment space
% minSpe = minimum of the wanted specificity
% minSpe = maximum of the wanted specificity
%
% Output:
% elemDec = list of elements on which we want to decide with the
%           minimum of specifity minSpe and the maximum maxSpe
```

```
%
% Copyright (c) 2008 Arnaud Martin

elemDecC{1}=[];

n=size(Theta,2);
ThetaSet=[];
for i=1:n
    ThetaSet=union(ThetaSet,Theta{i});
end
for s=minSpe:maxSpe
    tabs=nchoosek(ThetaSet,s);
    elemDecC(end+1:end+size(tabs,1))=num2cell(tabs,2)';
end

elemDecC=elemDecC(2:end);
```

Function 13. - *pignistic function*

```
function [BetP]=pignistic(expert)

% Generalized Pignistic Transformation
%
% [BetP]=pignistic(expert)
%
% Input:
% expert = contains the structures of the list of focal elements
%          and corresponding bba for all the experts
% expert.focal = list of focal elements
% expert.bba = matrix of bba
%
% Output:
% BetP = contains the structure of the list of focal elements and
%        the matrix of the plausibility corresponding
% BetP.focal = list of focal elements
% BetP.BetP = matrix of the pignistic transformation

% Comment : 1- the code of the focal elements must inculde
%                 the constraints
%           2- The pignistic is given only on the elements
%                  in the list of focal of expert (the
%                  bba can be 0)
%
```

```
% Copyright (c) 2008 Arnaud Martin

nbFocal=size(expert.focal,2);

BetP.focal=expert.focal;
BetP.BetP=zeros(1,nbFocal);

for focA=1:nbFocal
    for focB=1:nbFocal
       focI=intersect(expert.focal{focA},expert.focal{focB});
       if ~isempty(focI)
          BetP.BetP(focA)=BetP.BetP(focA)+size(focI,2)/...
          ...size(expert.focal{focB},2)*expert.bba(focB);
       else
           if isequal(expert.focal{focB},[])
           % for the empty set:
           % cardinality(empty set)/cardinality(empty set)=1,
           % so we add the bba
               BetP.BetP(focA)=BetP.BetP(focA)+expert.bba(focB);
           end
       end
    end
end
```

Function 14. - *credibility function*

```
function [Bel]=credibility(expert)

% Credibility function
%
% [Bel]=credibility(expert)
%
% Input:
% expert = contains the structures of the list of focal elements
%          and corresponding bba for all the experts
% expert.focal = list of focal elements
% expert.bba = matrix of bba
%
% Output:
% Bel = contains the structure of the list of focal elements and
%       the matrix of the credibility corresponding
% Bel.focal = list of focal elements
% Bel.Bel = matrix of the credibility
% Comment : 1- the code of the focal elements must inculde
%              the constraints
%           2- The credibility is given only on the elements
%              in the list of focal of expert (the
%              bba can be 0)
%
% Copyright (c) 2008 Arnaud Martin

nbFocal=size(expert.focal,2);

Bel.focal=expert.focal;
Bel.Bel=zeros(1,nbFocal);

for focA=1:nbFocal
    for focB=1:nbFocal
        indMem=ismember(expert.focal{focB},expert.focal{focA});
       if sum(indMem)==size(expert.focal{focB},2)...
       && ~isequal(expert.focal{focB},[])
        Bel.Bel(focA)=Bel.Bel(focA)+expert.bba(focB);
     else
         if isequal(expert.focal{focB},[])...
           && isequal(expert.focal{focA},[])
       % the empty set is included to all the focal elements
             Bel.Bel(focA)=Bel.Bel(focA)+expert.bba(focB);
         end
     end
    end
end
```

Function 15. *- plausibility function*

```
function [Pl]=plausibility(expert)

% Plausibility function
%
% [Pl]=plausibility(expert)
%
% Input:
% expert = contains the structures of the list of focal elements
%          and corresponding bba for all the experts
% expert.focal = list of focal elements
% expert.bba = matrix of bba
```

```
%
% Output:
% Pl = contains the structure of the list of focal elements and
%      the matrix of the plausibility corresponding
% Pl.focal = list of focal elements
% Pl.Pl = matrix of the plausibility

% Comment : 1- the code of the focal elements must include
%              the constraints
%            2- The plausibility is given only on the elements
%              in the list of focal of expert (the
%              bba can be 0)
%
% Copyright (c) 2008 Arnaud Martin


nbFocal=size(expert.focal,2);
Pl.focal=expert.focal;
Pl.Pl=zeros(1,nbFocal);

for focA=1:nbFocal
    for focB=1:nbFocal
       focI=intersect(expert.focal{focA},expert.focal{focB});
       if ~isempty(focI)
          Pl.Pl(focA)=Pl.Pl(focA)+expert.bba(focB);
       else
           if isequal(expert.focal{focB},[])...
            && isequal(expert.focal{focA},[])
           % for the empty set we keep the bba for the Pl
               Pl.Pl(focA)=Pl.Pl(focA)+expert.bba(focB);
           end
       end
    end
end
```

Function 16. - *DSmPep function*

```
function [DSmP]=DSmPep(expert,epsilon)

% DSmP Transformation
%
% [DSmP]=DSmPep(expert,epsilon)
%
```

```
% Inputs:
% expert = contains the structures of the list of focal elements
%          and corresponding bba for all the experts
% expert.focal = list of focal elements
% expert.bba = matrix of bba
% epsilon = epsilon coefficient
%
% Output:
% DSmPep = contains the structure of the list of focal elements
%          and the matrix of the plausibility corresponding
% DSmPep.focal = list of focal elements
% DSmPep.DSmP = matrix of the pignistic transformation
%
% Reference: Dezert & Smarandache, ''A new probbilistic
%            transformation of belief mass assignment'',
%   fusion 2008, Cologne, Germany.
%
% Copyright (c) 2008 Arnaud Martin


nbFocal=size(expert.focal,2);

DSmP.focal=expert.focal;
DSmP.DSmP=zeros(1,nbFocal);

for focA=1:nbFocal
    for focB=1:nbFocal
       focI=intersect(expert.focal{focA},expert.focal{focB});
       sumbbaFocB=0;
       sFocB=size(expert.focal{focB},2);
       for elB=1:sFocB
           ind=findeqcell(expert.focal,expert.focal{focB}(elB));
           if ~isempty(ind)
               sumbbaFocB=sumbbaFocB+expert.bba(ind);
           end
       end
       if ~isempty(focI)
           sumbbaFocI=0;
           sFocI=size(focI,2);
           for elB=1:sFocI
               ind=findeqcell(expert.focal,focI(elB));
               if ~isempty(ind)
                   sumbbaFocI=sumbbaFocI+expert.bba(ind);
               end
```

```
            end
            DSmP.DSmP(focA)=DSmP.DSmP(focA)+expert.bba(focB)...
            ...*(sumbbaFocI+epsilon*sFocI)/...
            ...(sumbbaFocB+epsilon*sFocB);
        end
    end
end
```

7.5.4 Decoding and generation of D_r^Θ

For the displays, we must decode the focal elements and/or the final decision. The function 17 decodes the focal elements in the structure expert that contains normally only one expert. This function calls the function 18 that really does the decoding for the user. This function is based on the generation of D_r^Θ given by the function 21 that is a modified and adapted code from [11]. To generate D_r^Θ we first must create the intersection basis. Hence in the function 18 we use a loop of 2^Θ in order to generate the basis and at the same time to scan the power set 2^Θ and also the elements of the intersection basis. These two basis (intersection and union) are in fact concatenated during the construction, so we scan also some elements such as intersections of previous unions and unions of previous intersections. This generated set of elements does not cover D_r^Θ. When all the searching focal elements (that can be only one decision element) are found, we stop the function and avoid to generate all D_r^Θ. Hence if the searching elements are not all found after this loop, we begin to generate D_r^Θ and stop when all elements are found. So, with luck, that can be fast.

We can avoid to generate D_r^Θ for only the display if we use Smarandache's codification. The function 19 transforms the used code of the focal elements in the structure expert in Smarandache's code, easier to understand by reading. This function calls the function 20 that really does the transformation. The focal elements are directly in string for the display.

Function 17. - *decodingExpert function*

```
function [expertDecod]=decodingExpert(expert,Theta,DTheta)

% The goal of this function is to decode the focal elements in
% expert
%
% [expertDecod]=decodingExpert(expert,Theta)
%
% Inputs:
% expert = contains the structure of the list of focal elements
% after combination and corresponding bba for all the experts
% (generally use for only one after combination)
% Theta = list of coded (and reduced with constraint) of the
%         elements of the discernement space
% DTheta = list of coded (and reduced with constraint) of the
%          elements of DTheta
%
% Output:
% expertDecod = contains the structure of the list of decoded
%       (for human) focal elements and corresponding bba for
%        all the experts
%
% Copyright (c) 2008 Arnaud Martin

    nbExp=size(expert,2);
    for exp=1:nbExp
        focal=expert(exp).focal;
        expertDecod(exp).focal=decodingFocal(focal,{'A'},Theta,...
         ...DTheta);
        expertDecod(exp).bba=expert(exp).bba;
    end

end
```

Function 18. - *decodingFocal function*

```
function [focalDecod]=decodingFocal(focal,elemDec,Theta,DTheta)

% The goal of this function is to decode the focal elements
%
% [focalDecod]=decodingFocal(focal,elemDec,Theta)
%
% Inputs:
% expert = contains the structure of the list of focal elements
% after combination and corresponding bba for all the experts
% elemDec = the description of the subset of uncoded elements
%           for decision
% Theta = list of coded (and reduced with constraint) of the
%         elements of the discernement space
% DTheta = list of coded (and reduced with constraint) of the
%           elements of DTheta, eventually empty if not necessary
% Output:
% focalDecod = contains the list of decoded (for human) focal
%              elements
%
% Copyright (c) 2008 Arnaud Martin
```

```
switch elemDec{1}
    case {'F','A','SF','Cm'}
        opt=1;
    case 'S'
        opt=0;
        elemDecC=Theta;
        for i=1:size(Theta,2)
            elemDec(i)={[i]};
        end
    case '2T'
        opt=0;
        natoms=size(Theta,2);
        elemDecC(1)={[]};
        elemDec(1)={[]};
        step =2;
        for i=1:natoms
            elemDecC(step)=codingFocal({[i]},Theta);
            elemDec(step)={[i]};
            step=step+1;
            indatom=step;
            for step2=2:indatom-2
                elemDec(step)={[elemDec{step2} -1 ...
                       ...elemDec{indatom-1}]};
                elemDecC(step)={[union(elemDecC{step2},...
                 ...elemDecC{indatom-1})]};
                step=step+1;
            end
        end

    otherwise
        opt=0;
        elemDecN=string2code(elemDec);
        elemDecC=codingFocal(elemDecN,Theta);
end

if ~opt
    sFoc=size(focal,2);
    for foc=1:sFoc
        [ind]=findeqcell(elemDecC,focal{foc});
        if isempty(ind)
            'Accident in decodingFocal: elemDec does not be 2T'
            pause
        else
```

```
            focalDecod(foc)=elemDec(ind);
        end
    end
else
    focalDecod=cell(size(focal));
    cmp=0;
    sFocal=size(focal,2);
    sDTheta=size(DTheta.c,2);
    i=1;
    while i<sDTheta && cmp<sFocal
       DThetai=DTheta.c{i};
       indeq=findeqcell(focal,DThetai);
       if ~isempty(indeq)
           cmp=cmp+1;
           focalDecod(indeq)=DTheta.s(i);
       end
       i=i+1;
    end
end
```

Function 19. - *cod2ScodExpert function*

```
function [expertDecod]=cod2ScodExpert(expert,Scod)

% The goal of this function is to code the focal elements in
% expert with the Smarandache's codification from the practical
% codification in order to display the expert
%
% [expertDecod]=cod2ScodExpert(expert,Scod)
%
% Inputs:
% expert = contains the structure of the list of focal elements
% after combination and corresponding bba for all the experts
% (generally use for only one after combination)
% Scod = list of distinct part of the Venn diagram coded with the
%        Smarandache's codification
% Output:
% expertDecod = contains the structure of the list of decoded
%               (for human) focal elements and corresponding bba
% for all the experts
%
% Copyright (c) 2008 Arnaud Martin
```

```
    nbExp=size(expert,2);
    for exp=1:nbExp
        focal=expert(exp).focal;
        expertDecod(exp).focal=cod2ScodFocal(focal,Scod);
        expertDecod(exp).bba=expert(exp).bba;
    end

end
```

Function 20. - *cod2ScodFocal function*

```
function [focalDecod]=cod2ScodFocal(focal,Scod)

% The goal of this function is to code the focal elements with
% the Smarandache's codification from the practical codification
% in order to display the focal elements
%
% [focalDecod]=cod2ScodFocal(focal,Scod)
%
% Inputs:
% expert = contains the structure of the list of focal elements
%     after combination and corresponding bba for all the experts
% Scod = list of distinct part of the Venn diagram coded with the
%        Smarandache's codification
% Output:
% focalDecod = contains the list of decoded (for human) focal
%              elements
%
% Copyright (c) 2008 Arnaud Martin

sFocal=size(focal,2);
for foc=1:sFocal
    sElem=size(focal{foc},2);
    if sElem==0
        focalDecod{foc}='{}';
    else
        ch='{';
        ch=strcat(ch,'<');
        ch=strcat(ch,num2str(Scod{focal{foc}(1)}));
        ch=strcat(ch,'>');
        for elem=2:sElem
            ch=strcat(ch,',<');
            ch=strcat(ch,num2str(Scod{focal{foc}(elem)}));
            ch=strcat(ch,'>');
        end
        focalDecod{foc}=strcat(ch,'}');
    end
end
```

Function 21. - *generationDThetar function*

```
function [DTheta]=generationDThetar(Theta)

% Generation of DThetar: modified and adapted code from
% Dezert & Smarandache Chapter 2 DSmT book Vol 1
% to generate DTeta
%
% [DTheta]=generationDThetar(Theta)
%
% Input:
% Theta = list of coded (and eventually reduced with constraint)
%         of the elements of the discernment space
%
% Output:
% DTheta = list of coded (and eventually reduced with constraint
%          in this case some elements can be the same) of the
%          elements of the DTheta
%
% Copyright (c) 2008 Arnaud Martin

n=size(Theta,2);
step =1;
for i=1:n
    basetmp(step)={[Theta{i}]};
    step=step+1;
    indatom=step;
    for step2=1:indatom-2
        basetmp(step)={intersect(basetmp{indatom-1},...
         ...basetmp{step2})};
        step=step+1;
    end
end
sBaseTmp=size(basetmp,2);
step=1;
for i=1:sBaseTmp
    if ~isempty(basetmp{i})
```

```
        base(step)=basetmp(i);
        step=step+1;
    end
end
sBase=size(base,2);
DTheta{1}=[];
step=1;
nbC=2;
stop=0;
D_n1 =[0 ; 1];
sDn1=2;
for nn=1:n
    D_n =[ ] ;

    cfirst=1+(nn==n);
    for i =1:sDn1
        Li=D_n1(i,:);
        sLi=size(Li,2);
        if (2*sLi>sBase)&& (Li(sLi-(sBase-sLi))==1)
            stop=1;
            break
        end

        for j=i:sDn1
            Lj=D_n1(j,:);
            if(and(Li,Lj)==Li)&(or(Li,Lj)==Lj)
                D_n=[D_n ; Li Lj ] ;

                if size(D_n,1)>step
                    step=step+1;
                    DTheta{step}=[];
                    for c=cfirst:nbC
                        if D_n(end,c)
                            if isempty(DTheta{step})
                                DTheta{step}=base{sBase+c-nbC};
                            else
                                DTheta{step}=union(DTheta{step},...
                                ...base{sBase+c-nbC});
                            end
                        end
                    end
                end
            end
        end
    end
    if stop
        break
    end
    D_n1=D_n;
    sDn1=size(D_n1,1);
    nbC=2*size(D_n1,2);
end
```

7.6 Acknowledgments

The author thanks Pascal Djiknavorian for the interesting discussion on the generation of D_r^Θ, Florentin Smarandache for his comments on the codification and Jean Dezert for his advices on the representation of the DSm cardinality $\mathcal{C}_\mathcal{M}(X)$.

7.7 References

[1] A. Appriou, *Uncertain Data Aggregation in Classification an Tracking Process*, in Aggregation and Fusion of Imperfect Information, Springer, ed. B. Bouchon-Meunier, pp. 231-260, 1998.

[2] A. Appriou, *Approche générique de la gestion de l'incertain dans les processus de fusion multisenseur*, Traitement du signal, 24(4), pp. 307-319, 2005.

[3] J.A. Barnett, *Computational methods for a mathematical theory of evidence*, International Joint Conference on Artificial Intelligence (IJCAI), pp. 868-875, Vancouver, Canada, 1981.

[4] M. Bauer, *Approximation algorithms and decision making in the Dempster-Shafer theory of evidence*, International Journal of Approximate Reasoning, 17, pp. 217-237, 1997.

[5] M. Daniel, *Classical combination rules generalized to DSm hyper-power sets and their comparison with the hybrid DSm rule*, Chap. 3, pp. 89-112, in [**?**].

[6] A.P. Dempster, *Upper and Lower probabilities induced by a multivalued mapping*, Anals of Mathematical Statistics, 38, pp. 325-339, 1967.

[7] T. Denœux, *A k-Nearest Neighbor Classification Rule Based on Dempster-Shafer Theory*, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 25(5), pp. 804-813, May 1995.

[8] T. Denœux, *Analysis of evidence-theoric decision rules for pattern classification*, Pattern Recognition, 30(7), pp. 1095-1107, 1997.

[9] T. Denœux, *Inner and outer approximation of belief structures using a hierarchical clustering approach*, Int. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 9(4), pp. 437-460, 2001.

[10] J. Dezert, *Foundations for a new theory of plausible and paradoxical reasoning*, Information & Security: An International Journal, 9, 2002.

[11] J. Dezert and F. Smarandache, *The generation of the hyper-power sets*, Chap. 2, pp. 37-48, in [29].

[12] J. Dezert and F. Smarandache, *Partial ordering on hyper-power sets*, Chap. 3, pp. 49-60, in [29].

[13] J. Dezert and F. Smarandache, *A new probabilistic transformation of belief mass assignment*, In Eleventh International Conference on Information Fusion, Cologne, Germany, July 2008.

[14] J. Dezert and F. Smarandache and M. Daniel, *The Generalized Pignistic Transformation*, In Seventh International Conference on Information Fusion, Stockholm, Sweden, June 2004.

[15] P. Djiknavorian and D. Grenier, *Reducing DSmT hybrid rule complexity thought optimisation of the calculation algorithm* Chap. 15, pp. 345-440, in [30].

[16] R. Haenni and N. Lehmann, *Resource-Bounded and anytime approximation of belief function computations*, International Journal of Approximate Reasoning, 32(1-2), pp. 103-154, 2002.

[17] R. Haenni and N. Lehmann, *Implementing belief function computations*, International Journal of Intelligent Systems, Special issue on the Dempster-Shafer theory of evidence, 18(1), pp. 31-49, 2003.

[18] R. Kennes, *Computational Aspect of the Möbius Transformation of Graphs*, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 22(2), pp. 201-223, 1992.

[19] A. Martin and A.-L. Jousselme and C. Osswald, *Conflict measure for the discounting operation on belief functions*, In International Conference on Information Fusion, Cologne, Germany, July 2008.

[20] A. Martin and C. Osswald,*Generalized proportional conflict redistribution rule applied to Sonar imagery and Radar targets classification*, Chap. 11, pp. 289-304, in [30].

[21] A. Martin and C. Osswald,*Human experts fusion for image classification*, Information & Security: An International Journal, Special issue on Fusing Uncertain, Imprecise and Paradoxist Information (DSmT), 2006.

[22] A. Martin and C. Osswald,*A new generalization of the proportional conflict redistribution rule stable in terms of decision*, Chap. 2, pp. 69-88, in [30].

[23] A. Martin and C. Osswald, *Toward a combination rule to deal with partial conflict and specificity in belief functions theory*, In International Conference on Information Fusion, Québec, Canada, July 2007.

[24] A. Martin and I. Quidu, *Decision support with belief functions theory for seabed characterization.*, In International Conference on Information Fusion, Cologne, Germany, July 2008.

[25] P. Orponen, *Dempster's rule of combination is #P-complete*, Artificial Intelligence, 44, pp. 245-253, 1990.

[26] G. Shafer, *A mathematical theory of evidence*, Princeton University Press, 1976.

[27] G. Shafer and R. Logan, *Implementing Dempster's Rule for Hierarchical Evidence*, Artificial Intelligence, 33, pp. 271-298, 1987.

[28] P.P. Shenoy and G. Shafer, *Propagating belief functions with local computations*, IEEE Expert, 1(3), pp. 43-51, 1986.

[29] F. Smarandache and J. Dezert (Editors),*Advances and Applications of DSmT for Information Fusion, Collected works, Vol. 1*, Rehoboth, U.S.A., 2004.

[30] F. Smarandache and J. Dezert (Editors),*Advances and Applications of DSmT for Information Fusion, Collected works, Vol. 2*, Rehoboth, U.S.A., 2006.

[31] F. Smarandache and J. Dezert, *Proportional conflict redistribution rules for information fusion*, Chap. 1, pp. 3-68, in [30].

[32] Ph. Smets, *Practical uses of belief functions*, In K.B. Laskey and H. Prade, editors, Fifteenth Conference on Uncertainty in Artifficial Intelligence, volume 99, pages 612-621, Stockholm, Sweden, July 1999.

[33] Ph. Smets, *The application of matrix calculus for belief functions*, International Journal of Approximate Reasoning, 31:130, 2002.

[34] Ph. Smets, *Decision making in the TBM: the necessity of the pignistic transformation*, International Journal of Approximate Reasonning, 38:133-147, 2005.

[35] Ph. Smets, *Analyzing the combination of conflicting belief functions*, Information Fusion, 8, pp. 387-412, 2006.

[36] B. Tessem, *Approximations for efficient computation in the theory of evidence*, Artificial Intelligence, 61, pp. 315-329, 1993.

[37] F. Voorbraak, *A computationally efficient approximation of Dempster-Shafer theory*, International Journal Man-Machine Studies, 30, pp. 525-536, 1989.

[38] N. Wilson, *Algorithms for Dempster-Shafer theory*, In D.M. Gabbay and Ph. Smets, editors, Hanbook of defeasible reasoning and uncertainty management, volume 5: Algorithms for uncertainty and Defeasible Reasoning, pp. 421-475. Kluwer Academic Publisher, Boston, 2000.