Dempster-Shafer Theory: How Constraint Programming Can Help

Alexandros Kaltsounidis(✉) and Isambo Karali(✉)

Department of Informatics and Telecommunications,
National and Kapodistrian University of Athens, Athens, Greece
`{A.Kaltsounidis,izambo}@di.uoa.gr`

Abstract. Dealing with uncertainty has always been a challenging topic in the area of knowledge representation. Nowadays, as the internet provides a vast platform for knowledge exploitation, the need becomes even more imminent. The kind of uncertainty encountered in most of these cases as well as its distributed nature make Dempster-Shafer (D-S) Theory to be an appropriate framework for its representation. However, we have to face the drawback of the computation burden of Dempster's rule of combination due to its combinatorial behavior. Constraint Programming (CP) has proved to be an efficient tool in cases where results have to satisfy some specified properties and pruning of the computation space can be achieved. As D-S theory measures' computation fulfills this requirement, CP seems a promising framework to employ for this purpose. In this paper, we present our approach to use CP to compute the *belief* and *plausibility* measures of D-S Theory and Dempster's rule of combination as well as the results of the effort. As it was expected, the results are quite promising and in many cases impressive.

Keywords: Dempster-Shafer theory · Uncertainty · Constraint Programming · ECLiPSe prolog

1 Introduction

In the area of Knowledge Representation there are many frameworks whose purpose is to model the raw information available so as to create a meaningful and useful inference. Unfortunately, knowledge is not always expressed if terms of indisputable facts. It can be either uncertain or vague or both. Uncertainty may be the product of incomplete or unreliable knowledge and it raises a significant confusion as to how it should be treated. A lot of research has been done in the area and different approaches have been suggested. Efforts combine results from computer science, statistics, game theory and philosophy [1]. These include, but are not limited to, probability, possibility theory [2], probabilistic reasoning such as Bayesian Networks [3], Non-monotonic reasoning [4] and Dempster-Shafer (D-S) theory [1, 5–7]. In case of imprecise or vague information, its modeling has been somehow interrelated with Fuzzy Sets and Fuzzy Logic [8, 9].

The Internet, the World Wide Web (WWW) and the ongoing evolution of the Semantic Web (SW) [10], provide an enormous information store for knowledge extraction and exploitation. The need for an efficient framework able to reason under uncertainty is more urgent than ever. Apart from the above mentioned problems, frequently missing information, the size of the available data and its distributed nature provide some special characteristics. As an example, consider the case of the online traveling sites which provide information about hotels and a user's need to be provided with some accommodation suggestion according to some personal criteria. Hotel information is distributed among the sites, information is stored heterogeneously and some of it may be missing from some sites. Regarding dealing with the uncertainty in the Web, a detailed discussion can be found in [11].

Dempster-Shafer (D-S) theory, also referred to as the *theory of belief functions* or *evidence theory* is able to deal with ignorance and missing information (i.e. epistemic uncertainty). It is a generalization of the Bayesian theory of subjective probabilities and as such it allows for a greater degree of flexibility. For compositionality purposes, it also offers a rule for combining evidence from different independent sources [12]. The Theory has already been adopted in WWW or SW environments even as a tool for inexact knowledge modelling [13–17]. Some criticism has been done regarding when to use Dempster's Rule of Combination, as it might produce counter intuitive results in some cases [18, 19]. However, the Theory is widely used, especially in practical situations relative to data fusion [20]. According to [21], whether Dempster's rule of combination is appropriate depends on the problem's characteristics. Although D-S theory is a valuable framework for handling uncertainty with the current traits of knowledge representation, the computation of Dempster's rule puts a significant barrier to the theory being used more in practice because of its high complexity. It has been proved that computing Dempster's rule of combination is a #P-complete problem [22] and so it cannot be performed in an acceptable time when the data grow significantly, although methods of computing Dempster's rule in a more efficient way have been proposed [23]. To overcome this obstacle of complexity, many researchers have resorted to approximation algorithms [23–27]. In general, their efforts fall into two categories. The first one contains algorithms that make use of Monte Carlo, or similar random methodologies, to compute a solution [23]. The second category [24–27] consists of algorithms which alter the input data, in order to create an easier to compute problem. This is carried out by disregarding facts that have little evidence.

In order to avoid losing accuracy and be able to use Dempster's Rule of Combination as is, our approach to the complexity of the rule is to use Constraint Programming for its computation. Constraint programming (CP) [28] is a programming paradigm where relationships between the variables of the problem's computational space are stated in the form of constraints. In the case of Dempster's Rule of Combination, we utilize constraints to compute only the combinations that have a non-empty intersection (we can extract the normalization factor out of these). CP is also used to avoid evaluating redundant combinations, when computing *belief* and *plausibility* of a specific set by setting constrains on subset and non-empty intersection relationships, respectively. In order to evaluate the method, we employed ECLiPSe Prolog [29] using its set constraints solver [30, 31]. We created a program that performs Dempster's rule of combination on any number of mass functions using constraints and we compared it with a *generate-and-test* implementation. The latter evaluates every possible focal point combination. For comparison reasons, both programs ran on a number of random test cases that we created by using a variable number of: i) mass functions, ii) focal points per mass function, and iii) elements of the Universe. The *constrain-and-generate* program outperformed the *generate-and-test* one in the tests. As expected, the time the constraint program needed was relative to the number of combinations which have a non-empty intersection, whereas the generate-and-test program's time was related to the total number of combinations. When computing the *belief* (or *plausibility*) for a set A, the time needed by the constraint program was relative to the number of combinations whose intersection is a subset of (or intersects with) A. Thus, for the *constraint-and-generate* method no redundant sets are generated.

It is worth mentioning that Constraint Systems (CS), the formalism that models constraint problems, and D-S theory have already been related since many years ago. CS have been employed to model the uncertainty expressed by belief functions in a variety of early works, for instance [32]. In addition, recently, Constraint Satisfaction Problems (CSPs) have been extended with uncertainty. In [33], a unifying CSP extension based on the D-S theory is presented that supports uncertainty, soft and prioritized constraints and preferences over the solutions.

In this paper, we present our approach in using CP to reduce the computation time for D-S theory measures, namely *belief* and *plausibility*, as well as Dempster's rule of combination and summarize our results. The paper is structured as follows. The necessary background on D-S theory will be recalled in Sect. 2. In Sect. 3 the complexity of Dempster's rule as well as related work will be discussed. A brief description of Constraint Programming and Logic Programming will be given in Sect. 4, as constraint programming within logic programming is used in our prototype implementation. Our approach, its implementation and test results will be presented in Sect. 5. Section 6 will then conclude this paper.

2 Dempster Shafer Theory

Dempster-Shafer (D-S) Theory is a framework designed for reasoning under uncertainty, which allows to combine evidence from different independent sources. The latter is achieved by Dempster's rule of combination. This rule produces common shared belief between multiple sources and distributes non-shared belief through a normalization factor. Dempster's rule of combination is a powerful and useful tool and one of the reasons why the D-S theory has been so widely spread in many areas in computer science, from artificial intelligence to databases as it allows the Theory to deal with distributed sources.

D-S theory considers a universe, U, of mutually exclusive events. A basic *mass function*, m, assigns subjective probabilities to subsets of U. Then, on top of m, *belief* and *plausibility* functions are defined for any subset of U assigning lower and upper likelihoods to each of them. Composition of different *mass functions* on the same universe is achieved using *Dempster's Rule of Combination* considering independent sources. More formally, these measures are defined as follows.

2.1 Formal Definition

Let U be the Universe, i.e. the set containing all possible states of a system under consideration. The power set 2^U of U, is the set of all subsets of U, including the empty set Ø.

Mass Function: A mass function (also called *basic probability assignment* or *basic belief assignment*) is the most fundamental function for assigning degrees of belief to a set of one or more hypotheses. Formally, a mass function m is a function from 2^U to [0, 1]: $m : 2^U \longrightarrow [0,1]$ with the following properties:

1. The mass of the empty set is equal to zero $m(\varnothing) = 0$, and
2. The masses of the elements of the power set must sum to a total of one. $\sum\limits_{A \in 2^U} m(A) = 1$

Belief Function: The *belief* (also known as *support*) of a set $A \in 2^U$, denoted by $bel(A)$ expresses the total amount of belief committed to A and is defined as:

$$bel(A) = \sum_{B \subseteq A} m(B), \qquad \forall A \subseteq U.$$

Plausibility Function: The *plausibility* for a set $A \in 2^U$, denoted by $pl(A)$ expresses the amount of belief not committed to the complement of A and thus declares how plausible A is. It is computed as:

$$pl(A) = \sum_{B \cap A \neq \varnothing} m(B), \qquad \forall A \subseteq U.$$

Dempster's Rule of Combination: The theory of evidence also handles the problem of how to combine evidence from different independent sources.

Let $m_1, m_2, \ldots, m_n$ be mass functions defined over the same Universe U. Then using Dempster's rule of combination we can compute a new mass function that incorporates the evidence of $m_1, m_2, \ldots, m_n$. We use $\oplus$ to denote the operator of Dempster's rule of combination. Then the combination of $m_1, m_2, \ldots, m_n$ is called the *joint mass* $m_{1,2,..,n} \equiv m_1 \oplus m_2 \oplus \ldots \oplus m_n$ and is defined as:

$$m_1 \oplus m_2 \oplus \ldots \oplus m_n = \bigoplus_{i=1}^{n} m_i(A) = \tfrac{1}{1-K} \sum_{B_1,..,B_n \mid \cap_{i=1}^{n} B_i = A} m_1(B_1) \cdot \ldots \cdot m_n(B_n), \ \forall A \subseteq U, A \neq \varnothing$$

where,

$$K = \sum_{B_1,..,B_n \mid \cap_{i=1}^{n} B_i = \varnothing} m_1(B_1) \cdot \ldots \cdot m_n(B_n).$$

Here, K is a normalization constant that accounts for the products of mass values corresponding to the empty intersections of focal points. It can be considered as a measure of conflict between the mass functions.

3 Complexity of Dempster's Rule of Combination – Approximation Algorithms

Dempster's Rule of Combination has exponential complexity. In [22], it was shown that evaluating the rule is a #P-complete problem.

To handle the complexity problem, many have resorted to approximation algorithms to compute the rule. An approximation algorithm is an efficient algorithm that finds approximate solutions for the desired problem with provable guarantees on the distance of the evaluated solution to the exact one. Towards this, a lot of work has been done into trying to reduce the size of input (i.e. the number of focal points of each mass function), with some of the most well-known methods being the Bayesian approximation [24], the *k-l-x* method [25], the summarization method [26], and the D1 Approximation [27]. There has also been effort into developing algorithms to directly compute an approximate value for Dempster's rule's result using Monte Carlo models and/or Markov Chain models [23].

When a lot of information is available, i.e. the size of the problem is big, we can resort to approximation algorithms, as a loss of accuracy can be tolerated, for the sake of a significant reduction in time needed for the computation. On the other hand, when we have only a few sources of evidence to combine, and/or a small number of focal points per mass function, an approximation can be deceptive, thus we have to compute the exact solution.

To give a better understanding of the problem, we can summarize it as follows. Given a frame of discernment U of size |U|, a mass function m can have up to $2^{|U|}$ ($2^{|U|} - 1$ to be precise, as the empty set cannot be a focal point) focal points. Given a mass function m, the computation of the *belief*, or the *plausibility* function for m is linear to the number of focal points of m (as computation of sums). Note, however, that the combination of two mass functions through Dempster's rule of combination requires the computation of up to $2^{2|U|}$ intersections. To generalize, let n be the number of mass functions $m_1, \ldots, m_n$, the computation of the *joint mass* $m_{1,\ldots,n} = \bigoplus_{i=11}^{n} m_i$ needs up to $2^{n|U|}$ intersections to be computed. Thus the worst case complexity of Dempster's rule is $O(2^{n|U|})$. To be precise, say we know the number of focal elements of each mass function, i.e. let $q_1, \ldots, q_n$- be the number of focal points of each mass function $m_1, \ldots, m_n$ respectively, then the complexity of the rule of combination is $\Theta(Q)$, where Q is the product $Q = q_1 \cdot q_2 \cdot \ldots \cdot q_n$.

When Constraint Programming is involved while computing Dempster's rule of combination, we consider only the number of combinations that result in a desired non-empty set. Let A be a set and Q_A the number of combinations whose intersection is a subset of A, then a method utilizing Constraint Programming would need time proportional to Q_A, where obviously $Q_A < Q$, where Q as above.

In the following section we shall give a short description of constraint programming as well as logic programming to allow a better understanding of the approach that we follow.

4 Constraint Programming and Logic Programming

Constraint Programming (CP) [28] is a powerful paradigm that can be used to solve a variety of problems referred to as Constraint Satisfaction Problems (CSP). In CP a declarative description of the problem is given by the programmer and a constraint solver is used to find an acceptable solution to it. More precisely, a CSP can be described as a set of Variables, each associated with a domain of values, and a set of Constraints over subsets of these Variables. A solution to the problem is an assignment of values to the Variables so that all Constraints are satisfied. Constraint Programming uses constraint propagation to reduce the search space allowing for solving the problem faster. However, the time that the constraint propagation process needs must be taken into account. We are concerned with CSPs where Variables are sets of integers. It is shown in [34] that a CSP with Set Variables with at least one binary constraint (i.e. a constraint that involves two Variables) has exponential complexity.

Logic Programming [35], as the name suggests is based on the idea "that logic can be used as a programming language". A logic program (i.e. a program written in a logic programming language) is a sequence of sentences in logical form, each of which expresses a fact or a rule for a given domain. More precisely, a logic program consists of clauses named *Horn Clauses*. Horn clauses can have the form of a *fact*, e.g. *likes (mary,john)*, denoting that "*mary likes john*", or a *rule*, e.g. *parent(x,y)∧male(x)→ father(x,y),* denoting that for any unknown individuals x and y, in case x is *parent* of y and x is *male* then x is *father* of y. Horn clauses can also have the form of a *goal*, e.g. *father(x,mary)→*. In this case, the goal is said to *be satisfied* if there is an individual x that is *father* of *mary*. In particular, Prolog [36] is a practical logic programming language based on Horn clauses.

Constraint Programming can be hosted by a Logic Programming language. Then it is referred to as Constraint Logic Programming [37, 38].

We will be working with ECLiPSe Prolog, a software system implementing Prolog that also offers libraries for Constraint Programming. We will be using ECLiPSe's *ic* library that supports finite domain constraints, as well as the *ic_sets* library which implements constraints over the domain of finite sets of integers and cooperates with ic.

The constraint propagation algorithm that ECLiPSe's solver uses for Set Variables has a complexity $O(ld + (e - l)dd')$, where l is the number of inclusion constraints, e the number of total constraints, d the sum of cardinalities of the largest domain bounds and d' their difference [30]. More about ECLiPSe Prolog can be found in [29].

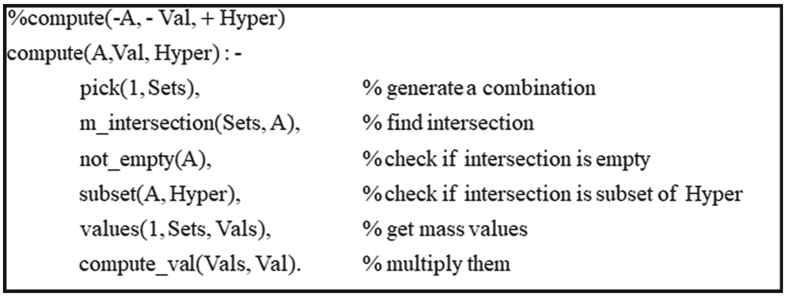
5 Constraint Programming for Computing Dempster's Rule of Combination

To face the complexity problem that Dempster's Rule of Combination introduces, we use Constraint Programming to perform the computation. The idea behind using CP is to reduce the number of computations needed to evaluate the Rule to the ones that are absolutely necessary. Constraint Logic Programming (CLP) has been chosen for the sake of simplicity and prototype experimentation. The Complexity of the Rule is directly related to the number of focal points each mass function has, as we have to evaluate all combinations of focal points. Each combination of focal points might either intersect to a set or not. By using Constraint Programming, we enumerate only those combinations of focal points whose intersection is not empty. We know beforehand that this method does not improve the computational class of the Rule, as in worst case, all combinations intersect, but nevertheless it might reduce the number of computations needed, and thus the time needed, to compute Dempster's Rule of Combination.

In addition, we use Constraint Programming to compute *belief/plausibility* for a given set. Recall that *belief* of a set $A \subseteq U$, where U is the Universe, is the sum of the mass values of all subsets of A. When computing the *belief* of a set A, given mass functions $m_1, \ldots, m_n$, we use constraints so that only the desired combinations of focal points, i.e. those that intersect to a subset of A, are created. In the general case, we need also to compute the normalization constant, K, in order to normalize the value, which means that every intersecting combination (or every non-intersecting combination) will have to be evaluated anyway, and thus the number of combinations to be evaluated is not reduced. However, even in the special case where we know that $K = 0$, that is there is no conflict between the different sources of evidence, we trust that constraints will be proved useful for computing the *belief* of a set A.

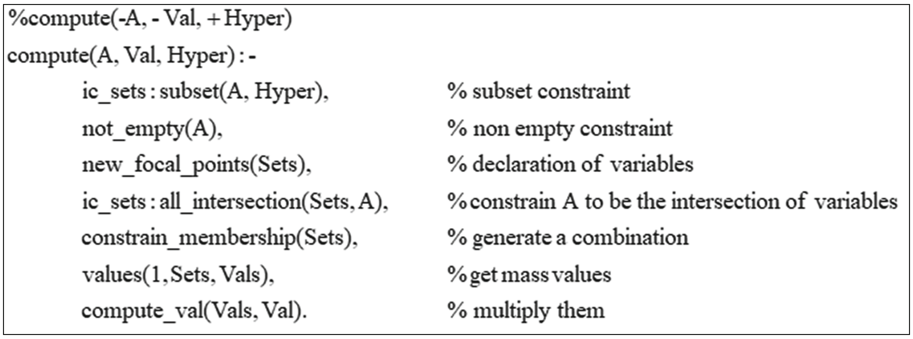
5.1 Implementation

We implemented both algorithms, i.e. the *generate-and-test* and the *constrain-and-generate*, in *ECLiPSe Prolog*, so that we accomplish a more fair time comparison. The generate-and-test algorithm evaluates every possible combination of focal points and, then, the unnecessary results are discarded. The *constrain-and-generate* one exploits CP and constraints are set in order to generate only combinations with the desired properties. In the following figures, the fundamental predicates for both algorithms are presented (Figs. 1 and 2).



```
%compute(-A, - Val, + Hyper)
compute(A,Val, Hyper) : -
        pick(1, Sets),                  % generate a combination
        m_intersection(Sets, A),        % find intersection
        not_empty(A),                   % check if intersection is empty
        subset(A, Hyper),               % check if intersection is subset of Hyper
        values(1, Sets, Vals),          % get mass values
        compute_val(Vals, Val).         % multiply them
```

Fig. 1. generate-and-test



```
%compute(-A, - Val, + Hyper)
compute(A, Val, Hyper) : -
        ic_sets : subset(A, Hyper),             % subset constraint
        not_empty(A),                           % non empty constraint
        new_focal_points(Sets),                 % declaration of variables
        ic_sets : all_intersection(Sets, A),    % constrain A to be the intersection of variables
        constrain_membership(Sets),             % generate a combination
        values(1, Sets, Vals),                  % get mass values
        compute_val(Vals, Val).                 % multiply them
```

Fig. 2. constrain-and-generate

The predicate *compute/3* is used to compute every combination of focal points whose intersection is a non-empty subset of *Hyper* for both the *generate-and-test* and the *constrain-and-generate* algorithms. This predicate is called while computing the *belief/plausibility* of a desired set, as well as the *joint mass*.

5.2 Test Results

In order to compare both methods and examine whether the use of Constraint Programming reduces the time needed for the computation we created a number of random test cases on which we ran both programs. When creating random sets, we experimented with different values of the following parameters: i) the cardinality of Universe, ii) the number of mass functions, and iii) the number of focal points per mass function (for simplicity this is the same for every mass function). Both methods were run on each test case and the time needed for the execution was recorded.

The values of many parameters were highly influenced from values used in [8]. A Universe of size $|U| = 20$ was assumed as a basis for most of the tests. As focal points are created at random, choosing a small Universe results in mass functions sharing the same focal points (high "density"), whereas picking a large Universe will result in mass functions with different focal points (low "density"). We found that the value $|U| = 20$ keeps a good balance between a "sparse" and a "dense" case.

In the following, the first part is concerned with comparing the two methods for evaluating Dempster's Rule of Combination for every possible set so as to compute the *joint mass*. Next, we focus on computing *belief* for a specific set.

Computing the Joint Mass

To demonstrate the time gain of the constrain-and-generate algorithm as discussed in Sect. 3, we created a number of random test cases with fixed parameters and recorded the number of combinations that intersect and time needed for the computation for both the *generate-and-test*, and the *constrain-and-generate* algorithms. Notice that the time needed for the *constrain-and-generate* algorithm is proportional to the number of intersecting combinations. On the other hand, the *generate-and-test* algorithm always evaluates all possible combinations and its run-time is unrelated to the number of intersecting combinations, but depends on the total number of possible combinations. Some sample test cases are presented below (Figs. 3, 4, 5 and Tables 1, 2).

Table 1. |U| = 20, number of focal points per mass function = 3

Number of mass functions	*generate-and-test* (s)	*constrain-and-generate* (s)
10	0.58	0.22
11	1.77	1.28
12	5.39	0.52
13	8.45	0.20
14	21.55	0.66
15	113.41	0.86

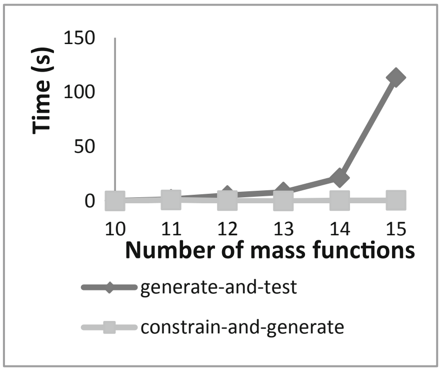


Fig. 3. |U| = 20, number of focal points per mass function = 3

Table 2. |U| = 20, number of mass functions = 10

Number of focal points per mass function	*generate-and-test* (s)	*constrain-and-generate* (s)
3	0.58	0.22
4	7.94	4.97
5	66.70	8.23
6	249.36	56.89

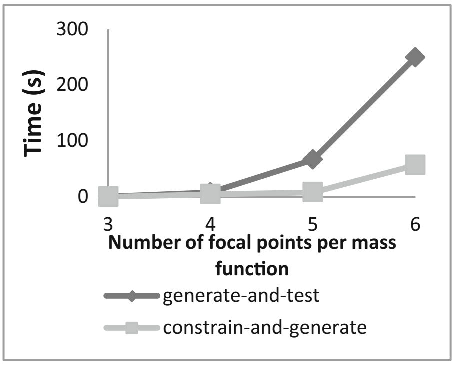


Fig. 4. |U| = 20, number of mass functions = 10

Table 3. |U| = 20, number of mass functions = 5, number of focal points per mass function = 10

Possible combinations	Intersecting combinations	*constrain-and-generate* (s)
100000	22240	2.17
	25204	2.36
	36753	3.05
	39085	3.16
	41821	3.84
	48379	3.98
	53184	4.68

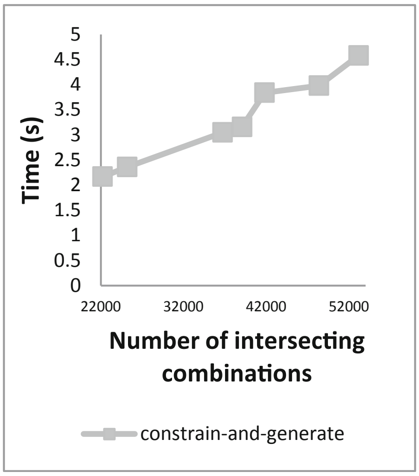


Fig. 5. |U| = 20, number of mass functions = 5, number of focal points per mass function = 10

Table 3 does not contain time results for the generate-and-test algorithm as it did not execute within an acceptable time.

Computing Belief

In this section, we discuss the results of the algorithms regarding computing *belief* of a specific set. Note that, similar results hold for computing *plausibility*, as we can compute *plausibility* (resp. *belief*) for a set by computing the *belief* (resp. *plausibility*) of its complement. Recall that when computing the *belief* of a set the normalization factor K must be evaluated in order to normalize the result. Note that, computing K is of the same complexity as computing the *joint mass* and the benefit of using CP in this case was discussed in the previous paragraph. So, we considered a case where $K = 0$ in order to examine the extra benefit by using CP in the case of belief evaluation.

We worked as before, creating random test cases so as to compare both methods. The number of all possible combinations is q^n, where n is the number of mass functions, and q is the number of focal points for each mass function. We have already mentioned that the *generate-and-test* (*g-t*) algorithm has to evaluate all q^n combinations and then discard those that do not intersect to the desired set, i.e. a subset of the set under consideration, whereas the *constrain-and-generate (c-g)* one sets constraints to be satisfied, so as to avoid evaluating every possible combination. We tested with different sizes of Universe, number of focal points per mass function, and number of mass functions. The run-time for both algorithms and the number of combinations that had to be evaluated for each set were recorded. The last set in each trial was the Universe itself. In all test cases for sets different from the Universe, improvement was noticed on run-time. A sample result is shown below.

Table 4. |U| = 20, number of mass functions = 13, number of focal points per mass function = 3

Set	Combinations	*g-t* (s)	*c-g* (s)
[1, 7, 9]	0	4.438	0.156
[4, 5, 9, 10]	14848		1.422
[2, 3, 6, 9, 10]	15360		1.625
[2, 6, 8]	15888		1.828
[2, 4–7, 10]	16128		1.719
[4, 8–10]	30736		3.313
[1, 2, 4, 5, 8–10]	31488		3
[2, 3, 7, 8, 10]	31788		3.5
[1, 3, 4, 6–8, 10]	31808		3.234
[1–8, 10]	32256		4.391
U	32768		4.438

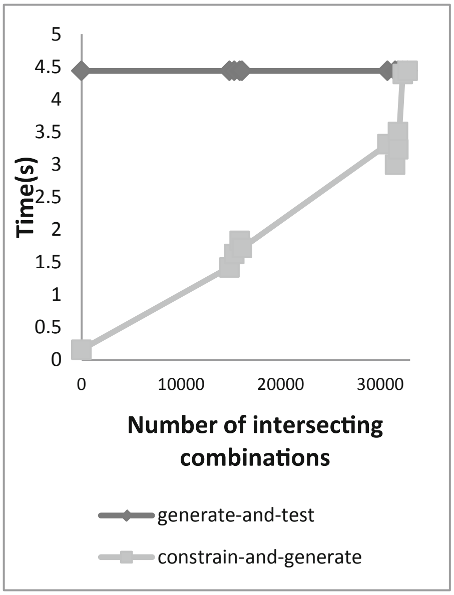


Fig. 6. |U| = 20, number of mass functions = 13, number of focal points per mass function = 3

Table 4 and Fig. 6 highlight the importance of constraints for computing *belief* for sets that are smaller than the Universe, or, otherwise stated, sets that are formed out by fewer combinations than the Universe.

To sum up, from the test results verified that Constraint Programming can help to reduce the time needed for computing Dempster's Rule of Combination whether we wish to compute the *joint mass* or *belief/plausibility* for a set. How much we can benefit from CP, depends deeply on the number of intersections that must be evaluated, which, unfortunately, is unknown prior to the computation. In general, the *constrain-and-generate's* performance is remarkable in cases where empty intersections exist, as it executes the calculation much faster than the *generate-and-test* method.

6 Conclusions and Future Work

Dempster-Shafer theory remains one of the most expressive frameworks for reasoning under uncertainty. However, the high complexity of Dempster's rule of combination imposes a significant restriction to its application. This becomes worse, considering the tremendous amount of data available. The method that we proposed to overcome this problem makes use of Constraint Programming to optimize the evaluation of Dempster's rule by computing only the appropriate combinations. The conclusion that can be drawn from the empirical testing that we performed is that the *constrain-and-generate* algorithm utilizing Constraint Programming needs considerable less time to compute the *joint mass*, or *belief/plausibility* of a specific set, than the *generate-and-test* algorithm. Therefore, such a method allows the computation of Dempster's rule of combination to be performed in an acceptable time even for more complex cases.

However, concerning the implementation platform, we have to make some remarks. *ECLiPSe Prolog* is one of the first systems to embed Constraint Programming libraries. While working with the *ic_sets* library, though, we encountered some anomalies. In cases where the combinations that have to be evaluated are significantly large, even the generation part of the *constrain-and-generate* method takes too much time, even more than the time needed by the simple *generate-and-test*. As far as we know, this could be an overhead of the library itself, or the way it handles some constraints.

As we have already pointed out, Constraint Programming allowed us to avoid generating non intersecting combinations. Notice that we can compute the normalization factor, *K,* by generating either all intersecting combinations or all non-intersecting ones. An idea that we would like to exploit is the concurrent computation of the intersecting and non-intersecting combinations in order to evaluate the normalization factor. This approach should ensure the fastest termination in all cases.

As mentioned, our implementation provides us with a prototype experimentation. It would be worthwhile to compare the use of constraint programming with approximation methods using data from a real-life application so as the comparison depicts the benefit we would gain in real world. Moreover, we believe it would be meaningful to compare our method with methods using the Fast Mobius Transformation [39–41]. In this case, another CP platform may be considered.

Acknowledgements. This research was partially funded by the National and Kapodistrian University of Athens Special Account of Research Grants no 13233.

References

1. Shafer, G.: A mathematical theory of evidence turns 40. Int. J. Approx. Reasoning **79**, 7–25 (2016)
2. Dubois, D., Prade, H.: Possibility theory, probability theory and multiple-valued logics: a clarification. Ann. Math. Artif. Intell. **32**(1), 35–66 (2001)
3. Pearl, J.: Fusion, propagation, and structuring in belief networks. Artif. Intell. **29**(3), 241–288 (1986)
4. McDermott, D., Doyle, J.: Non-monotonic logic I. Artif. Intell. **13**(1–2), 41–72 (1980)
5. Dempster, A.P.: Upper and lower probabilities induced by a multivalued mapping. Ann. Math. Stat. **38**(2), 325–339 (1967)
6. Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press, Princeton (1976)
7. Shafer, G.: Perspectives on the theory and practice of belief functions. Int. J. Approximate Reasoning **4**(5–6), 323–362 (1990)
8. Zadeh, L.A.: Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems. World Scientific Press, Singapore (1996)

9. Zadeh, L.A.: Fuzzy sets. Inf. Control **8**(3), 338–353 (1965)
10. World Wide Web Consortium. Semantic Web. https://www.w3.org/standards/semanticweb/. Accessed 13 Feb 2020
11. Laskey, K.J., Laskey, K.B., Costa, P.C.G., Kokar, M.M., Martin, T., Lukasiewicz, T.: Uncertainty reasoning for the world wide web. W3C Incubator Group Report, Technical report (2008)
12. Shafer, G.: Dempster's rule of combination. Int. J. Approx. Reasoning **79**, 26–40 (2016)
13. Lukasiewicz, T., Straccia, U.: Managing uncertainty and vagueness in description logics for the semantic web. Web Semant. **6**(4), 291–308 (2008)
14. Ortega, F.B.: Managing vagueness in ontologies. Ph.D. dissertation, Universidad de Granada (2008)
15. Stoilos, G., Stamou, G.B., Tzouvaras, V., Pan, J.Z., Horrocks, I.: Fuzzy OWL: uncertainty and the semantic web. In: Proceedings of the OWLED*05 Workshop on OWL: Experiences and Directions, Galway, Ireland (2005)
16. Karanikola, L., Karali, I.: Towards a Dempster-Shafer fuzzy description logic - handling imprecision in the semantic web. IEEE Trans. Fuzzy Syst. **26**(5), 3016–3026 (2018)
17. Karanikola, L., Karali, I.: Semantic web and ignorance: Dempster-Shafer description logics. In: FLAIRS Conference 2017, pp. 68–73 (2017)
18. Zadeh, L.A.: A simple view of the Dempster-Shafer theory of evidence and its implications for the rule of combination. AI Mag. **7**(2), 86–90 (1986)
19. Pei, W.: A defect in Dempster-Shafer theory. In: Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence, Seattle, pp. 560–566 (1994)
20. Khan, N., Anwar, S.: Time-domain data fusion using weighted evidence and Dempster-Shafer combination rule: application in object classification. Sensors (Basel) **19**(23), 5187 (2019). https://doi.org/10.3390/s19235187
21. S. Mckeever, J. Ye, A Comparison of Evidence Fusion Rules for Situation Recognition in Sensor-Based Environments. In: Communications in Computer and Information Science. pp. 163–175 (2013). https://doi.org/10.1007/978-3-319-04406-4_16
22. Orponnen, P.: Dempster's rule of combination is #P-complete. Artif. Intell. **44**(1–2), 245–253 (1990)
23. Wilson, N.: Algorithms for Dempster-Shafer theory. In: Kohlas, J., Moral, S. (eds.) Handbook of Defeasible Reasoning and Uncertainty Management Systems, vol. 5, pp. 421–475. Springer, Dordrecht (2000). https://doi.org/10.1007/978-94-017-1737-3_10
24. Voorbraak, F.: A computationally efficient approximation of Dempster-Shafer theory. Int. J. Man Mach. Stud. **30**(5), 525–536 (1989)
25. Tessem, B.: Approximations for efficient computation in the theory of evidence. Artif. Intell. **61**(2), 315–329 (1993)
26. Lowrance, J.D., Garvey, T.D., Strat, T.M.: A framework for evidential-reasoning systems. In: 5th National Conference on Artificial Intelligence, Menlo Park, California, pp. 896–903 (1986)
27. Bauer, M.: Approximation algorithms and decision making in the Dempster-Shafer theory of evidence – an empirical study. Int. J. Approximate Reasoning **17**(2–3), 217–237 (1997)
28. Mayoh, B., Tyugu, E., Penjam, J.: Constraint Programming. Springer, Heidelberg (1993). https://doi.org/10.1007/978-3-642-85983-0
29. The ECLiPSe Constraint Programming System. https://eclipseclp.org/. Accessed 13 Feb 2020
30. Gervet, C: Conjunto: constraint logic programming with finite set domains. In: ILPS (1994)
31. Eclipse conjunto libray. http://eclipseclp.org/doc/bips/lib/conjunto/index.html. Accessed 13 Feb 2020

32. Kohlas, J., Monney, P.-A.: Propagating belief functions through constraint systems. Int. J. Approximate Reasoning **5**(5), 433–461 (1991)
33. Rouahi, A., Ben Salah, K., Ghédira, K.: Belief constraint satisfaction problems. In: 12th International Conference of Computer Systems and Applications (AICCSA), Marrakech, pp. 1–4 (2015)
34. Aiken, A., Kozen, D., Vardi, M., Wimmers, E.: The complexity of set constraints. In: Börger, E., Gurevich, Y., Meinke, K. (eds.) CSL 1993. LNCS, vol. 832, pp. 1–17. Springer, Heidelberg (1994). https://doi.org/10.1007/BFb0049320
35. Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag, Heidelberg (1984). https://doi.org/10.1007/978-3-642-96826-6
36. Colmerauer, A., Kanoui, H.: Philippe Roussel et Robert Pasero, Un système de communication homme-machine en Français, rapport de recherche. Groupe de recherche en Intelligence Artificielle, Marseille (1973)
37. Jaffar, J., Lassez, J.L.: Constraint logic programming. In: Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, Munch, West Germany, pp. 111–119 (1987)
38. Van Hentenryck, P.: Constraint Satisfaction in Logic Programming. MIT Press, Cambridge (1989)
39. Kennes, R., Smets, P.: Computational aspects of the Mobius transformation. In: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence, pp. 401–416. Elsevier Science Inc., USA (1990)
40. Kennes, R., Smets, P.: Fast algorithms for Dempster-Shafer theory. In: Uncertainty in Knowledge Bases. IPMU 1990, Paris, France, pp. 14-26 (1990)
41. Chaveroche, M., Davoine, F., Cherfaoui, V.: Efficient Möbius transformations and their applications to D-S theory. In: 13th International Conference on Scalable Uncertainty Management, Compiègne, France, pp. 390–403 (2019)