NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Dempster-Shafer Theory Application in Recommender Systems and Comparison of Constraint Programming's and Möbius Transform's Implementations

Tatiana P. Boura

Supervisor: **Isambo Karali,** Assistant Professor

ATHENS

JULY 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή της Θεωρίας Dempster-Shafer σε Συστήματα Συστάσεων και Σύγκριση Χρήσης Προγραμματισμού με Περιορισμούς και του Αλγόριθμου Möbius Transform για τον Υπολογισμό της

Τατιάνα Π. Μπούρα

Επιβλέπουσα: **Ιζαμπώ Καράλη,** Επίκουρη Καθηγήτρια

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2022

BSc THESIS

Dempster-Shafer Theory Application in Recommender Systems and Comparison of Constraint Programming’s and Möbius Transform’s Implementations

Tatiana P. Boura
S.N.: 1115201700100

SUPERVISOR: **Izambo Karali**, Assistant Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή της Θεωρίας Dempster-Shafer σε Συστήματα Συστάσεων και Σύγκριση Χρήσης Προγραμματισμού με Περιορισμούς και του Αλγόριθμου Möbius Transform για τον Υπολογισμό της

Τατιάνα Π. Μπούρα

Α.Μ.: 1115201700100

ΕΠΙΒΛΕΠΟΥΣΑ: **Ιζαμπώ Καράλη**, Επίκουρη Καθηγήτρια

ABSTRACT

In this thesis, we deal with the subject of Handling Uncertainty in the field of Knowledge Representation using Dempster-Shafer theory. Our goal is to study an application of Dempster-Shafer theory to Recommended Systems and measure the performance of Dempster's rule and belief computation when using an implementation that utilizes Constraint Logic Programming (CLP). Also, we aim to compare the performance of the CLP implementation on random test cases to the performance of an implementation of Dempster-Shafer theory using Möbius Transforms. In general, the computational time for the application was rational. Regarding the comparison, each implementations has its pros and cons.

SUBJECT AREA: Artificial Intelligence

KEYWORDS: Dempster-Shafer theory, Uncertainty, Möbius Transform, Recommender Systems, ECLiPSe Prolog, ibelief, Comparison

ΠΕΡΙΛΗΨΗ

Στην πτυχιακή εργασία αυτή μας απασχολεί το θέμα του Χειρισμού Αβεβαιότητας στον τομέα της Αναπαράστασης Γνώσης χρησιμοποιώντας τη θεωρία των Dempster-Shafer. Σκοπός μας είναι να μελετήσουμε μια εφαρμογή της θεωρίας σε Συστήματα Συστάσεων και να μετρήσουμε την απόδοση του κανόνα του Dempster και του υπολογισμού εμπιστοσύνης χρησιμοποιώντας μια υλοποίηση της θεωρίας βασισμένη στον Λογικό Προγραμματισμό με Περιορισμούς. Ακόμη, επιθυμούμε να συγκρίνουμε την απόδοση της υλοποίησης βασισμένη στον Λογικό Προγραμματισμό με Περιορισμούς σε τυχαίες περιπτώσεις δοκιμής με μια υλοποίηση που χρησιμοποιεί μετασχηματισμούς Möbius. Σε γενικά πλαίσια ο υπολογιστικός χρόνος για την εφαρμογή υπήρξε λογικός. Όσον αφορά τη σύγκριση, και οι δύο περιπτώσεις είχαν θετικά και αρνητικά.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή Νοημοσύνη

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Θεωρία των Dempster-Shafer, Χειρισμός Αβεβαιότητας, Möbius Transform, Συστήματα Συστάσεων, ECLiPSe Prolog, ibelief, Σύγκριση

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr.Izambo Karali, first of all, for introducing me to the subject of Uncertainty and for letting me work with different programming languages and environments as I requested from the start. Also, I am grateful to her for the constant communication and guidance and for answering even the tiniest questions. With her as a mentor I learned to study and understand foreign bibliography and scientific articles and though this process I discovered the majestic world of research. For all the above reasons I never had any seconds though about my thesis, even when things got rough, and I have only her to thank for that.

Also, I am thankful to Mr. Alexandros Kaltsounidis, as without his work in his BSc thesis, I would not have been able to conduct mine.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In the recent years, we've noticed a thrive in fields of Computer Science such as Artificial Intelligence (AI), Machine Learning (ML) and Data Science. The intriguing part is that most of the algorithms used in those fields where implemented a few decades ago. The reason why there is this burst of attention regarding them is because nowadays we have the data and the computational power. Essentially, this means that now we can collect many data and information and process it in order to come to certain conclusions.

This newly obtained amount of information has lead to the growth of the fields "Knowledge Handling" and "Knowledge Representation", where experts care to receive knowledge, understand it and use it. However, knowledge is not always absolute and easily obtainable. So, in many cases there is *Uncertainty* involved, which can occur from the random behavior of a system, the lack of knowledge about a system or the unreliable sources of information. Understandably, this has been a trouble to scientists for many years and since uncertainty cannot be overtaken, many theories for handling it have been developed. These theories suggest different approaches to reasoning under uncertainty. A pretty interesting one was framed theoretically by Glenn Shafer and mathematically defined by Arthur P. Dempster and is called Dempster-Shafer [30, 31, 10]. Dempster-Shafer theory(DST), evidence theory or theory of belief functions is a generalization of the Bayesian theory of subjective probability. With this method, if there is a substantial amount of questions related to an initial question, we can find a satisfying Bayesian analogy.

Although Dempster-Shafer theory has some similarities with Probability Theory it should not be confused with it. The belief functions used in Dempster-Shafer use probabilities in order to achieve a degree of belief. Sometimes these degrees of belief have similar properties to probabilities, but that does not always happen. The most significant difference is that Dempster-Shafer theory allows a source to assign credit to a group of states, whereas Bayesian theory can be used to represent the belief in only one state. The mass function, that will be explained later on, is the means to do the above assignment of belief and thus cannot be equated with a classical probability in general.

Dempster-Shafer theory deals well with uncertainty which we can either describe as insufficient, incomplete or controversial knowledge of some fact or as lack of information on which to evaluate a probability. As we mentioned already, uncertainty and randomness can be modeled by assigning degrees of belief to groups of states, but classic Probability Theory would be a poor choice in some type of problems. For example, given a fact (*evidence*) $A = "Covid19\ will\ be\ a\ non-existent\ disease\ in\ the\ year\ 2030$" and $P(A) = 0.7$, in Probability Theory someone would say (without any doubt) that the complement fact of A, $\overline{A} = "Covid19\ will\ be\ an\ existent\ disease\ in\ the\ year\ 2030$" has the probability of $P(\overline{A}) = 0.3$ happening, as it is well known that $P(\overline{A}) = 1 - P(A)$. But, in some situations knowing about A does not give us straightforward knowledge about the exact opposite fact and vice versa and, thus, these problems cannot be correctly modeled using probabilities. For this reason, classic Probability Theory is not always chosen for representing knowledge when there is uncertainty involved.

At this point, it must be noted that probabilities described in the classical way are suitable for problems where there is *Aleatory Uncertainty* involved, which is the type of uncertainty that results from the fact that a system can behave in random ways. It has been observed that when probabilities are used to characterized *Subjective Uncertainty*, which is the type of uncertainty that results from the lack of knowledge about a system and is a property of the analysts performing the analysis, a great amount of times the characterization is poor. Subjective Uncertainty is involved in many problems related to Artificial Intelligence and Machine Learning, so Dempster-Shafer theory seems to be ideal for those kind of problems. Of course, before applying the DS method to any AI problem, one should take into consideration if that AI application is based on the frequency of the data's appearance or on some subjective belief.

Dempster-Shafer theory utilizes a mathematical entity called *mass function* in order to assign confidence about a question. From these mass functions one can obtain the amount of certainty, or as called *belief* for a question related to the events in our system. Also we are able to combine these assignments, through *Dempster's rule of combination* in order to produce a unique confidence assignment for the system. However, the computation of Dempster's rule is a #P-complete problem and cannot be performed in an acceptable time for a large amount of data. These definitions and more characteristics of DST are studied in Chapter 2.

Because *Dempster's rule* has exponential computational complexity, a lot of research has been done in order to reduce this complexity. Some algorithms that prove to do so, are algorithms based on Möbius Transforms. In Chapter 3 we discuss these algorithms and the theory behind them.

In Chapter 4 we introduce two implementations of the DST, one based on Möbius Transforms, as we studied their relation to the theory, and a second one based on CLP. The first one was implemented by CRAN and the second one by Alexandros Kaltsounidis in his BSc Thesis [16]. In later chapters we will be comparing them in terms of computational time and other characteristics of the data.

Dempster-Shafer theory seems ideal for modeling real-life situations and problems. For that reason, many applications of DST can be found in bibliography. In Chapter 5 we study an application to Recommender Systems [33]. In Chapter 6 we use a dataset to measure its performance using the CLP implementation when computing the combined mass functions and the belief of some sets. The results are promising and the computations where completed in a logical time frame.

Continuing to Chapter 7 of this dissertation, we compare the two implementations presented in Chapter 4 on randomly generated test cases. In retrospect, the implementation using constraints completes the computation of Dempster rule in acceptable time when the total number of combinations is small, whereas the implementation using Möbius Transforms is better when the possible number of events in our system is relatively small. For the computation of the belief function such comparison could not be made, for reasons that will be explained.

Finally, in Chapter 8 we provide a general conclusion of our work in this thesis and mention ideas to expand our research.

2. DEMPSTER-SHAFER THEORY

Dempster-Shafer theory (DST) [30, 31, 10] is a framework designed for reasoning under uncertainty, that allows us to combine evidence from different independent sources. Through the mass function or basic probability assignment (bpa), degrees of confidence are assigned to an event. These mass functions can also be combined with Dempster's rule of combination in order to produce a combined bpa. All the above will be studied in this chapter.

2.1 Definition of the theory

Below, the DS theory will be presented using the Axiomatic Approach.

As U (*Universe* or Θ) is denoted the set containing all possible states of a system under consideration or, more intuitively, all possible outcomes of an event. We are explicitly interested in the powerset of U: 2^U that includes all subsets of U thus, all combinations of all possible outcomes, because we are looking for evidence to be associated with multiple possible events in order to accomplish a higher level of abstraction.

The most fundamental function for assigning degrees of belief to a set of hypotheses *h* such as *h* $\in 2^U$ is the **mass function** m (known as bpa).

Before the function m is formally defined, it should be noted that DS is suitable when there is no objective way of assigning belief values to the powerset. Here is an example to make this statement understandable:

Example 2.1.1. *Imagine a witness to a robbery being called into the police station to identify the robber. The officers ask the witness what was the colour of the robber's hair, but because it was dark the witness cannot give an answer with certainty. Consequently, they grant the belief of* $A = \{black \mid The\ robber\ has\ black\ hair\}$ *the value* $m(A) = 0.6$*,* $B = \{blonde \mid The\ robber\ has\ blonde\ hair\}$ *the value* $m(B) = 0.1$ *and* $C = \{black \vee blonde \mid The\ robber\ has\ either\ black\ or\ blonde\ hair\}$ *the value* $m(C) = 0.3$*. These values are representative of the* ***independent*** *belief of this witness, which means that another independent witness may have assigned different values to each hypotheses.*

That is a problem that classical probability theory would model poorly. As G.Shafer quoted in [31]: "The theory of belief functions provides one way to use mathematical probability in subjective judgement" and as mentioned before, alternatives to Probability Theory are preferred when the system suffers from *Subjective Uncertainty*.

Also, assigning a belief value to a set $A \in 2^U$ does not provide any information about the degree of belief of subsets of A. Going back to the Example 2.1.1 we notice that although $A \subset C$ and $B \subset C$, $m(C) > m(B)$ but $m(C) < m(A)$. So, the mass function of a given set A, expresses the proportion of all relevant and available evidence that supports the claim that a particular element of U belongs to A but to no particular subset of A.

Now, we formally define as **mass function** m a function $m : 2^U \to [0,1]$ with the following properties:

1. $m(\emptyset) = 0$
2. $\sum_{A \in 2^U} m(A) = 1$

Note that in Example 2.1.1: $m(\emptyset) = 0$ and $\sum_{X \in 2^U} m(X) = m(\emptyset) + m(A) + m(B) + m(C) = 1$.

The bpa is a useful means for computing the lower (*belief*) and the upper bound (*plausibility*) of a probability interval for the set of interest, as defined later on.

It should be noted that every $A \in 2^U$ that has non-zero bpa is called a *focal element*.

2.1.1 Belief Function

The belief of a set $A \in 2^U$, denoted as $bel(A)$ is the total amount of certainty that the event A has happened and is defined as follows:

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

From the properties of the mass function we can easily presume that:

$$bel(\emptyset) = 0 \text{ and } bel(U) = 1$$

Since belief function defines the lower bound of a probability of an event A:

$bel(A) \leq P(A)$, where $P(A)$ is the probability of the set A in the classical probabilistic definition. *(1)*

2.1.2 Plausibility Function

The belief of a set $A \in 2^U$, denoted as $pl(A)$ is the total amount of certainty that not the event A has happened and is defined as follows:

$$pl(A) = bel(U) - bel(\overline{A}) \xRightarrow{\text{bel(U) = 1}} pl(A) = 1 - bel(\overline{A})$$

Also, since for $A, B \in 2^U$, $A \subseteq B$ iff $A \cap \overline{B} = \emptyset \Rightarrow bel(A) = 1 - pl(\overline{A})$, because

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

From the properties of the mass function and belief function we can easily presume that:

$pl(\emptyset) = 1 - bel(U) = 0$ and $pl(U) = 1 - bel(\emptyset) = 1$

Theorem 2.1.1. *The plausibility of a set A can be computed as* $pl(A)$ = $\sum_{B\cap A\neq\emptyset} m(B), \quad \forall A \subseteq U.$

Proof. As defined, $\forall A \subseteq U$, $pl(A) = bel(U) - bel(\overline{A})$ = $\sum_{X\subseteq U} m(X) - \sum_{Y\subseteq\overline{A}} m(Y)$ *(2)*, which means that from the masses of all subsets of U we subtract the masses of the subsets of $\overline{A}$, so now we have the masses of the subsets of U that intersect with $A \Rightarrow$ *(2)* = $\sum_{B\cap A\neq\emptyset} m(B)$.

□

Since plausibility function defines the upper bound of a probability of an event A:

$pl(A) \geq P(A)$, where $P(A)$ is the probability of the set A in the classical probabilistic definition. *(3)*

Conclusively,

$$\xRightarrow{(1),(3)} bel(A) \leq P(A) \leq pl(A), \quad \forall A \subseteq U \text{ (4)}$$

In *(4)* bel and pl are considered upper and lower probabilities as introduced by Dempster in [10].

When $bel(A) = pl(A)$ all probabilities $P(A)$ are uniquely determined $\forall A \subseteq U$, which corresponds to classical probability.

However, as noted in [31] the degrees of belief given by belief and plausibility functions should not be interpreted as lower or higher bounds respectively on some unknown true probability. If we wanted to define a probability-bound interpretation of belief and plausibility functions, then we would show interest only in groups of belief and plausibility functions whose degrees of belief, when interpreted as probability bounds, can be satisfied simultaneously.

After seeing *(4)* we understand that the model of DST is designed to cope with varying levels of precision based on the current information without any further assumptions.

Let's compute the bel and pl function for Example 2.1.1 for $A = \{black\}$, $A \in 2^{\{black,blonde\}}$:

$$\begin{aligned} bel(A) &= \textstyle\sum_{B \subseteq A} m(B) = m(\emptyset) + m(black) = 0 + 0.6 = 0.6 \\ pl(A) &= \textstyle\sum_{B \cap A \neq \emptyset} m(B) = m(\emptyset) + m(black) + m(black \vee blonde) \\ &\qquad\qquad\quad = 0 + 0.6 + 0.3 = 0.9. \end{aligned}$$

So, the probability of the robber's hair being black is $0.6 \leq P(black) \leq 0.9$.

2.1.2.1 Belief Intervals

As defined before $P(A)$, that is the probability of a set A in the classical probabilistic definition, belongs to the bounded interval $[bel(A), pl(A)]$. Since this interval is the indicator of the probability's value, it should be studied in order to determine the possibility of the event A occurring. The width w of the interval is

$$w = |pl(A) - bel(A)| = |1 - bel(\overline{A}) - bel(A)| = 1 - bel(\overline{A}) - bel(A).$$

In the Bayesian situation we know that $bel(\overline{A}) + bel(A) = 1 \Rightarrow w = 0$.
Since in DST $bel(\overline{A}) + bel(A) \leq \sum_{X \in 2^U} m(X) = 1 \Rightarrow w > 0$.

The width of a belief interval (w) can also be interpreted as the amount of uncertainty with respect to a hypothesis, given the evidence. More specifically, w is belief that is committed by the evidence to neither the hypothesis nor the negation of the hypothesis [13].

Even though w is a good indicator, the amount of uncertainty is not only defined by the width of the belief interval, but also by the values of $pl(A)$ and $bel(A)$ respectively. If $pl(A)$ and $bel(A)$ are evaluated low, we are quite certain A that the event will *not* happen.

That being said, only if w's value is close to 0 and both values of belief and plausibility are close to 1, we can be certain that the event A will happen. Consequently, if w's value is close to 0 and both values of belief and plausibility are close to 0, we can be certain that the event $\overline{A}$ will happen.

2.1.3 Dempster's rule of combination

Continuing with the Example 2.1.1, let's suppose there is another witness to the robbery and the police asks them the same question about the robber's hair. They, then, grant the belief of $A = \{black \mid The\ robber\ has\ black\ hair\}$ the value $m_2(A) = 0.7$ and $B = \{blonde \mid The\ robber\ has\ blonde\ hair\}$ the value $m_2(B) = 0.3$. This means that over the same Universe $\{black, blonde\}$ a second mass function $m_2 : 2^{\{black,blonde\}} \rightarrow [0, 1]$ is defined by the second witness. The second mass function m_2 is now needed to be combined with the mass function m defined by the first witness in order to come to a conclusion about the robber's hair colour. It is important to notice that the opinion of the second witness is ***independent*** as the second witness has not heard the opinion of the first one and vice versa. Dempster's rule of combination is used for combining degrees of belief that are based on independent items of evidence.

For the time being, we assume that all sources of information are ***reliable*** (Dempster's rule is symmetric).

The combination of these two mass functions is performed by *Dempster's rule of combination* and is denoted by the operator $\oplus$. The combination of two mass functions m_1, m_2 from the same Universe U is declared as $m_{1,2} \equiv m_1 \oplus m_2$ and is called joint mass. More specifically:

1. $m_{1,2}(\emptyset) = 0$
2. $m_{1,2}(A) = \frac{1}{1-K}\sum_{B\cap C=A} m_1(B)\cdot m_2(C), \quad \forall A \subseteq U, A \neq \emptyset$ *where* $K = \sum_{B\cap C=\emptyset} m_1(B)\cdot m_2(C)$

This rule strongly emphasizes the agreement between multiple sources and ignores all the conflicting evidence through the normalization constant K.

Given m_0, m_1, m_2 and m_3 defined over the same Universe.

Dempster's Rule of combination is relative to the following algebraic properties:

1. Commutativity : $m_1 \oplus m_2 = m_2 \oplus m_1$
2. Associativity : $(m_1 \oplus m_2) \oplus m_3 = m_1 \oplus (m_2 \oplus m_3)$
3. Neutral element: $m_0(U) = 1$, $m_0 \oplus m_1 = m_1$.

The third holds because $m_0(U) = 1$ means that we don't know exactly what is the degree of belief of each subset of U, and thus no further information is provided.

Dempster's Rule of combination is not relative to:

1. Idempotence : $m_1 \oplus m_1 \neq m_1$

In order to completely understand Dempster's rule let's consider the following example:

Example 2.1.2. *Ann and Bill are a couple and are considering buying a pet. They decided that they will choose as pet one of the:* $cat(c)$, $dog(d)$, $fish(f)$, $horse(h)$, $iguana(i)$, $parrot(p)$, *so* $U = \{c, d, f, h, i, p\}$. *Bill is more eager to buy a parrot and he tries to compute the range of the probability of getting a parrot,* $P(\{p\})$. *He gathers some evidence:*

The family lives in a farm: $m_1(\{c, d, h, p\})$ *= 0.5 and* $m_1(U) = 0.5$.

Their child is allergic to pet hair: $m_2(\{f, i, p\}) = 0.9$ *and* $m_2(U) = 0.1$*.*
They are often out of town for the weekend: $m_3(\{c, f\}) = 0.8$ *and* $m_3(U) = 0.2$*.*

Firstly, we'll be computing K_1 *for* $m_{1,2}$*,* $K = \sum_{B\cap C=\emptyset} m_1(B) \cdot m_2(C) = 0$*. Then,*

$m_{1,2}(\{p\}) = 0.5 \times 0.9 = 0.45$
$m_{1,2}(\{c, d, h, p\}) = 0.5 \times 0.1 = 0.05$
$m_{1,2}(\{f, i, p\}) = 0.5 \times 0.9 = 0.45$
$m_{1,2}(U) = 0.5 \times 0.1 = 0.05$

Also, K_2 *for* $m_{1,2,3}$*,* $K = \sum_{B\cap C=\emptyset} m_{1,2}(B) \cdot m_3(C) = m_{1,2}(\{p\}) \cdot m_3(\{c, f\}) = 0.36$*. Then,*

$m_{1,2,3}(\emptyset) = 0$
$m_{1,2,3}(\{c\}) = 0.0625$
$m_{1,2,3}(\{f\}) = 0.5625$
$m_{1,2,3}(\{c, f\}) = 0.0625$
$m_{1,2,3}(\{p\}) = 0.140625$
$m_{1,2,3}(\{c, d, h, p\}) = 0.015625$
$m_{1,2,3}(\{f, i, p\}) = 0.140625$
$m_{1,2,3}(U) = 0.015625$

Now, we compute bel *and* pl *functions as follows,*

$bel(\{p\}) = \sum_{B\subseteq\{p\}} m_{1,2,3}(B) = 0.140625$ *and* $pl(\{p\}) = \sum_{B\cap\{p\}\neq\emptyset} m_{1,2,3}(B) = 0.3125$

and so, $0.140625 \leq P(\{p\}) \leq 0.3125$*.*

2.2 Properties of DST

In this section will be showcased some not straightforward properties of the theory.

2.2.1 Combining information of different evidentiary spaces

In order to proceed we should define the term *compatibility relation* C between two spaces, S and T, that characterizes possibilistic relationships between the elements of two spaces, as stated in [35]: "An element $s \in S$ is compatible with an element $t \in T$ if s is an answer to S and t is an answer to T at the same time, or as we will from now on symbolize, sCt. The granule of s is the set of all elements in T that are compatible with s is symbolized as $G(s) = \{t \mid t \in T,\ sCt\}$. "

So, knowing a probability distribution of space S and a compatibility relation C between S

and T, one can deduce the probability distribution of space T which is the a basic probability assignment of T denoted as:

$$m(A) = \frac{\sum\limits_{G(s_i)=A} p(s_i)}{1 - \sum\limits_{G(s_i)=\emptyset} p(s_i)} \text{ where } A \in 2^T \qquad (5)$$

The above result is very useful because it gives us the opportunity to combine probabilities as they are defined by the classical way (probability distribution of space S) with a bpa as defined by the DS theory. It, also, comes in handy in real life applications as it is more likely to have indirect knowledge about one space of evidence which means having information about another space of evidence and how it is related to the one we are interested in. Let's provide an example in order for the reader to strengthen their understanding of this paragraph:

Example 2.2.1. *On a not so busy street of Athens a robbery of a jewelry store has occurred and the only witness is the owner of the shop, who called the police immediately after the incident. The police's response was quick and four suspects were arrested: Ann, Bill, Conor and Dolly, so the suspect's Universe is $U = \{Ann, Bill, Conor, Dolly\}$. Out of the four suspects Ann and Dolly identify as women and Bill and Conor identify as men. Also, Ann and Bill are less than 180 cm tall and Conor and Dolly have height equal to or more than 180 cm. The robber was wearing a mask so their only feature that could be used for suspect elimination is their height. The owner's answer when they were asked about the thief's height was: "I am 80% sure that their height was equal to or more than 180 cm".*

Using this information and defining:

$C_1 := one\ person's\ height,$ *as compatibility relation*

$S_1 := \{height < 180\ cm,\ height\ \geq 180\ cm\}$ *and* $T := U$

$p(height\ equal\ to\ or\ greater\ than\ 180\ cm) = 0.8,$ *from the owner's interview,*

the inducted bpa m_1 is:

$m_1(\{Conor, Dolly\}) = 0.8$ *and* $m_1(U) = 0.2$

because for $s := height\ equal\ to\ or\ greater\ than\ 180\ cm,\ s \in S_1$ the granule $G_1(s_1) = (\{Conor, Dolly\})$ and applying (5) we have :

$m_1(\{Conor, Dolly\}) = \frac{0.8}{1-0} = 0.8$

Due to the fact that there are no other witnesses and none of the security cameras has recorded any useful information, the investigators decide to use some statistical facts in their possession. It is true that in this area 60% of the robberies have been conducted by men. Using the above logic and defining:

$C_2 := one\ person's\ sex$, *as compatibility relation*

$S_2 := \{woman, man\}$ *and* $T := U$

$p(man) = 0.6$, *from the statistical fact,*

the inducted bpa m_2 *is:*

$m_2(\{Bill, Conor\}) = 0.6$ *and* $m_2(U) = 0.4$

Using Dempster's rule, the combined mass function $m_{1,2}$ *is:*

$m_{1,2}(\{Conor\}) = 0.48$, $m_{1,2}(\{Bill, Conor\}) = 0.12$, $m_{1,2}(\{Conor, Dolly\}) = 0.32$ *and* $m_{1,2}(U) = 0.08$

By computing the belief and plausibility for each suspect, the person that is most likely to have conducted the crime is Conor, with a probability of $0.48 \leq P(\{Conor\}) \leq 1.0$*.*

So, in the above example, the number one suspect was found because of having a probability distribution about a suspect's characteristic from an independent source combined with statistical information about another characteristic of the suspect, which proves that DS handles real life applications with ease and can lead to useful information.

2.2.2 Conducting more general conclusions

Let's assume that, given the Example 2.2.1, the police has now been informed that Ann, Bill and Dolly live in the same building. Having that new information the authorities are considering putting an undercover officer in front of the building in order to observe the moves of the three neighbours. Due to the fact that there are not many officers assigned to the case, the head-officer uses belief and plausibility definitions in order to see how possible it is that the robbery was involving one of the three suspects somehow, as they want to be sure that putting a guard will benefit the case. Therefore they compute: $bel(\{Ann, Bill, Dolly\}) = 0$ and $pl(\{Ann, Bill, Dolly\}) = 0.52$, which indicates that the probability of Ann, Bill or Dolly having conducted the crime ranges from 0 to 0.52, so, although not probable, it is a possibility worth investigating.

In this case DS was able to give us a more general information that we needed involving a group, *a set*, and not a single element. Thus, DS is optimal when different levels of *abstraction* have to be examined in order to come to a conclusion.

2.2.3 Dealing with various types of evidence and conflict

When gathering information from various independent sources, the type of evidence can be [29]:

1. *Consonant*: Every independent source gives us an additional information, so information is obtained over time and the size of evidentiary set is altered.

2. *Consistent*: There is at least one piece of information that is believed to be true by all independent sources.

3. *Arbitrary*: There is no piece of information supported by all sources, but some sources believe in the same evidence.

4. *Disjoint*: All sources give different information. In this case we have *conflicting evidence*.

The above types of evidence cannot be handled by traditional probability without proceeding to further assumptions. DS Theory, on the other hand, is able to combine many kinds of evidence. There are only two problems:

The first one is that DST with Dempster's rule as rule of combination cannot deal with *Disjoint evidence* because, theoretically, the information the evidentiary set provides us is undefined and, practically, because the value of K would be equal to *1* and, for that reason, the value of the combined mass function would be undefined. The second problem is that DST with Dempster's rule as rule of combination may produce some unwanted results when one deals with high conflict data.

Let's take a look at this example:

Example 2.2.2. *Let's suppose the two spouses Ann and Bill go to a wine tasting event. They are put into different rooms so that they won't be able to communicate with each other at all, in order for them to form their own independent opinions. The budget is very low and, thus, they are only given one glass of the same wine. After the tasting, a friend of Ann and Bill's named Conor asks them what was the kind of the wine they drank. They have to chose between three grape types:* $U = \{Merlot, Grenache, Cabernet\ Sauvignon\}$*. According to Ann* $m_A(\{Merlot\}) = 0.98$ *and* $m_A(\{Cabernet\ Sauvignon\}) = 0.02$ *and according to Bill* $m_B(\{Grenache\}) = 0.99$ *and* $m_B(\{Cabernet\ Sauvignon\}) = 0.01$*. No doubt the opinions of the two spouses are conflicting, so Conor decides to combine the information using Dempster's rule:*

$$m_{A,B}(\{Merlot\}) = \frac{1}{1-0.9998} \cdot 0 = 0$$
$$m_{A,B}(\{Grenache\}) = \frac{1}{1-0.9998} \cdot 0 = 0$$
$$m_{A,B}(\{Cabernet\ Sauvignon\}) = \frac{1}{1-0.9998} \cdot (0.01 \cdot 0.02) = 1$$

The result of the combination is that the wine's kind was $\{Cabernet\ Sauvignon\}$, even though both Ann and Bill had serious doubts about that. This happens, because Dempster's rule treats every opinion as equal and assumes all sources are reliable. Conflicting items of evidence erode each other.

However, through the years many have suggested alternative combination rules in order to deal with conflicting evidence and unreliable sources of information. All of them suggest a way of treating conflicting data such as giving different degrees of trust to a belief function or treating conflicting evidence as uncertainty etc. Some of these combination rules are *Discount and Combine rule*, *Disjunctive Polling Rule*, *Yager's rule* [34], *Inagaki's rule* [15] and a few more with equal importance. In [29] many of them are presented shortly. In this section we will review two of the above: Discount and Combine rule and Disjunctive Polling Rule.

2.2.3.1 Discount and Combine rule

This rule is mentioned as it is very intuitive and is an implementation of one's first thought when considering giving significance to an opinion. When the sources of evidence are not completely reliable, this rule applies a discounting process based on source-reliability and then combines the newly defined bpa's. The goal of discounting is to reduce global conflict before combination. The discounting operation may be defined as:

$$m(A) = \begin{cases} \alpha \times m(A), & \forall A \subset U \\ 1 - \alpha + \alpha \times m(U), & if \quad A = U \end{cases}$$

where, $\alpha \in [0,1]$ is the reliability degree of mass function m [37].

If $\alpha = 1$, the sources and thus the evidence is completely reliable and the mass function will not change, but if $\alpha = 0$, the evidence is totally unreliable and we have no knowledge about our evidentiary space, since $m(U) = 1$.

Prior to starting the discounting process one should evaluate the reliability of each source. It can be assumed that the conflict is derived from the unreliability of the sources and, therefore, the source reliability estimation is linked to some extent to the amount of conflict between different sources. So, one suggestion is to compute the relative reliability of the source after the degree of conflict is computed. For more details the reader may refer to [37].

2.2.3.2 Disjunctive rule

In order to introduce to the reader the Disjunctive rule of combination we are going to review shortly the *Transferable Belief Model (TBM)*. The TBM was introduced by Smets and Kennes in [19] as a subjectivist interpretation of DST. The TBM is a framework ideal for representing and manipulating aleatory and epistemic uncertainties. It is based on two levels: the credal level, where available pieces of information are represented by belief functions, and the pignistic or decision level, where belief functions are transformed into probability measures. In contrast to the original evidence theory the TBM propagates the open world assumption suggesting that all possible outcomes are not known. Under the open world assumption Dempster's rule of combination is adapted so there is no normalization. Also, in TMB is introduced a new combination rule: the *Disjunctive rule* as one can thoroughly read in [32].

This combination rule is more robust than *Dempster's rule* when conflicting evidence is involved, and its use is appropriate when the conflict is originated from poor reliability of some of the sources. It is similar to *Dempster's rule*, but instead of intersections it combines unions of bpa's as follows :

Disjunctive rule of combination: [11, 12, 32]

$$m_{1,2_D}(A) = \textstyle\sum_{B \cup C = A} m_1(B) \cdot m_2(C), \quad \forall A \subseteq U,\, A \neq \emptyset$$

The disjunctive rule can be used if only we assume that at least one of the sources is reliable (*consistent evidence*), but we cannot know which. Note that by assuming at least one source is reliable, we are weakening the assumption that was made by Dempster's rule. Also, just like in Dempster's rule the assumption is that the sources are independent and, thus, uncorrelated.

Because the union $B \cup C$ is empty only if both B and C are empty, there is no conflict resulting from the disjunctive rule of combination and therefore no need for normalization [25]. By definition, this rule is applied in order to combine information from different (possibly unreliable) sources, so when we come across empty unions it means that the information is contradictory and, thus, no attention is paid since we know that at least one source is reliable. Actually, when applying this rule we consider having an *Open World*, which means that our U is not exhaustive and for that matter the value of the uncertainty regarding U is found in the empty unions.

Now, we will use the data from the Example 2.2.2 and applying the Disjunctive rule.

In order for the reader to understand, we will use the Table 2.1, where the first column has the focal points of the mass function m_A and the value assigned to them, the first row has the focal points of the mass function m_B and their value and in each of the remaining cells the union of the focal points is presented as well as the product of their mass values.

Note that a similar table can be created when applying Dempster's rule. This approach was not preferred when Dempster's rule was presented as the goal then was to focus on the algebraic computation itself.

Table 2.1: Unions of focal points in Example 2.2.2 and the product of their mass values

m_A \ m_B	$\{Grenache\}$ / 0.99	$\{Cabernet\ S.\}$ / 0.01
$\{Merlot\}$ / 0.98	$\{Grenache, Merlot\}$ / 0.9702	$\{Cabernet\ S., Merlot\}$ / 0.0098
$\{Cabernet\ S.\}$ / 0.02	$\{Grenache, Cabernet\ S.\}$ / 0.0198	$\{Cabernet\ S.\}$ / 0.0002

One can easily observe that the event more likely happening after applying the *Disjunctive rule* is the event $\{Grenache, Merlot\}$, since $m_{A,B_D}(\{Grenache, Merlot\}) = 0.9702$.

This means that based on the information given by Ann and Bill, the wine's kind is most likely to be either Grenache or Merlot. That outcome is reasonable as Ann was quite sure that the wine is Merlot and Bill was almost certain that the wine was Grenache. Given the fact that one of their opinion's is reliable (but we cannot know who), it is only correct to assume that the wine's kind is the first choice of either Ann or Bill.

3. MÖBIUS TRANSFORM

Dempster-Shafer theory provides a very good modeling of the data, but the computational cost of Dempster's Combination Rule is significant as the possible intersection computations can reach the amount of $2^{|U|+1}$. Because of this fact, a lot research has been done in order to reduce the complexity of the transformations used in the Combination Rule. The basic two approaches are (1) *powerset-based*, (2) *evidence-based* [8]. In this thesis, of our concern will be the powerset-based approach that introduces algorithms whose implementation is based on the constitution of the powerset of U, 2^U and especially the Möbius Transform family. Of course, it is worth bearing in mind that most of the time approximation methods are used, e.g. Monte Carlo in order to achieve a lower computational cost.

3.1 Lattices, distributive lattices and lattice functions

Let $(P, \leq)$ be a partially ordered set, where $\leq$ is the satisfiable relation. Let S be a subset of P. The greatest element of P, that is less than all the elements of S, is called the infimum of S (noted $\bigwedge S$ or $\perp$), if it is unique. The least element of P, that is greater than all the elements of S, is called the supremum of S(noted $\bigvee S$ or $\top$), if it is unique. If $S = \{x, y\}$, we may represent the supremum with the binary operator $\vee$ such that $x \vee y$ and the infimum with the binary operator $\wedge$ such that $x \wedge y$ [21]. When $(2^\Omega, \subseteq)$ is considered, the infimum operator $\wedge$ is the intersection operator $\cap$, while the supremum operator $\vee$ is the union operator $\cup$. [7]

If any set S such that $S \in P$ and $S \neq \emptyset$ has a supremum, we say that P is an upper semilattice. If any set S such that $S \in P$ and $S \neq \emptyset$ has a infimum, we say that P is a lower semilattice. When P is both upper and lower semilattice, we say that P is a lattice. From now on we will be noting the lattices as L. In $(2^\Omega, \subseteq)$, any $S \in 2^\Omega$ such that $S \neq \emptyset$ has an intersection and a union in 2^Ω. They respectively represent the common elements of the sets in S and the cumulative elements of all the sets in S. For that reason, 2^Ω is a lattice.

Definition: We say that x covers y in a lattice $(L, \leq)$, if $x \neq y$ and $y \leq x$ and there $\nexists\, z \in L \setminus \{x, y\}$, such that $y \leq z \leq x$.

An element $j \in L$ is join-irreducible if it cannot be expressed as a supremum of other elements or, equally, if it covers only one element in L. The set of all join-irreducible elements of L are denoted by J(L). In $(2^\Omega, \subseteq)$, the join-irreducible elements are the singletons $\{\omega\}$, where $\omega \in \Omega$.

If $\quad \forall x, y, z \in L, \quad (z \vee y) \wedge (z \vee x) = z \wedge (y \vee x)$, the L is distributive.
In the powerset lattice 2^Ω, for any sets $\omega_1, \omega_2, \omega_3 \in 2^\Omega$ it is true that $(\omega_1 \cap \omega_2) \cup (\omega_1 \cap \omega_3) = \omega_1 \cap (\omega_2 \cup \omega_3)$. Thus, the lattice 2^Ω is a distributive lattice.

It is suggested that $N := \{1, \ldots, n\}$ is a finite set which can be thought as the set of players, also voters, criteria, states of nature etc. We consider finite distributive lattices $(L_1, \leq_1)$, ... , $(L_n, \leq_n)$ and their product $L := L_1 \times \cdots \times L_n$ endowed with the product order $\leq$ as L is also a distributive lattice. Elements $x \in L$ can be written in their vector form $(x_1 , \ldots , x_n)$. The elements in *J*(L) are of the form $(\perp_1 , \ldots , \perp_{i-1} , j_i , \perp_{i+1} , \ldots ,$ $\perp_n)$, for some i and some join-irreducible element $j_i \in$ *J*(L).

Lattice functions are real-valued mappings defined over product lattices as defined above. Lattice functions which vanish at $\perp$ are called lattice games (or games) on $(L, \leq)$. We denote by *R*^{L} the set of lattice functions over L, and by *G*(L) the subspace of games. Each lattice $(L_i, \leq_i)$ may be different, and represents the (partially) ordered set of actions, choices, levels of participation of player i in the game.

3.2 Boolean lattices

In a lattice $(L, \leq)$ with supremum $\top L = \sup(L) = s$ and infimum $\perp L = \inf(L) = i$, an element $x \in L$ is said to have a complement $x^* \in L$, if $x \vee x^* = i$ and if $x \wedge x^* = s$. For example, if L is a powerset 2^U, then $x^* = x^c = E \backslash x$. L is called complemented, if $\forall x \in L$ exists a complement x^* [23].

A Boolean lattice is defined as any lattice that is complemented and distributive. In a Boolean lattice B, the complement of each element is unique and involutive $x = (x^*)^*$. The mapping $x \rightarrow x^*$ is a negation (i.e., an involutive dual automorphism) on B. Thus, any Boolean lattice is self-dual. A Boolean lattice can be also called a Boolean algebra.

For each $x, y \in B$, the following additional properties hold:

$x \vee x^* = i$ and $x \wedge x^* = s$,

$(x \vee y)^* = x^* \wedge y^*$ and $(x \wedge y)^* = x^* \vee y^*$

The powerset $(2^U, \subseteq)$ containing all subsets of a set U ordered by inclusion is a lattice, as we showcased previously. This lattice is also complemented and distributive and, thus, is a Boolean lattice. It can be proven that the cardinality of a finite Boolean lattice must be of the form 2^N, where $N \geq 1$ is the number of elements.

3.3 Möbius Transform

For any partially ordered set $(P, \leq)$, the Möbius Transform of a mapping $g : P \rightarrow \mathbb{R}$ is f, the unique solution to the equation (that is the Möbius inversion formula) [27]:

$$f(y) = \textstyle\sum_{x \leq y} g(x)\mu(x, y) \text{ , } \forall y \in P$$

where $\mu(x, y)$ is the Möbius function of $(P, \leq)$ and is defined in its recursive form as follows [8] :

$$\forall x, y \in P \quad \mu(x, y) = \begin{cases} 1, & if \quad x = y \\ -\sum_{x<t\leq y} \mu(t, y), & \text{otherwise} \end{cases}$$

It should be noted that the mapping $g : P \to \mathbb{R}$ of a function $f : P \to \mathbb{R}$ is called *zeta transform* and is defined:

$$\forall y \in P, \quad g(y) = \textstyle\sum_{y\leq x} f(y).$$

In other words the Möbius transform undoes the zeta transform it is referring to.

As mentioned in [17], if $(P, \subseteq)$ is a Boolean lattice 2^N, endowed with inclusion, $\mu(A, B) = (-1)^{|B|-|A|}$, $\forall A, B \in P$ such as $A \subseteq B$.

In the next two sections we will summarize the contents of [7] regarding the topic of the sequence of graphs that compute the zeta and Möbius transform.

3.3.1 Sequence of graphs and computation of the zeta transform

The goal is to minimize the number of operations when computing the *zeta transform* $g(y)$ for some $y \in P$, where P is a partially ordered set. The main idea is the following: instead of computing $g(y)$ alone, one can compute $g(y)$, $\forall y \in P$ and reuse the partial sums that appear. It is easy to notice that we can recursively built partial sums in order to get $g(y)$ by summing the values on some elements $\{x \in P \mid x \leq y\}$ as for each $y \in P$ if $\exists\ z \in P$ such that $y \leq z$, then

$$g(z) = g(y) + \textstyle\sum_{\substack{x\leq z \\ x\nleq y}} f(x) \text{ [7]}$$

Basically, what we would like to achieve is to define an ordered sequence of transformations computing g from f.

3.3.1.1 Graph Theory Formalization

We denote as $G_{\leq} = (P, E_{\leq})$ a directed acyclic graph whose nodes match P and every arrow is directed by $\leq$. The set of arrows is referred as $E_{\leq} = \{(x, y) \in P^2 \mid x \leq y\}$.

We, now, present the operation $f(x) + \cdot$, where the dot$(\cdot)$ represents the current state of the sum associate with y. This operation in $G_{\leq}$, and more specifically the $f(\cdot) + \cdot$, describes the transformation of 0 to g. More briefly, we will initialize our algorithm with values through f instead of 0 and exploit the combination of $G_<$ and $+$. From now on, we will say that the transformation $(G<, f, +)$ computes the *zeta transform* of f in $(P, \leq)$. Redefining our initial goal: we want to minimize the total number of arrows to follow in order to compute every $g(y)$ and specifically, we are aiming to find an ordered sequence of graphs that can compute the *zeta transform* with less arrows in total than $(G<, f, +)$.

In order to work towards achieving our goal, we define as $I_P = \{(x, y) \in P^2 \mid x = y\}$ the set containing all identity arrows and $(H_i)_{i\in\{1,\cdots,n\}}$ be a sequence of n directed acyclic graphs $H_i = (P, E_i)$. Let $y \in P$ be initialized with $f(y)$. What we want is to find a graph-sequence $(H_i)_{i\in\{1,\cdots,n\}}$ equal to the arrows of $G_<$. The notation for transforming f to H_1 through the arrows of E_1, then H_1 to H_2 through the arrows of E_2 etc. until H_{n-1} to H_n through the arrows of E_n is $((H_i)_{i\in\{1,\cdots,n\}}, f, +)$. In this computation, all identity arrows are ignored and for that reason even though while computing this sequence we consider a total of $\sum_{i=1}^{n} \mid E_i \mid$ arrows, it transforms f to H_n using $\sum_{i=1}^{n} \mid E_i \setminus I_P \mid$ operations.

Proposition 1. *Let $\Omega = \{\omega_1, \omega_2, \cdots \omega_n\}$ and $(H_i)_{i\in\{1,\cdots,n\}}$ be a sequence of graphs such as $H_i = (L, E_i)$ where L is a Boolean lattice 2^Ω with the relation $\subseteq$ and the set of arrows $E_i = \{(X, Y) \in 2^\Omega \times 2^\Omega \mid Y = X \cup \{\omega_i\}\}$. The sequence H_i computes the same zeta transformation as $G_\subset = (2^\Omega, E_\subset)$, where $E_\subset = \{(X, Y) \in 2^\Omega \times 2^\Omega \mid X \subset Y\}$.*

This preposition can be proved using the Theorem 3 in [17]. It is proven in [7].

For the reader to understand the above more easily we will now provide an example from [7]. The example is illustrated in Figures 3.1 and 3.2:

Example 3.3.1. *Let $\Omega = \{\alpha, \beta, \gamma\}$ we have:*

- $E_1 = \{(X, Y) \in 2^\Omega \times 2^\Omega \mid Y = X \cup \{\alpha\}\} =$

  $\{\ (\emptyset, \{\alpha\}),\ (\{\alpha\}, \{\alpha\}), (\{\beta\}, \{\alpha, \beta\}),\ (\{\alpha, \beta\}, \{\alpha, \beta\}),\ (\{\gamma\}, \{\alpha, \gamma\}),$
  $(\{\alpha, \gamma\}, \{\alpha, \gamma\}), (\{\beta, \gamma\}, \Omega),\ (\Omega, \Omega)\ \} \xRightarrow{\textit{ignoring all identity arrows}}$

  $E_1 = \{\ (\emptyset, \{\alpha\}),\ (\{\beta\}, \{\alpha, \beta\}),\ (\{\gamma\}, \{\alpha, \gamma\}),\ (\{\beta, \gamma\}, \Omega)\ \}$

- $E_2 = \{(X, Y) \in 2^\Omega \times 2^\Omega \mid Y = X \cup \{\beta\}\} =$

  $\{\ (\emptyset, \{\beta\}),\ (\{\alpha\}, \{\alpha, \beta\}),\ (\{\beta\}, \{\beta\}),\ (\{\alpha, \beta\}, \{\alpha, \beta\}),\ (\{\gamma\}, \{\beta, \gamma\}),$
  $(\{\alpha, \gamma\}, \Omega), (\{\beta, \gamma\}, \{\beta, \gamma\}),\ (\Omega, \Omega)\ \} \xRightarrow{\textit{ignoring all identity arrows}}$

  $E_2 = \{\ (\emptyset, \{\beta\}),\ (\{\alpha\}, \{\alpha, \beta\}),\ (\{\gamma\}, \{\beta, \gamma\}),\ (\{\alpha, \gamma\}, \Omega)\ \}$

- $E_3 = \{(X, Y) \in 2^\Omega \times 2^\Omega \mid Y = X \cup \{\gamma\}\} =$

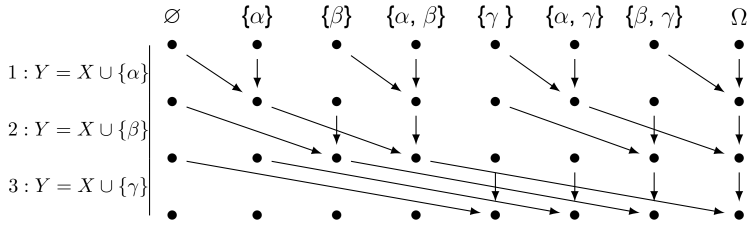

∅
{α}
{β}
{α, β}
{γ}
{α, γ}
{β, γ}
Ω
$1 : Y = X \cup \{\alpha\}$
$2 : Y = X \cup \{\beta\}$
$3 : Y = X \cup \{\gamma\}$

Figure 3.1: Illustration of example 3.3.1.

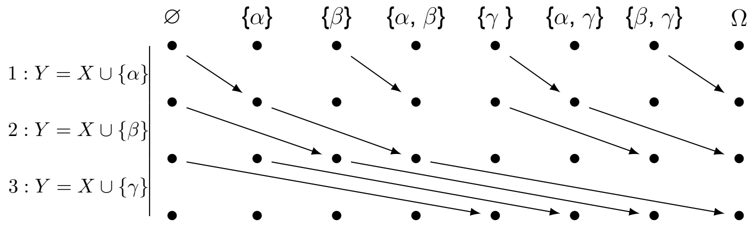

∅
{α}
{β}
{α, β}
{γ}
{α, γ}
{β, γ}
Ω
$1 : Y = X \cup \{\alpha\}$
$2 : Y = X \cup \{\beta\}$
$3 : Y = X \cup \{\gamma\}$

Figure 3.2: Illustration of example 3.3.1, without identity arrows.

$$\{\ (\emptyset,\{\gamma\}),\ (\{\alpha\},\{\alpha,\gamma\}),\ (\{\beta\},\{\beta,\gamma\}),\ (\{\alpha,\beta\},\Omega),\ (\{\gamma\},\{\gamma\}),$$
$$(\{\alpha,\gamma\},\{\alpha,\gamma\}),(\{\beta,\gamma\},\{\beta,\gamma\}),\ (\Omega,\Omega)\ \} \xRightarrow{\textit{ignoring all identity arrows}}$$

$$E_3 = \{\ (\emptyset,\{\gamma\}),\ (\{\alpha\},\{\alpha,\gamma\}),\ (\{\beta\},\{\beta,\gamma\}),\ (\{\alpha,\beta\},\Omega)\ \}$$

In Figure 3.1 are represented the paths generated by the arrows contained in the sequence $(H_i)_{i\in\{1,\cdots,3\}}$, where $H_i = (2^\Omega, E_i)$ and $E_i = \{(X,Y) \in 2^\Omega \times 2^\Omega /\ Y = X \cup \{\omega_i\}\}$ and $\Omega = \{\omega_1,\omega_2,\omega_3\} = \{\alpha,\beta,\gamma\}$. This sequence computes the same zeta transformations as $G_\subset = (2^\Omega, E_\subset)$, where $E_\subset = \{(X,Y) \in 2^\Omega \times 2^\Omega\ /\ X \subset Y\}$. The dot represents the node of its column and each row i is correlated with both the tail of a possible arrow in H_i and the head of a possible arrow in H_{i-1}. All arrows can be viewed as the actual arrows in each graph H_i. The bottom row corresponds to the heads of all potential arrows that could be in H_3.

The illustrations presented in Figures 3.1 and 3.2 were introduced in [18] and display the sequence of graphs.

After executing $((H_i)_{i\in\{1,\cdots,n\}}, f, +)$ each element $y \in 2^\Omega$ is linked with the sum $\sum_{x\subseteq y} f(x)$. In order to understand this statement we will look at Ω in Example 3.3.1, since it is the perfect example to display the whole operation. The operation will be presented step by step below. The reader is, also, advised to study the steps while looking at Figure 3.2. Note that h_i is the function resulting from the transformation $((H_i)_{i\in\{1,\cdots,n\}}, f, +)$ ignoring identity arrows, at each step i.

1. Ω is associated with $f(\Omega)$.

2. The value on Ω is summed with $f(\{\beta,\gamma\})$.
3. The result from the previous step is now summed with $h_1(\{\alpha,\gamma\})$ that equals with $f(\{\gamma\}) + f(\{\alpha,\gamma\})$.
4. The result from the previous step is now summed with $h_2(\{\alpha,\beta\})$ that equals with $h_1(\{\alpha\}) + h_1(\{\alpha,\beta\}) = f(\emptyset) + f(\{\alpha\}) + f(\{\beta\}) + f(\{\alpha,\beta\})$.

After executing all the above steps we have $H_3(\Omega) = f(\emptyset) + f(\{\alpha\}) + f(\{\beta\}) + f(\{\alpha\}) + f(\{\gamma\}) + f(\{\alpha,\gamma\}) + f(\{\beta,\gamma\}) + f(\Omega)$.

This graph sequence's computation complexity is $O(n \cdot 2^n)$ in time and $O(2^n)$ in space. It is fundamental for the *Fast Möbius Transform* (FMT) algorithms.

3.3.2 Sequence of graphs and computation of the Möbius transform

At this point we would like to transform g into f, i.e. the Möbius transform of g in $(P, \leq)$. This means that we would like to undo the previous computation. In order to achieve this, check that for any step i in the transformation $((H_i)_{i\in\{1,\cdots,n\}}, f, +)$, we have $\forall y \in P$,

$$h_i(y) = h_{i-1}(y) + \sum_{(x,y)\in E_i \backslash I_P} h_{i-1}(x) \quad \Leftrightarrow \quad h_{i-1}(y) = h_i(y) - \sum_{(x,y)\in E_i \backslash I_P} h_{i-1}(x).$$

This means that, as long as we know all $H_{i-1}(x), \quad \forall (x,y) \in E_i \backslash I_P$ at every step i and $\forall y \in P$, or equally that for every arrow $(x,y) \in E_i \backslash I_P$, there is no arrow (w,x) in $E_i \backslash I_P$, we can simply reverse the order of the sequence $(H_i)_{i\in\{1,\cdots,n\}}$ and change the operator $+$ to $-$. Then, we can state that $((H_{n-i+1})_{i\in\{1,\cdots,n\}}, g, -)$ computes the Möbius transform of g in $(P, \leq)$.

Theorem 3.3.1. *Let $(H_i)_{i\in\{1,\cdots,n\}}$ be a sequence of directed acyclic graphs $H_i = (P, E_i)$ and h_n be the function resulting from the transformation $((H_i)_{i\in\{1,\cdots,n\}}, f, +)$, ignoring identity arrows. If for every arrow $(x,y) \in E_i \backslash I_P \quad \nexists (w,x)$ in $E_i \backslash I_P$, then $((H_{n-i+1})_{i\in\{1,\cdots,n\}}, h_n, -)$ yields the initial function f.*

So, if $((H_i)_{i\in\{1,\cdots,n\}}, f, +)$ computes the zeta transform g of f in $(P, \leq)$ and Theorem 3.3.1 is met, then $((H_{n-i+1})_{i\in\{1,\cdots,n\}}, g, -)$ *computes the Möbius transform f of g in $(P, \leq)$.*

3.3.2.1 Fast Möbius Transform - Application to the powerset lattice 2^Ω

Let's remember the sequence $(H_i)_{i\in\{1,\cdots,n\}}$, from the application of section 3.3.1 that is equivalent to the computation of the zeta transform of f in $(2^\Omega, \subseteq)$. Since, if $\exists (X,Y) \in E_i \backslash I_{2^\Omega}$, then $\omega_i \notin X \Rightarrow \nexists W$ in 2^Ω such that $W \cup \{\omega_i\} = X$ and, for that reason, $\nexists (W,X)$ in $E_i \backslash I_{2^\Omega}$. One can notice that the Theorem 3.3.1 is applicable and so $((H_{n-i+1})_{i\in\{1,\cdots,n\}}, h_n, -)$

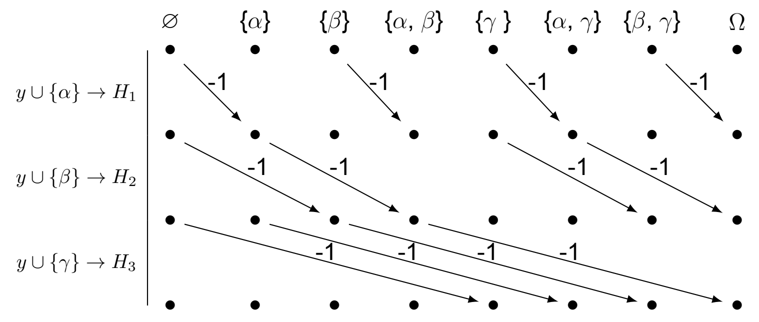


Figure 3.3: Illustration similar to Figure 3.2, but altered to showcase the Möbius Transform.

computes the Möbius transformation of $((H_i)_{i\in\{1,\cdots,n\}}, f, +)$, i.e. the function f. Also, any order in $((H_{n-i+1})_{i\in\{1,\cdots,n\}}, h_n, -)$ computes the Möbius transformation of $((H_i)_{i\in\{1,\cdots,n\}}, f, +)$ because Ω is a set and each graph H_i concerns an element ω_i, independently from the others, any order in the sequence $(H_i)_{i\in\{1,\cdots,n\}}$ computes the zeta transformation.

Now, we can apply this knowledge to the Example 3.3.1 and compute the Möbius transform f of g in $(2^\Omega,\subseteq)$, given that $\Omega=\{\alpha,\beta,\gamma\}$. At this point, the reader is advised to look at the Figure 3.3 in order to understand the following with ease. Note that, after executing $((H_i)_{i\in\{1,\cdots,n\}}, h_n, -)$, every element $y\in 2^\Omega$ is mapped with $f(y)$. Again, we will be looking at Ω, since it is the perfect example to display the whole operation in steps:

1. Ω is associated with $g(\Omega)$.
2. From the value on Ω we subtract $g(\{\beta,\gamma\})$.
3. From the result of the previous step's subtraction $h_1(\{\alpha,\gamma\})$ is now subtracted and that equals with $g(\{\alpha,\gamma\})-g(\{\gamma\})$.
4. From the result of the previous step's subtraction $h_2(\{\alpha,\beta\})$ is now subtracted and that equals with $h_1(\{\alpha,\beta\})-h_1(\{\alpha\})=g(\{\alpha,\beta\})-g(\{\beta\})-(g(\{\alpha\})-g(\emptyset))$.

After executing all the above steps we have

$$\begin{aligned}h_3(\Omega)&=g(\Omega)-g(\{\beta,\gamma\})-[\,g(\{\alpha,\gamma\})-g(\{\gamma\})\,]-[\,g(\{\alpha,\beta\})-g(\{\beta\})-[\,g(\{\alpha\})-g(\emptyset)\,]\,]\\&=g(\Omega)-g(\{\beta,\gamma\})-g(\{\alpha,\gamma\})+g(\{\gamma\})-g(\{\alpha,\beta\})+g(\{\beta\})+g(\{\alpha\})-g(\emptyset)\\&=\sum_{X\subseteq\Omega}g(x).(-1)^{|\Omega|-|X|}.\end{aligned}$$

We stated earlier that μ in $(2^\Omega,\subseteq)=(-1)^{|Y|-|X|}$ for every couple $(X,Y)\in 2^\Omega\times 2^\Omega$ Thus, we have $h_3(\Omega)=f(\Omega)$.

3.4 Möbius Transform in Dempster-Shafer theory

The Möbius Transform defines a mapping between two categories. In our case we are interested in a mapping between bel and m functions as well as pl and m [18, 17]. Taking into consideration the definitions presented in section 3.3 and it's subsections, one can present these the Möbius transform formulas.

If g is equated to the belief function $bel(A) = \sum_{B \subseteq A} m(B),\ \forall A \in 2^U$ and, thus, in this context $g = bel$ and $f = m$, then the Möbius Transform of bel is:

$$m(A) = \sum_{B \subseteq A} bel(B) \cdot (-1)^{|A|-|B|},\ \forall A \in 2^U.$$

Due to the relation between bel and pl functions $bel(A) = 1 - pl(\overline{A})$, the Möbius Transform of pl is:

$$m(A) = \sum_{B \subseteq A} pl(\overline{B}) \cdot (-1)^{|A|-|B|+1},\ \forall A \in 2^U.$$

4. IMPLEMENTATIONS OF DEMPSTER-SHAFER THEORY TO COMPARE IN THE THESIS

In this section we will study and comment on two different implementations of DST. Specifically, we are going to present one implementation based on CLP and one based on *Fast Möbius Transforms*. Bear in mind, that in later sections we are going to use these implementations, so that is why we will focus on explaining the code that frames them.

4.1 Constraint Logic Programming Implementation

This implementation was developed by Alexandros N. Kaltsounidis in his BSc Thesis [16]. Kaltsounidis focused on the optimization of Dempster's rule of combination, by using CLP. He implemented such a method in *ECLiPSe Prolog* and tested and compared it against a method utilizing Logic Programming, but not CLP, computing the combined mass function and the belief of random sets, of randomly generated test cases. Overall, his implementation offered reduced run-time in all computations.

4.1.1 Constraint Logic Programming and Prolog

Logic Programming is a formalism suitable for programming and representing knowledge [22] and is based on first order logic. With the help of a logic programming language, one can write logic programs. A logic program is a relation definition and is a sequence of sentences in logical form, each of which expresses a fact or a rule for a given domain. One example of such a programming language is *Prolog* [9, 28], a declarative language, where rules and facts are defined in the form of Horn clauses.

A constraint satisfaction problem (CSP) [20] is described by set of variables, where every variable has a finite and discrete domain, and a set of constraints. Every constraint is defined over some subset of the original set of variables and limits the combinations of values that the variables in this subset can take. The goal is to find one or more assignments to the variables such that the assignment satisfies all the constraints. Constraint programming (CP) is a general framework for modeling and solving CSP's. [24]

Constraint Programming can be integrated into a logic programming language, where it is referred to as Constraint Logic Programming (CLP) [14]. Dempster's rule and the belief function was implemented in *ECLiPSe Prolog* [2], a software system implementing *Prolog* that also offers libraries for Constraint Programming. The implementation uses *ECLiPSe*'s `ic` library that supports finite domain constraints and the `ic_sets` library which implements constraints over the domain of finite sets of integers and cooperates with `ic`. These are libraries that are used in Kaltsounidis' implementation.

4.1.2 Predicates

Since, later on, we are going to be referring to and using predicates from [16], these predicates should me mentioned and described.

The first predicate we will present is the one responsible for combining the mass functions and is no other that Dempster's rule, `bpa/2`. This predicate is defined as `bpa( -Tars, -NewVals)` and produces the new focal points `Tars` and their corresponding combined values `NewVals`.

The second one will, of course, be the `belief/2` predicate, or more specifically `belief(+A, -V)`, that computes belief `V` for set `A`. Note that in order to find `V`, it is not required to call `bpa/2` prior to calling the belief predicate. This happens, because both predicates `belief/2` and `bpa/2` use the predicate `findall( _, compute(_, _, _), _)`, where `compute(-A,-Val,+Hyper )` finds a combination of sets whose intersection is A and computes its value.

Similarly to the belief predicate, is defined the plausibility one `plau/2`, which is `plau(+A, -FinalV)` that finds the plausibility `FinalV` for set `A`, using the `belief/2`.

One last useful predicate is the predicate `compute_K/1`, `compute_K(K)`, that computes the normalization normalization factor K.

For more detailed descriptions of the above predicates as well as some examples, please refer to [16].

4.2 Fast Möbius Transforms Implementation

In previous sections the computation of Möbius Transforms and FMT's, as well as the relation between DST and Möbius Transforms was thoroughly explained. Theoretically this way of computing belief and plausibility is efficient and that is the reason why in next sections we are going to use an implementation of belief provided by CRAN that utilizes FMT's.

4.2.1 Package ibelief

The package *ibelief* [1] from the *Comprehensive R Archive Network (CRAN)* provides some basic implementations of belief functions. It includes transformation between belief functions using the method introduced in [18]. It also includes evidence combination, evidence discounting, decision-making, constructing and randomly generating mass functions. In the current version, *ibelief 1.3.1*, are supported thirteen(13) combination rules and six(6) decision rules. The package includes an implementation of Dempster's rule as well as FMT's that compute belief from mass functions. We will use this package indirectly through *Python* with the help of the package *rpy2* [4] that creates an interface between *Python* and *R*.

4.2.1.1 ibelief modules

Again, as later on we will utilize functions from this package, here we will present these functions and some of their characteristics.

First of all, to measure the performance of the combination rule and the belief function, mass functions should be created. The library provides an implementation to randomly create mass functions with some requirements. The function is called `RandomMass( nbFocalElement, ThetaSize, nbMass, Type, singleton, Include)`. It's arguments represent the following:

- `nbFocalElement`, number of focal elements per generated mass function.
- `ThetaSize`, length of the discernment frame Θ.
- `nbMass`, number of masses to generate.
- `Type`, which kind of mass to generate. For example, `Type=1` for focal elements can be everywhere. For more options refer to page 9 of 'ibelief' package reference.
- `singleton`, the singleton element (with only one element) in the focal sets. It should be given a number from 1 to ThetaSize if `Type` is from 5 to 11.
- `Include`, the natural id of the focal element (not Θ) of SSFs, if `Type` is 15.

The return value of the function is the generated mass matrix, where each column represents a piece of mass. Each column has $2^{|Theta|}$ rows where the bpa for every subset of the powerset is stored.

After creating the mass functions, the computation of the combined mass function is next. The package provides different combination rules through the function `DST(MassIn , criterion, TypeSSF = 0)` and the arguments are the following:

- `MassIn`, matrix containing the masses. Again, each column has $2^{|Theta|}$ rows and represents a piece of mass for every subset of the powerset.
- `criterion`, combination criterion. We are interested in `criterion=2`, that is the Dempster-Shafer normalized criterion. For more options refer to page 5 of 'ibelief' package reference.
- `TypeSSF`, the parameter of LNS and LNSa rule.

The function returns the combined mass vector that has dimensions $2^{|Theta|} \times 1$.

Since the combination process has been explained, the following step is to present the FMT functions provided by *ibelief*. The package provides a handful of them (see page 6 of 'ibelief' package reference), but the one we used was `mtobel(InputVec)` in order to compute the belief from a combined mass. The argument `InputVec` is the measure to transform (here: the combined mass) and should be of size $2^{|Theta|} \times 1$. The function returns the associated converted new measure. Here that is a matrix $1 \times 2^{|Theta|}$, which holds the belief for every subset $s \in 2^{|Theta|}$. In this module exists also the FMT `mtopl(InputVec)` that computes the plausibility from the combined mass. We did not use that, as we can compute plausibility for a set by computing the belief of its complement.

Below is provided a code example of how the functions are used.

```
# import for R - ibelief
import rpy2.robjects.packages as rpackages

# import R package needed
ibelief = rpackages.importr('ibelief')

# create masses
Mass=ibelief.RandomMass(nbFocalElement=5, ThetaSize=3, nbMass=4, Type=1)
print("Printing masses:")
print(Mass)

# combine masses
MassCombined=ibelief.DST(Mass,2)
print("Printing combined mass:")
print(MassCombined)

# compute belief
Bel = ibelief.mtobel(MassCombined)
print("Printing belief:")
print(Bel)
```

Output:

```
Printing masses:
           [,1]       [,2]       [,3]       [,4]
[1,] 0.00000000 0.00000000 0.02336222 0.15781926
[2,] 0.10842573 0.44519333 0.00000000 0.00000000
[3,] 0.00000000 0.00000000 0.14233238 0.00000000
[4,] 0.09396573 0.11848799 0.41035060 0.25968586
[5,] 0.00000000 0.01033163 0.00000000 0.02614128
[6,] 0.13453683 0.00000000 0.09491176 0.20112460
[7,] 0.52519882 0.08784273 0.32904304 0.35522901
[8,] 0.13787288 0.33814432 0.00000000 0.00000000

Printing combined mass:
            [,1]
[1,] 0.000000000
[2,] 0.238677809
[3,] 0.493537463
[4,] 0.031072186
```

```
[5,] 0.140933796
[6,] 0.004843127
[7,] 0.090935618
[8,] 0.000000000

Printing belief:
     [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7] [,8]
[1,]    0 0.2386778 0.4935375 0.7632875 0.1409338 0.3844547 0.7254069    1
```

5. AN APPLICATION OF DEMPSTER-SHAFER THEORY IN RECOMMENDER SYSTEMS

After studying DST and its properties as well as seeing some rather limited examples of the theory's application to real-life situations it is only normal to want to see how practical and applicable DST is in extended real-life situations. One particularly interesting application is creating a model for discovering user preferences from item characteristics [33] that can find usage when implementing a Recommender System.

5.1 Recommender Systems

Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user [26]. A RS uses data provided to it that can be a direct representation of the user's liking, for example products that the user has 'liked' or rated, or an indirect representation of the user's liking, for example a set of characteristics of a product that the user has purchased. There are many ways a RS can make suggestions based on the way it processes the data. As mentioned in [6], based on the filtering approach, the RS algorithm can be marked as:

1. *Content-based recommenders*, that make recommendations to a user based on the user's preferences in the past.
2. *Collaborative recommenders*, that make recommendations to a user based on the preferences of users with similar activity.
3. *Demographic recommenders*, that make recommendations to a user based on the preferences of users that belong to the same group based on personal attributes (age, sex, socioeconomic situation etc.).
4. *Hybrid recommenders*, that make recommendations to a user combining the above recommenders in some way.

5.2 Recommender System based on Dempster-Shafer theory

In this section we will present the RS model of [33]. Note that this is a model that:

1. Assumes that all items are described by the same characteristics. These characteristics can have single or multiple values.
2. Assumes that the number of the focal points is significantly larger that the number of items.

3. Assumes that items can be grouped and user preferences can be expressed on item groups. The values assigned to groups of items are obtained by the union of characteristic values assigned to each item.
4. Does not consider rating or negative feedback.
5. Assumes that if the user prefers items with specific characteristic-values they will, most likely, in the future prefer items with the same features.
6. Considers users to be homogeneous in terms of profiling characteristics.

5.2.1 Some definitions

In this approach, the preferences are deducted by 'like' votes. Each item is described by its characteristics, since they are our main focus. For that reason, we describe the item set with the feature set $C = \{C_1, C_2, \ldots, C_p\}$, where C_i with $i \in \{1, 2, \ldots, p\}$ is a discrete variable with domain $\Phi_i = \{\phi_{i,1}, \ldots, \phi_{i,k_i}\}$. The integer k_i describes the number of the different values the feature described can take. The feature domain is described as $\Phi = \{\Phi_1, \Phi_2, \ldots, \Phi_p\}$. Now, $I = \{I_1, I_2, \ldots, I_m\}$ we denote as the item set in our model. Every item I_j is described by the tuple - vector $(c_{1,j}, c_{2,j}, \ldots, c_{p,j})$, where $c_{i,j} \in 2^{\Phi_i}$. Last but not least, the set of users is denoted as $U = \{U_1, U_2, \ldots, U_n\}$.

Since the preferences are deducted by 'like' votes, a user liking an item I_j means that they like all the characteristics that describe it : $(c_{1,j}, c_{2,j}, \ldots, c_{p,j})$. As we believe that a user has a repeated behaviour and likes a certain and restrained amount of features in our system, by taking into account the features and not the items we are indeed reducing our search space.

5.2.2 Basic probability assignment for a feature set

As the model assumes that the users are homogeneous, the search can be limited to the votes expressed be the users that as homogeneous to the querying user.

Definition. *Feature basic probability assignment*: Let C_i with $i \in \{1, 2, \ldots, p\}$ be a discrete variable with domain $\Phi_i = \{\phi_{i,1}, \ldots, \phi_{i,k_i}\}$. The basic probability assignment $m(K)$, $\forall K \in 2^{\Phi_i}$ is defined as follows,

$$m_i(K) = \frac{\#U_K}{\#U}$$

with $U_K \subseteq U$ being the users whose selected products with features $c_{i,j}$ are part of K, i.e. $\bigcup_{j \in I(U_K)} c_{i,j} = K$ and $I(U_K)$ being the collection of items chosen by U_k.

Usually, for a feature Φ_i, the k_i is not big. Even with small numbers, though, the powerset of the feature domain would be significantly large and it would be hazardous to compute.

But since the bpa depends on the users as defined above, the number of property sets with a bpa larger that 0 is at most the n, where n is the number of users.

After computing the mass function for each feature, they cannot be combined using Dempster's combination rule as the sets probably won't intersect, since they belong to different features with different domains. The solution to that, is to project the bpa of each feature to the item set i.e. compute a bpa for items with regard to a feature. We introduce the item set as a shared domain on which to combine user-preferences expressed on each feature. Specifically, in the article the below definitions explain the process:

Definition. *Item projection*: Let C_i with $i \in \{1, 2, \ldots, p\}$ be a feature with domain Φ_i. Then $\forall\ \Psi = \{I_1, I_2, \ldots, I_s\} \in 2^I : p_{C_i}(\Psi) = K_j \subseteq 2^{\Phi_i}$, with $\phi_{i,q} \in K_j$ iff $\phi_{i,q} \subseteq c_{i,m}$ such that $I_m = (c_{1,m}, c_{2,m}, \ldots, c_{p,m}), \quad \forall m \in \{1, 2, \ldots, s\}$ and $q \in \{1, 2, \ldots, k_i\}$.

Definition. *Item bpa*: Let $\Psi = \{I_1, I_2, \ldots, I_s\} \in 2^I$ be an item set and $p_{C_i}(\Psi) = K_j$ the item set's projected feature set with regard to the feature C_i. The mass function of the projected feature set is : $m(\Psi)$ = $m_i(p_{C_i}(\Psi))$ = $m_i(K_j)$.

With the above definitions it is possible for us to match, for each characteristic, the basic probability assignment for its domain to a bpa whose domain is the item powerset. After that matching, the new mass functions can be combined using Dempster's rule in order to get a mass joint that represents the dataset.

In the following section we will apply the described model on a chosen dataset.

5.3 The Thesis System

Provided that we analyzed the theory behind the application, the next stage is to develop a system based on the approach described in the section 5.2. The system is provided an input-dataset. The system's goal is firstly to process it appropriately and then do a series of procedures in order to create the output, that is the projected mass functions.

The process that undergoes the dataset in the system in order to produce the output can be seen in the Data Flow Diagram (DFD) 5.1. If you are not familiar with this type of diagram, the DFD [36] maps out the flow of information for any process or system. The types of nodes in this diagram are three:

1. an external entity that is an outside system that sends or receives data and is defined with a simple rectangle
2. a process that is any process that changes the data, producing an output and is illustrated with a rectangle with a horizontal line

3. a data store that are files or repositories that hold information for later use and are presented by rectangles with a vertical line

The edges showcase data transferred from one entity to another.

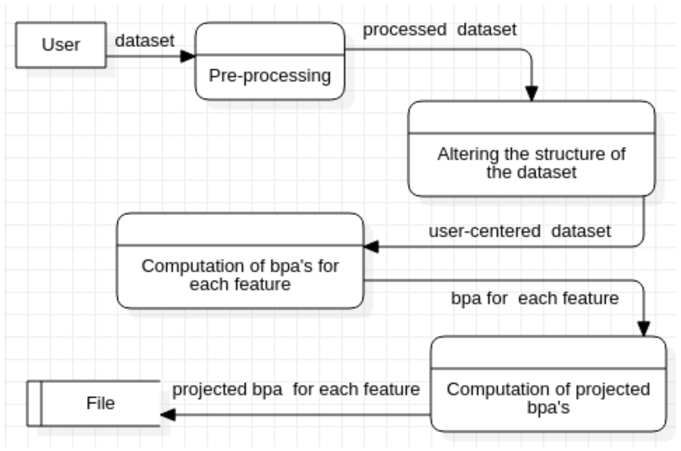

User
dataset
Pre-processing
processed dataset
Altering the structure of the dataset
user-centered dataset
Computation of bpa's for each feature
bpa for each feature
Computation of projected bpa's
projected bpa for each feature
File

Figure 5.1: Data Flow Diagram of DST Application

Let us explain the diagram 5.1. The User, who is an external entity, provides the process `Pre-processing` with the data (here: the dataset). The process processes the dataset (erases the information that is not needed and "polishes" the rest) and the processed dataset is supplied to the process `Altering the structure of the dataset`, where it changes its form and becomes user-centered. Afterwards, the user-centered dataset is used by the processes `Computation of bpa's for each feature` and `Computation of projected bpa's` which produce the mass functions for each feature and, then, the projected mass functions respectively. These projected mass functions are stored in a data store with the purpose to be used later on. The processes and the data introduced in the diagram are going to be thouroughly presented in the following sections.

5.3.1 Dataset of application

In order to apply DST as suggested in [33], a dataset was needed. The dataset had to have items with characteristics and the users that preferred them, so the search was quite explicit. After spending some time searching and researching a dataset was found in Kaggle: netflix-audience-behaviour-uk-movies.

5.3.2 Description

The dataset found in the above link includes users in the UK who opted-in to have their anonymous browsing activity tracked and the movies they watched on Netflix from their desktop and/or laptop. The features of the movies that are:

- *row ID*
- *datetime*, date and time of the click
- *duration*, seconds between this click and the user's next tracked click on Netflix
- *title*, movie title
- *genres*, movie genre(s)
- *release_date*, movie's original release date in movie theaters
- *movie_id* title ID
- *user_id* user ID

As one can see in the description of the dataset, the record number is 671736. Each record is described by the above features. Also, there are 7925 unique values in the *title's domain*, 8472 in the *movie_id's* domain and 161918 unique values in the *user_id's* domain. Some movies (movie_id's) do not have genre(s) and/or release_date and when that happens those feature are marked with the string $'\mathrm{NOT\ AVAILABLE}'$.

5.3.3 Processing

In order for us to use the dataset, a few adjustments are needed. For the procedure of processing we will use *Python* and specifically the *Pandas* library [3] . This library provides a visualisation (the *DataFrame* type) of the dataset, as well as a handful of operations on the dataset.

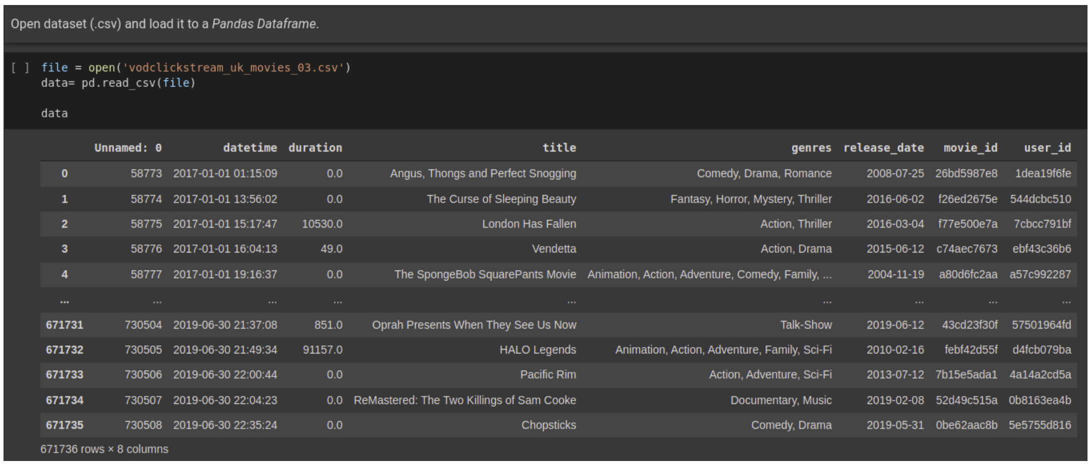
5.3.3.1 Altering size of the dataset

Since the dataset is in *.csv* format, it can easily be loaded to a *DataFrame* structure that is implemented in *Pandas*, as shown in Figure 5.2:

As mentioned before, in *genres* and *release date* if there is no value in those columns regarding a row, then that cell holds the value $'\mathrm{NOT\ AVAILABLE}'$. Those rows are dropped, because (as quoted in [33]'s 7^{th} page) *"In this model it is assumed that only items sharing all the characteristics are provided"*. Below is provided the code that allows us to take such action.

```
data = data.drop(data[data.genres == 'NOT AVAILABLE'].index)
data = data.drop(data[data.release_date == 'NOT AVAILABLE'].index)
```

After that, the remaining dataset consists of 639842 records. This is a huge dataset, therefore fractions of it will be studied. To be precise, we will use a script in order use the


Open dataset (.csv) and load it to a *Pandas Dataframe.*

```
[ ] file = open('vodclickstream_uk_movies_03.csv')
    data= pd.read_csv(file)

    data
```

	Unnamed: 0	datetime	duration	title	genres	release_date	movie_id	user_id
0	58773	2017-01-01 01:15:09	0.0	Angus, Thongs and Perfect Snogging	Comedy, Drama, Romance	2008-07-25	26bd5987e8	1dea19f6fe
1	58774	2017-01-01 13:56:02	0.0	The Curse of Sleeping Beauty	Fantasy, Horror, Mystery, Thriller	2016-06-02	f26ed2675e	544dcbc510
2	58775	2017-01-01 15:17:47	10530.0	London Has Fallen	Action, Thriller	2016-03-04	f77e500e7a	7cbcc791bf
3	58776	2017-01-01 16:04:13	49.0	Vendetta	Action, Drama	2015-06-12	c74aec7673	ebf43c36b6
4	58777	2017-01-01 19:16:37	0.0	The SpongeBob SquarePants Movie	Animation, Action, Adventure, Comedy, Family, ...	2004-11-19	a80d6fc2aa	a57c992287
...	...	...	...	...	...	...	...	...
671731	730504	2019-06-30 21:37:08	851.0	Oprah Presents When They See Us Now	Talk-Show	2019-06-12	43cd23f30f	57501964fd
671732	730505	2019-06-30 21:49:34	91157.0	HALO Legends	Animation, Action, Adventure, Family, Sci-Fi	2010-02-16	febf42d55f	d4fcb079ba
671733	730506	2019-06-30 22:00:44	0.0	Pacific Rim	Action, Adventure, Sci-Fi	2013-07-12	7b15e5ada1	4a14a2cd5a
671734	730507	2019-06-30 22:04:23	0.0	ReMastered: The Two Killings of Sam Cooke	Documentary, Music	2019-02-08	52d49c515a	0b8163ea4b
671735	730508	2019-06-30 22:35:24	0.0	Chopsticks	Comedy, Drama	2019-05-31	0be62aac8b	5e5755d816

671736 rows × 8 columns

Figure 5.2: Image of dataset after loading it to a pandas DataFrame

following fractions: 0.002%, 0.004%, 0.006%, 0.008%, 0.01%. Larger fractions crush the program with *'global_trail_overflow'* error.

```
data = data.sample(frac = sys.argv[1])
```

Also, the features *duration* and *row ID* do not give us any indication about the user's preferences, so we will not be considering them.

```
data = data.drop(['duration'], axis=1)
data = data.drop(data.columns[0], axis=1)
```

The feature $release\_date$ seems quite specific, so at first we thought about using release year as a preference. The problem with that is that $|\Phi_{release\_year}|$ is large and, actually, it is quite strange for one to watch movies released only in one specific year, but one can prefer watching movies that were released a certain decade. So we will use as feature the release decade ($release\_decade$).

```
release_decade = []

for row in data['release_date']:
    # split release date and use only release year
    date_split_list = row.split("-")
    # change last digit (ex. 1992 -> 1990)
    list_of_char = list(date_split_list[0])
    list_of_char[3] = '0'
    date_split_list[0] = ''.join(list_of_char)
    # store release decade
    release_decade.append(date_split_list[0])

data['release_decade'] = release_decade
# dropping release date, because now we have release decade!
data = data.drop(['release_date'], axis=1)
```

The feature $datetime$ seems quite specific as well, so we thought of using click hour ($click\ hour$) as a preference. But, again, because one is more likely to watch a movie within a range of hours, we will make 5 categories for the $click\ hour$: Morning (06-11), Midday (12-16), Afternoon(17-20), Evening(21-00), Night(01-05):

```
click_hour = []

for row in data['datetime']:
    # split date and time
    date_split_list = row.split(" ")
    # split time
    date_split_list = date_split_list[1].split(":")

    # use only hour
    h = date_split_list[0]
    if (h=='06') or (h=='07') or (h=='08') or (h=='09') or (h=='10')
    or (h=='11') :
        click_hour.append('Morning')
    elif (h=='12') or (h=='13') or (h=='14') or (h=='15')
    or (h=='16'):
        click_hour.append('Midday')
    elif (h=='17') or (h=='18') or (h=='19') or (h=='20'):
        click_hour.append('Afternoon')
    elif (h=='21') or (h=='22') or (h=='23') or (h=='00'):
        click_hour.append('Evening')
    else:
        click_hour.append('Night')

data['click hour'] = click_hour
 # dropping date_time, because now we have click hour!
data = data.drop(['datetime'], axis=1)
```

Continuing the processing of the dataset we thought of computing $|\Phi_G|$ regarding c_G where c_G is the feature genre.

```
genre_values = []

for row in data['genres']:
    split_list = row.split(", ")
    genre_values += split_list

genre_values= unique(list(set(genre_values)))
print(genre_values)
print(len(genre_values))
```

When the above code was executed, we noticed that $|\Phi_G| \backsimeq 25$ (The number is not fixed as it depends on the fraction of the dataset after taken). In order to see the approach of [33] in action, such big domain was no use to us and, thus, we decided to remove from the dataset all movies that correspond to at least one of the following genres: *Documentary,* *History, War, Talk-Show, Biography, Musical, Music, Sport*.

```
movies_to_be_dropped = []

for index, row in data.iterrows():
  split_list = row['genres'].split(", ")
  if ('Documentary' in split_list) or ('History' in split_list) or
     ('War' in split_list) or ('Talk-Show' in split_list) or
     ('Biography' in split_list) or ('Musical' in split_list) or
     ('Music' in split_list) or ('Sport' in split_list) :
        movies_to_be_dropped.append(row['movie_id'])

for movie in movies_to_be_dropped:
    data = data.drop(data[data.movie_id == movie].index)
```

Lastly, we observed that there are some duplicate rows in the dataset, so we drop them, as they are no use to our modeling.

```
data = data.drop_duplicates()
data = data.reset_index()  # make sure indexes pair with num of rows
data = data.drop(['index'], axis=1)
```

5.3.3.2 Altering the structure of the dataset

In this paragraph, our goal is to bring the dataset into a form that will allow us to compute the mass functions of each feature easily. Gladly, in [33] in *Table 1: "Structure of dataset assumed by the model"* one such form can be viewed by the reader. That structure is the one we will use, with just a few mild alterations. Specifically, we are going to create a new *Pandas DataFrame* with regards to the user's preferences. The result will be a *DataFrame* that for every user holds the films they have liked, the genres they have liked, the range of hours they prefer to watch films and the decades their preferred movies were released.

```
# create new dataframe
d = {'user_id': [] }
data_per_user_ = pd.DataFrame(data=d)

# first column will be the users (1 user per row and no duplicates)
data_per_user_['user_id'] = (data['user_id'].unique()).tolist()
empty_sets = []

# create different lists of empty sets
for i in range(5):
  empty_sets.append([set() for x in range(len(data_per_user_))])

# make those sets part of the dataframe column-wise
data_per_user_['title'] = empty_sets[0]
data_per_user_['genres'] = empty_sets[1]
data_per_user_['movie_id'] = empty_sets[2]
data_per_user_['release_decade'] = empty_sets[3]
data_per_user_['click hour'] = empty_sets[4]
```

```python
# for each user
for index, row in data_per_user_.iterrows():

    # make a dataframe with the preferences of this particular user
    df_temp = data.loc[data['user_id'] == row['user_id']]

    '''
    now for every preference in the temporary dataframe (for every
    film and its other characteristics) collect them together to
    make only one row of the user preferences in the new user-
    oriented dataset. Note that with sets being used duplicates
    are avoided.
    '''

    for movie in df_temp['movie_id'].unique().tolist():
        row['movie_id'].add(movie)

    for title in df_temp['title'].unique().tolist():
        row['title'].add(title)

    for genres in df_temp['genres'].unique().tolist():
        split_genres = genres.split(", ")
        for genre in split_genres:
            row['genres'].add(genre)

    for release_decade in (df_temp['release_decade'].unique()).tolist():
        row['release_decade'].add(release_decade)

    for hour in df_temp['click hour'].unique().tolist():
        row['click hour'].add(hour)
```

Now in this new *DataFrame*, what we want to do is drop the rows that have only one preferable movie, because we would like to see how DS theory behaves with multiple user preferences.

```python
# compute number of liked movies for each row(user) and store in new column
data_per_user_['length'] = data_per_user_.movie_id.str.len()
# drop rows that have cell value equal to 1
data_per_user_ = data_per_user_.drop(data_per_user_[data_per_user_.length ==
    1].index)
# make sure indexes pair with number of rows
data_per_user_ = data_per_user_.reset_index()
data_per_user_ = data_per_user_.drop(['index'], axis=1)
# drop length column, as we no longer need it
data_per_user_ = data_per_user_.drop(['length'], axis=1)
```

5.3.4 Computation of the basic probability assignment for each feature set

The first step in order to compute the mass function for an item (a movie) is to compute the mass functions of each feature as showcased in section 5.2.2. There will not be any computation of m_{title}, m_{movie_id} as these are not characteristics but two different ways for us to refer to the item-movie . So, we want to compute the bpa, which is: $m(A)$, $\forall A \in 2^{\Phi_i}$ for every feature of the item-movie considered. The first approach one can think is to compute all the subsets of the feature domain and then compute the mass function for every subset. This process is computationally heavy as the complexity of computing the subsets is exponential. But, because the bpa depends on the users and the maximum number of focal points equals with the number of users, the computation of all subsets is not needed. So, the computation of mass function becomes a task with polynomial complexity. The process of the computation for each characteristic will be provided later on accompanied by code.

Firstly, the computation of the total number of users (#U) is necessary.

```
number_of_users = len(data_per_user_['user_id'])
```

Also, we need to introduce a function that does bit-wise rounding of the numbers, as we want to avoid an arithmetic malfunction and specifically an overflow, because then the values of each mass function will sum to >1, which is a problem for the *Prolog* implementation we will use later.

```
# function to round float to the nearest allowable value
def roundf(x,bitsToRound):
    i = cast(pointer(c_float(x)), POINTER(c_int32)).contents.value
    bits = bin(i)
    bits = bits[:-bitsToRound] + "0"*bitsToRound
    i = int(bits,2)
    y = cast(pointer(c_int32(i)), POINTER(c_float)).contents.value
    return y
```

For the feature $genres$ compute #U_k from the user centered dataset:

```
 # create new dataframe to store bpa-related values
d = {'genres': [], 'count' : [], 'bpa' : [], 'movies_proj' : [], 'sets_indexed
    ' : []  }
df_bpa_genres_ = pd.DataFrame(data=d)

# for each user
for index, row in data_per_user_.iterrows():
    # only focal points - from users only!
    if (df_bpa_genres_['genres'] == set(row['genres'])).any() == False:
        df_bpa_genres_.loc[df_bpa_genres_.shape[0]] =
        [row['genres'], 1, 0, set(), set()]
    else:
        df_bpa_genres_.loc[df_bpa_genres_['genres'] ==
        set(row['genres']), 'count'] +=1
```

```
# use formula defined earlier (#Uk/#U)
for index, row in df_bpa_genres_.iterrows():
    # round bt by 10 digits binary
    df_bpa_genres_.loc[index, 'bpa'] = roundf((row['count'])/number_of_users
    ,10)

# sum bpa's - must be ~1 (may not be exactly due to python arithmetic)
sum_g = np.sum(df_bpa_release_decades_['bpa'].tolist())
```

Note that it is normal for the summation of bpa's not to be exactly 1 as the process' computations might cause number underflow or overflow.

For the other two characteristics, the computation is similar.

5.3.5 Projection of the basic probability assignment of each feature set to items

We computed the mass function for each feature, but they cannot be combined using Dempster's combination rule as the sets correspond to different features with different domains. The solution to that, provided by [33] and mentioned here in subsection 5.2.2, is to project the bpa of each feature to the item set i.e. compute a bpa for items with regard to a feature. So, we are going to compute the projected bpa's for the features in our dataset.

Firstly, we need to get a hold of the movies that we are going consider after the process of deleting entries (rows) in the dataset.

```
movies_considered = set()
# get the unique movies from the user-centered dataset
for index, row in data_per_user_.iterrows():
    movies_considered |= row['movie_id'] # A |= B  -> A U B
# turn set into list for future purposes
movies_considered = list(movies_considered)
```

After that, we can compute the projected mass function with regards to each feature.

For the feature $c_{release\_decade}$:

```
# loop through all the movies we are considering
for i in range(len(movies_considered)):
    # make a temp df of the entries that share the particular movie_id
    df_temp = data.loc[movies_considered[i] == data['movie_id']]
    # for every release decade found in the temporary df (rd)
    for rd in df_temp['release_decade'].unique().tolist():
        # for every focal point of the characteristic
        for index_rd, row_rd in df_bpa_release_decades_.iterrows():
            # if the rd intersects with the focal point...
            if rd in row_rd['release_decades']:
                # ... then it belongs to the projected itemset
                row_rd['movies_proj'].add(movies_considered[i])
```

For the feature $c_{click\ hour}$, the computation is similar.

For the feature c_{genres}:

```
for i in range(len(movies_considered)):
    # make a temp df of the entries that share the particular movie_id
    df_temp = data.loc[movies_considered[i] == data['movie_id']]
    # for every genre(s) found in the temporary df (rd)
    for rd_ in df_temp['genres'].unique().tolist():
        # genres is a multi-valued feature and
        # genre are strings like this : 'Drama, Romance'
        split_list = rd_.split(", ")
        for rd in split_list:
            # for every focal point of the characteristic
            for index_rd, row_rd in df_bpa_genres_.iterrows():
                # if the rd intersects with the focal point...
                if rd in row_rd['genres']:
                    # ... then it belongs to the projected itemset
                    row_rd['movies_proj'].add(movies_considered[i])
```

When the the multi-valued features $click\ hour$ and $genres$ are projected onto movies it is possible that two or more identical sets may occur. Let's take a look at an example to understand why that event is possible.

Example 5.3.1. *Let's suppose we have a dataset where we only have three(3) movies, two(2) users that looks like in Table 5.1.*

Table 5.1: Dataset example

$Movie\ title$	$Click\ hour$	$Liked\ by\ user$
$'Pulp\ Fiction'$	$\{'Morning', 'Night'\}$	1
$'Reservoir\ Dogs'$	$\{'Morning', 'Night'\}$	2
$'The\ Hateful\ Eight'$	$\{'Evening', 'Morning', 'Night'\}$	1

Then, we can compute for the characteristic $click\ hour$ the respective bpa's: {'Morning', 'Night'} = $\frac{1}{2}$ and {'Evening', 'Morning', 'Night'} = $\frac{1}{2}$. After proceeding to project the bpa's to movies the result is:

Click hour set	***Projected set***	***bpa***
{'Morning', 'Night'}	*{'Pulp Fiction', 'Reservoir Dogs', 'The Hateful Eight'}*	*0.5*
{'Evening', 'Morning', 'Night'}	*{'Pulp Fiction', 'Reservoir Dogs', 'The Hateful Eight'}*	*0.5*

As the computations take place with the projected set, when the above result occurs, the two projected sets should be zipped into one and their bpa's should be summed. In this example, that processing would result to the following projected set:

Projected set	***bpa***
{'Pulp Fiction', 'Reservoir Dogs', 'The Hateful Eight'}	*1.0*

So, we should consider that situation in our implementation, as we could end up with multiple belief assignments for the same set (for the multi-valued characteristics). This can be handled by adding the belief assignment values of the same set and keeping only one instance: the one that contains the summation. The below code showcases that procedure for the feature c_{genres}. For $c_{click\ hour}$, the computation is similar.

For the feature c_{genres}:

```
# drop columns that are not needed
df_bpa_genres_
= df_bpa_genres_.drop(['genres', 'count', 'movies_proj'], axis=1)
# get the sets and store them into a list
list_of_sets_l =
[sorted(list(s)) for s in df_bpa_genres_["sets_indexed"].tolist()]
# sort the list
list_of_sets_l.sort()
# get only one istance of each set
list_of_sets_l =
list(list_of_sets_l for list_of_sets_l,_ in itertools.groupby(list_of_sets_l))
# create new dict and then new df
dictionary =
{'sets_indexed': list_of_sets_l, 'bpa' : [0.0 for i in range(len(
    list_of_sets_l))] }
df_bpa_genres_n = pd.DataFrame(dictionary)

# from old df sum bpa of same sets
for index_gn, row_gn in df_bpa_genres_n.iterrows():
    for index_g, row_g in df_bpa_genres_.iterrows():
        if sorted(list(row_g['sets_indexed']))
        == sorted(list(row_gn['sets_indexed'])):
            df_bpa_genres_n.at[index_gn,'bpa']+= row_g['bpa']
```

Note: Because Dempster's rule in the *ECLiPSe Prolog* implementation combines sets of int's where each int $i : i \in \{1, \ldots, |\Omega|\}$ we did an extra operation and converted the item-sets to int-sets with $1-1$ correspondence. These sets are stored in the column $'sets\_indexed'$ in the *DataFrame*.

6. USING THE IMPLEMENTATION OF DEMPSTER-SHAFER THEORY IN CONSTRAINT LOGIC PROGRAMMING FOR THE APPLICATION

In the previous section an application of DS Theory in Recommender Systems was studied and a dataset was provided in order to explain the process computationally by code in a real-life example. In this section we are going to use the implementation of Dempster's rule in CLP in order to combine the projected mass functions we computed earlier and compute the belief of some item-sets. After that we will measure how well did the *ECLiPSe* implementation of DST perform when we compute the mass joint and belief regarding the dataset of application.

6.1 Package PyCLP

We will present Dempster's rule *Prolog* implementation in order to combine preferences for movies and the belief implementation for computing the belief of different events. As mentioned in Chapter 4, the implementation of DST in CLP was developed in the *ECLiPSe Prolog* environment. The dataset pre-processing was done in *Python*, so the library *PyCLP* [5] was installed in order for *Python* to communicate with *ECLiPSe Prolog*. The *PyCLP* can interface with *ECLiPSe*'s version 6.1 and that is the version we will use. Here we are going to describe the modules of the package and explain the differences between the interface and the *ECLiPSe* engine. Only after that we are going to introduce the code and it's components.

6.1.1 PyCLP modules

In this paragraph we will present the modules we used in our code in order to start our *ECLiPSe* environment, load the needed predicates and initialize and use terms.

To begin with, to initialize the *ECLiPSe* engine and prior to any other function the `pyclp .init()` method is called.

After the initialization, in order to resume the *ECLiPSe* engine, `pyclp.resume(in_term=None)` is used, with an optional argument for the *Prolog* predicate `yield/2`. This function returns a tuple :

- *(pyclp.SUCCEED,None)*, if execution succeed (equivalent to True)
- *(pyclp.FAIL,None)*, if the goal fails
- *(pyclp.FLUSHIO, stream_number)*, if some data is present in stream $<$ stream_number $>$

- *(pyclp.WAITIO, stream_number)*, if *ECLiPSe* engine try to read data from stream $<$ stream_number $>$
- *(pyclp.YIELD, yield_returned_value)*, in case of predicate call `yield/2`
- *(pyclp.THROW, Term TagExit)*, an event have been thrown and no one have caught it

To create an Atom, `pyclp.Atom(atom_id)` is called, where `atom_id` is a string parameter with the atom's name.

Two classes that are often used in our code are `pyclp.Var`, that creates a *Prolog* variable as well as the class `pyclp.PList` that creates and reads *Prolog* lists. A list is constructed from an instance of *Python* list or tuple and *Python* strings, floats and integers are automatically transformed into terms. `PList`'s are the only terms in *PyCLP* whose value can change.

Another useful class of pyclp is `pyclp.Compound(functor_string, *args)`, that creates compound terms. In this Thesis by the term 'compound' we shall refer to these compound terms .Here, also, *Python* strings, floats and integers are automatically transformed in terms. It's arguments are: `functor_string`, a string with functor name and `args`, any number of arguments of type integer, float, string and PList, Atom, Compound.

When aiming to post a goal, the method `post_goal()` from the class `pyclp.PList` is used.

In closing, to shutdown the *ECLiPSe* engine, we call the function `pyclp.cleanup()`. After this function is called any operation on pyclp object or class produces undefined behaviour.

For more references one can visit PyCLP Module Reference.

6.1.2 PyCLP particularities

In this case, *PyCLP* was used in order to load Kaltsounidis' code as a module in the *Python* environment and utilize the exported Dempster's rule and belief in order to combine the mass functions and compute belief. The mass functions, as well as other compounds that were needed, were written in another file that was used by the program where Dempster's rule is located as a module. These compounds where not created in the *ECLiPSe* engine generated by *PyCLP*, but were stored in a new file, as the first option does not work. That would work only if all Kaltsounidis' code was transferred in *PyCLP* using the classes and functions presented earlier. For this to happen external predicates are needed which is a functionality provided by *PyCLP*. Specifically, the function `pyclp.addPythonFunction()` registers a *Python* function to be called from *ECLiPSe* using the predicate `call_python`. An example from the module reference manual is seen below:

```
from pyclp import *
```

```
def external_predicate(arguments):
   # arguments store all arguments passed with call_python_function
   # implements unify as described in ec_unify
   return unify(arguments[0],arguments[1])

# init ECLiPSe engine
init()
# register function with 'my_name' atom
add_python_function('my_name',external_predicate)
my_var=Var()
# call_python_function,'my_name',[1,My_var])
Compound('call_python_function',Atom('my_name'),[1,my_var]).post_goal()
resume()
if my_var.value() != 1:
   print("Failed resume ")
```

Even though that seems like a manageable option, there is a feature of this functionality that make the process of $1-1$ transferring all the *Prolog* code to *Python* quite difficult. In the registered *Python* function it is possible to use all classes to represent an *ECLiPSe* object (Atom(), Compound() etc.) but not `pyclp.resume()`, `pyclp.init()`, `pyclp.cut()` or `pyclp.cleanup()`. This means that if `pyclp.resume()` or `pyclp.cut()` cannot be called, the debugging process becomes difficult to perform and handle, especially when nested user-defined predicates are used.

But, external predicates are not the only obstacle to translate *Prolog* code to *Python*. Another problem is that compounds are created, but not universally stored and for that reason predicates like `findall/3` and `bagof/3` do not work as expected. Let's take a look at an example:

```
init()  # init ECLiPSe engine

print("Compounds are locally created")

my_compound1=Compound("b",1) # b(1).
my_compound2=Compound("b",5) # b(5).
# create a prolog variable L
L = Var()
# findall(X, b(X), L)
Compound("findall", X, Compound("b", X), L).post_goal()
res, st = resume()  # resume execution of ECLiPSe engine
print("findall(X,b(X),L) is:", res, "and value of L is: ", L.value())

if res == FLUSHIO:
  # open output stream
  outStream = Stream(st)
  # return data in a bytes object
  data = outStream.readall()
  print(data)
  # not required but implemented to support RawIO protocol
  outStream.close()

cleanup()  # shutdown ECLiPSe engine
```

Listing 6.1: findall/3 example, creating compounds through *PyCLP*

The expected output of the code snippet in Listing 6.1 is:

```
Compounds are locally created
findall(X,b(X),L) is: True and value of L is:  [1,5]
```

But the actual output is:

```
Compounds are locally created
findall(X,b(X),L) is: 7 and value of L is:  None
b'calling an undefined procedure b(_1663) in module ECLiPSe\n'
```

From the output it is clear that some other way of defining procedure `b()` must be found. Let's take a look at another example, which is the same as Listing 6.1 with the difference that the compounds `b(1)` and `b(5)` are stored as a module in the file `b.pl`.

```
init()  # init ECLiPSe engine
print("\nCompounds are loaded from file")
L = Var()  # create a prolog variable L
# use_module b.pl
Compound("use_module",Atom("/home/tatiana/b.pl")).post_goal()
# findall(X, b(X), L)
Compound("findall", X, Compound("b", X), L).post_goal()
res, _ = resume()  # resume execution of ECLiPSe engine
print("findall(X,b(X),L) is:", res, "and value of L is: ",
    L.value())
cleanup()  # shutdown ECLiPSe engine
```

Listing 6.2: findall/3 example, loading compounds from module

Listing 6.2 produces the correct-expected output:

```
Compounds are loaded from file
findall(X,b(X),L) is: True and value of L is:  [1,5]
```

The conclusion is that for our *ECLiPSe* engine to access and use compounds they need to be stored in an external file, which is inconvenient, but, unfortunately, it is something that cannot be avoided.

There is another characteristic of the *PyCLP* library that should be taken into consideration when using it. When there are more than one evaluations of a variable that make a predicate true, `pyclp.resume()` returns (True, None), but the variable is not instantiated. If `findall/3` is performed in order to collect all the values, the value of the list returned is not the one expected. Again, we present an example.

```
init()  # init ECLiPSe engine

Compound("lib", "ic_sets").post_goal() # lib(ic_sets)

X = Var()  # create a prolog variable X
L = Var()  # create a prolog variable L

Compound("subsetof",PList([1]), PList([1,2,3])).post_goal()
res, _ = resume()  # resume execution of ECLiPSe engine
print("[1] subsetof [1,2,3] is:", res)

Compound("subsetof", X, PList([1,2,3])).post_goal()
res, _ = resume()  # resume execution of ECLiPSe engine
print("X subsetof [1,2,3] is:", res)
print("Value of X is: ", X.value())

# findall(X,subsetof(X,[1,2,3]),L)
Compound("findall", X, Compound("subsetof", X, PList([1,2,3])), L).post_goal()
res, _ = resume()  # resume execution of ECLiPSe engine
print("findall(X,subsetof(X,[1,2,3]),L) is:", res)
print("Value of L is: ", L.value())

cleanup()  # shutdown ECLiPSe engine
```

Listing 6.3: ic_sets:subsetof/2 example

The expected output of the code snippet in Listing 6.3 is:

```
[1] subsetof [1,2,3] is: True
X subsetof [1,2,3] is: True
Value of X is:  X{([] .. [1, 2, 3]) : _240{0 .. 3}}
findall(X,subsetof(X,[1,2,3]),L) is: True
Value of L is:  [X{([] .. [1, 2, 3]) : _215{0 .. 3}}]
```

The actual output is:

```
[1] subsetof [1,2,3] is: True
X subsetof [1,2,3] is: True
Value of X is:  None
findall(X,subsetof(X,[1,2,3]),L) is: True
Value of L is:  [_]
```

To sum up, *PyCLP* has some limitations compared to *ECLiPSe Prolog* environment and for that reason code written in *Prolog* may not be able to be written in *Python*. For our code, these limitations caused some difficulties but were not deterrent and everything works as expected.

6.2 Computation of the mass joint and belief from dataset

6.2.1 Alteration of the initial implementation of Kaltsounidis

As the *PyCLP* module was explained previously, we are going to use it in order to combine user-preferences and compute belief of item-sets from the dataset of application. As mentioned earlier, Dempster's rule was implemented in [16] and that is the code we use with few minor alterations. These alterations will be explained shortly, in order for the code to be understandable.

The first alteration is that the file `dst_master.pl`, where the combination rule is written, is used as a module and the needed predicates are exported to our *Python* script.

```
# :- use_module(dst_master.pl)
Compound("use_module",Atom("/home/tatiana/dst_master.pl")).post_goal()
```

The second one is that the mass functions (`m/3`), Theta (`theta/1`) and number of mass functions (`num_of_m/1`) are stored by *Python* in a *Prolog* module that is used by `dst_master.pl`.

```
# create module that will contain the information about
# the Universe: theta, num_of_m, mass functions
file_object = open('/home/tatiana/mass_func.pl', 'w+')

# write the information for module exporting
file_object.write(":-module(mass_func).\n")
file_object.write(":-export(m/3).\n")
file_object.write(":-export(theta/1).\n")
file_object.write(":-export(num_of_m/1).\n")

# write about theta and num_of_func
file_object.write("\ntheta("+str(Theta)+").\n")
file_object.write("num_of_m(3).\n")

file_object.write("\n")
for m_num, focal_set, bpa in m1_rd:
  file_object.write("m("+str(m_num)+","+str(focal_set)+","+str(bpa)+").\n")

file_object.write("\n")
for m_num, focal_set, bpa in m2_ch:
  file_object.write("m("+str(m_num)+","+str(focal_set)+","+str(bpa)+").\n")

file_object.write("\n")
for m_num, focal_set, bpa in m3_g:
  file_object.write("m("+str(m_num)+","+str(focal_set)+","+str(bpa)+").\n")

file_object.close()
```

Also, the predicate's `bpa/2` arity was altered to `bpa/3` as for the plots that will be presented in the next section we needed an argument that represents the number of intersecting combinations. It's functionality remains the same.

```
%Tars are the subsets and NewVals their corresponding values
%bpa( -Tars, -NewVals, -TarsAndVals) produces the new focal points
%Tars, their combined values NewVals and returns combinations list
bpa(Tars, NewVals, TarsAndVals):-
    theta(Theta),
    findall( (A,Val), compute(A, Val, Theta), TarsAndVals),
    sort(TarsAndVals, SortedTarsAndVals),
    sum_same_sets(SortedTarsAndVals, NewTarsAndVals),
    split_to_Tars_Vals(NewTarsAndVals, Tars, Vals),
    sum(Vals,Div),
    list_div(Div, Vals, NewVals).
```

Last but not least, we created a new predicate `belief_comb/3` that has the same functionality as the predicate `belief/2`, with the difference that it returns the `Vals` list from which we obtain the number of combinations. We did not change the existing predicate but created a new, as not in all test cases in this article we care about the number of combinations and for those we don't there is no need of adding an extra functionality.

```
%belief_comb(+A, -V, -Vals) finds bel, V, for set A
% and returns Vals list
belief_comb(A, V, Vals):-
    findall(Val, compute(_,Val,A), Vals),
    norm_compute(Vals, V).
```

6.2.2 Code's structure

Until now, we have mentioned some alterations that we did to the code of [16], not only regarding the predicates, but also regarding the structure of the code as we use *ECLiPSe* through *Python*. Also, we noted that we will use test cases in order to measure the performance of the implementation when applied on the dataset. All this information may be confusing to the reader when thinking about the architecture of the system, but mostly when thinking about the data's flow.

For this reason, the Data Flow Diagram (DFD) 6.1 is presented. In 6.1, the external entity User provides the dataset to the script `dst_application_mult_exec.sh` which runs multiple executions of the process `dst_applycation.py` providing each time a different fraction of the dataset to be used. The process `dst_applycation.py` creates the process `mass_func.pl` that is imported as a module from the process `dst_master.pl`. The latter provides `dst_applycation.py` with the predicates needed to compute belief and the mass joint. The results of the computations accompanied with some information are stored in five(5) data stores (`.csv` files), that accord the input to the processes `plot_results_DR.py` and `plot_results_Bel.py`. Those two processes illustrate the plots that we will use as a measure of the implementation's performance.

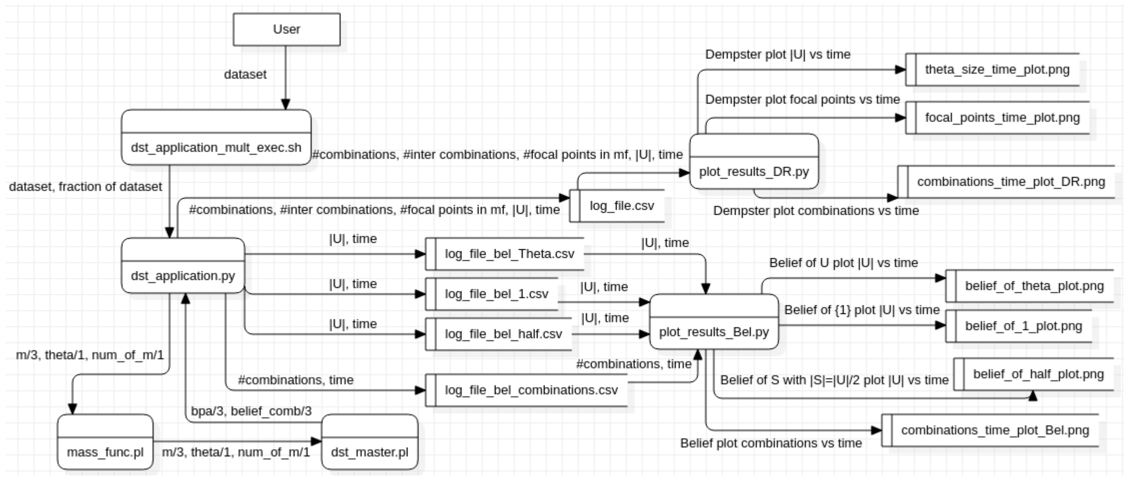

User
dataset
dst_application_mult_exec.sh
#combinations, #inter combinations, #focal points in mf, |U|, time
dataset, fraction of dataset
#combinations, #inter combinations, #focal points in mf, |U|, time
log_file.csv
plot_results_DR.py
Dempster plot |U| vs time
theta_size_time_plot.png
Dempster plot focal points vs time
focal_points_time_plot.png
combinations_time_plot_DR.png
Dempster plot combinations vs time
dst_application.py
|U|, time
log_file_bel_Theta.csv
|U|, time
log_file_bel_1.csv
|U|, time
log_file_bel_half.csv
#combinations, time
log_file_bel_combinations.csv
plot_results_Bel.py
Belief of U plot |U| vs time
belief_of_theta_plot.png
Belief of {1} plot |U| vs time
belief_of_1_plot.png
Belief of S with |S|=|U|/2 plot |U| vs time
belief_of_half_plot.png
Belief plot combinations vs time
combinations_time_plot_Bel.png
m/3, theta/1, num_of_m/1
bpa/3, belief_comb/3
mass_func.pl
m/3, theta/1, num_of_m/1
dst_master.pl

Figure 6.1: Data Flow Diagram of DST Application using CLP

So, the main process in our program is the `dst_applycation.py`, that does all the preprocessing in the dataset, computes the mass functions as explained in Chapter 5 and then the mass joint and belief. Taking into consideration the altered predicates and *PyCLP*'s functionality, an example of the code in `dst_applycation.py` used to call Dempster's rule and belief's predicates is listed below.

```python
# :- use_module(dst_master.pl)
Compound("use_module",Atom("/home/tatiana/dst_master.pl")).post_goal()

# resume execution of ECLiPSe engine
result, dummy = resume()
if result != SUCCEED:
  print(result,dummy)

# create variables for Dempster's rule
V=Var()
CombinedMass=Var()
TarsAndVals=Var()

# create compound for Dempster's rule and post goal
Compound("bpa",CombinedMass,V,TarsAndVals).post_goal()

# compute Dempster's rule
result, dummy = resume()
if result != SUCCEED:
  print(result,dummy)

# create variables for belief
A = Var()
B = Var()
C = Var()

# initialize A as [1]
Compound("=", A, PList([1])).post_goal()
```

```
resume()

# create compound for belief and post goal
Compound("belief_comb",A,B,C).post_goal()

# compute belief of {1}
result, dummy = resume()
if result != SUCCEED:
  print(result,dummy)
```

Listing 6.4: Computing mass joint and belief of {1} for the application-dataset

As explained in the comments, the code's purpose is to compute the mass joint from the three mass functions of the dataset and the belief of set $\{1\}$. The actual code we used is based on the one showed in Listing 6.4, but is more enhanced as we cared to produce many test cases and store data for the plots presented next.

6.3 Results

After seeing the outline of the code, the next step is to evaluate the performance of it, which, in our case, is the the performance of Kaltsounidis' implementation when used on the application of DST in Recommender Systems and specifically on our chosen dataset. In this section we will do just that, by providing some plots to showcase the relation between the computational time and some other features of the data set. Before proceeding to the plots, the reader should take into consideration that the code was executed in a Dell, Inspiron 15, 5000 series laptop with Intel Core i5 5200U, 8 GB RAM and 1 TB HDD, so in another machine the time results could be different, but we are more interested in time growth and not the time value itself.

6.3.1 Computing the mass joint

In this section we demonstrate some figures that showcase the performance of the *ECLiPSe Prolog* implementation of Dempster's rule when used to combine mass functions from the chosen dataset. In particular, three(3) figures are presented. Figure 6.2 shows the relation between the time taken by Dempster's rule to compute the combined mass in nanoseconds and the number of focal points in every mass mass function. The second figure, 6.3, demonstrates the relation between the time it took Dempster's rule to compute the combined mass in nanoseconds and the number of intersecting / non-intersecting combinations. The third one, 6.4, displays the relation between the time it took Dempster's rule to compute the combined mass in nanoseconds and the size of Θ.

In the first plot, 6.2, the number of focal points regarding the first mass function are colored blue, the ones regarding the second mass function are colored orange and the ones

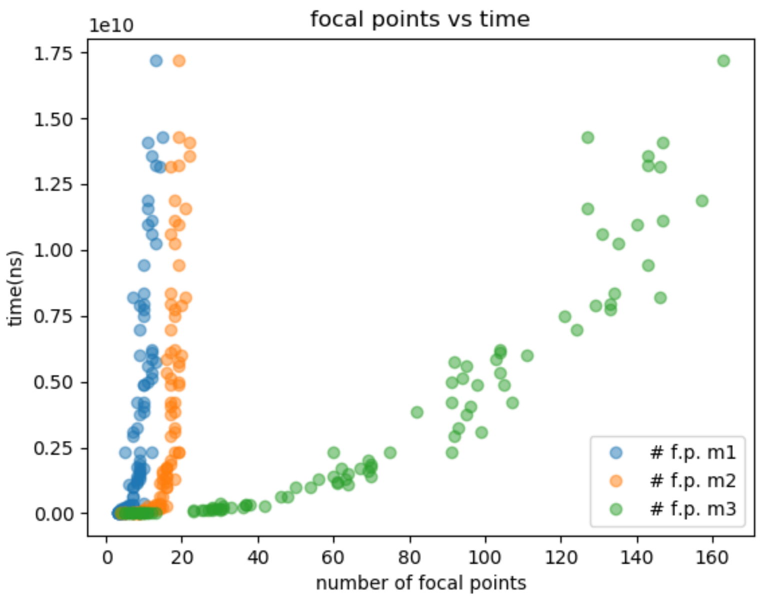


Figure 6.2: Dempster's rule : Plot - time(ns) and number of focal points per mass function

regarding the third, green. Here, the growth of the focal points of mass functions 1 and 2 regarding the time is exponential. This happens because due to the small domain of those two features ($|release\_date| \simeq 5$ and $|click\_hour| \simeq 5$) even when we consider a large fraction of the dataset, the number of possible focal points does not change drastically. This does not apply to the third mass function, as the feature $genres$ has a large domain and the number of its focal points increases as the considered fraction is increased. We may conclude that the summation of the focal points of the three mass functions shares a similar behaviour to the plot of the focal points of the third mass function regarding time, which is that as the number of focal points increases the computational time increases as well.

From the plot 6.3 we notice that as the computational time increases, the number of combinations either intersecting or not becomes larger and vice versa. Again, as the number of combination grows larger, the growth of time is exponential. This behavior is expected, as CLP implementation produces results in reduced time when the number of non-intersecting combinations is large.

Continuing to the observation of the plot 6.4, as one can see, increased computational times correspond to larger Θ sizes and for $|\Theta| \leq 200$ the computational time is low.

Decisively, the computation of the combined mass functions of the dataset is efficient, for the amount of mass functions and their focal points that were provided. Even for big Θ's the computational time is acceptable as the maximum is $1.75 \times 1e10$ ns $= 17.5$ seconds. For a larger amount of mass functions, focal points and movies the performance

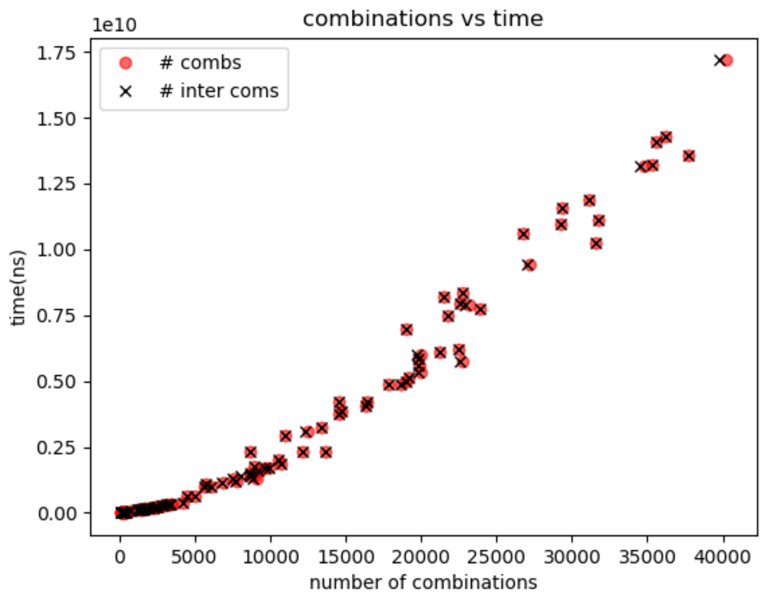


Figure 6.3: Dempster's rule : Plot - time(ns) and number of combinations

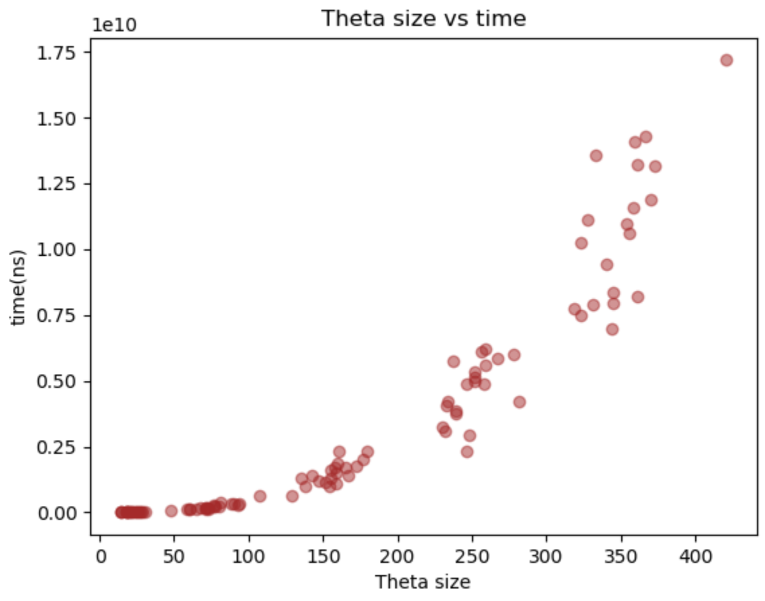


Figure 6.4: Dempster's rule : Plot - time(ns) and Θ size

could not be measured, due to lack of processing power which prevents us from coming to general conclusions.

6.3.2 Computing belief

As we demonstrated the performance of Dempster's rule, it is anticipated that we study the performance of belief computation of item-sets in the dataset of application. For that reason, we introduce four(4) plots. Plot 6.5 shows the relation between the number of combinations performed by `belief_comb/3` predicate and belief's computational time. The next three Figures 6.6, 6.7 and 6.8 present the relation between the size of the Universe and the time it took *Prolog* to compute the belief of the set $\{1\}$, a set S in the powerset of Θ with the size of S being halt of Θ's and the belief of Θ respectively. Note that we won't be studying the performance of plausibility, as the plausibility of a set A can be computed through the belief of $\overline{A}$.

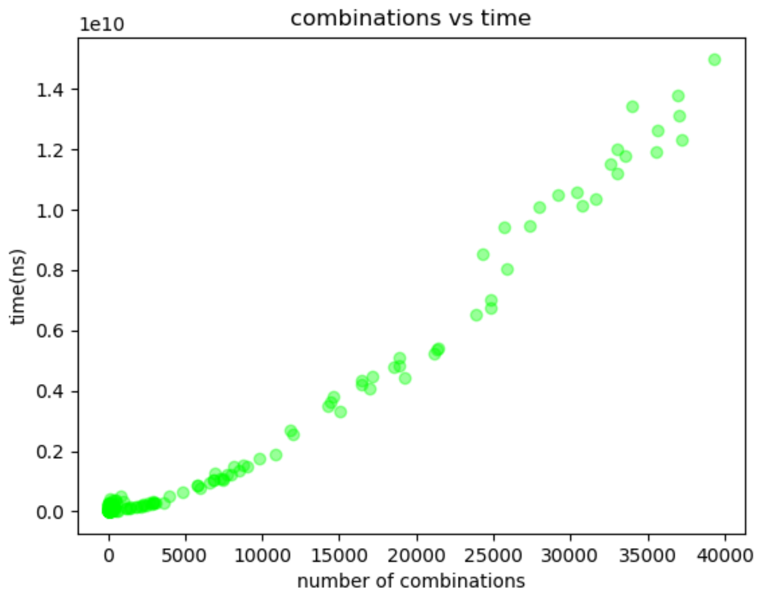


Figure 6.5: Plot - time(ns) and number of combinations

Figure 6.5 shows a linear growth of the time needed to compute the belief (regardless the set) as the number of combinations ascends. This time is high for a large number of combinations as, for example, for $number\ of\ computations \backsimeq 40000$ the time needed is $\backsimeq 1.5 \cdot 1e10ns = 15000000000ns = 15seconds$, but still acceptable, as most likely these combinations correspond to the computation of Θ's belief.

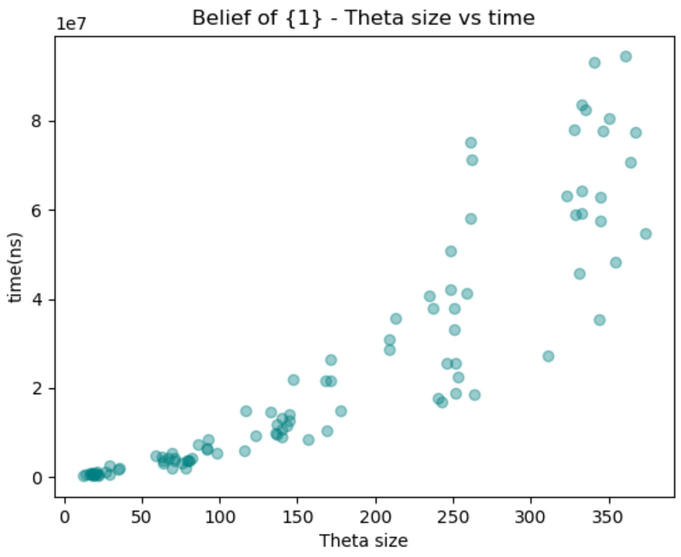


Figure 6.6: Belief of $\{1\}$: Plot - time(ns) and Θ size

Each of the Figures 6.6, 6.7 and 6.8 shows the computational time of set's belief regarding the $|\Theta|$. In general, in all plots one can see that as $|\Theta|$ grows, time grows higher as well. Computing belief of sets with bigger cardinality, though, is more time consuming as one may notice in the 'time' axis of the three Figures, since it ranges $(0 - 1.5) \cdot 1e7$ in 6.6, $(0 - 5) \cdot 1e8$ in 6.7 and $(0 - 1.5) \cdot 1e10$ in 6.8.

What is interesting is commenting on the growth of each Figure. Starting backwards, in Figure 6.8 we notice a more "strict" ascend. By that we mean that, if we were to draw a line to follow these points, an exponential one would fit. From this we could assume that as $|\Theta|$ rises, the time to compute the belief of it rises with an exponential growth. As for the other two Figures, 6.6 and 6.7, we do notice that for small Θ's the computational time of $\{1\}$ and S such that $\in 2^{\Theta}$, $|S| = |\Theta|/2$ is low, but as the Universe grows larger, it is not necessary that the time grows with a fixed rate, or follows a specific function as before. From that we conclude that the computational time of belief for a set $A \subset \Theta$ depends, to a degree, on the size of Θ, but also on other parameters such as the number of focal points and the intersecting sets, which is essentially the number of combinations. For sure for computing Θ's belief the number of combinations is relevant, but in this case this number is fixed in the system, as the number of combinations equals to the number of possible combinations of Dempster's rule ($|focal\ points\ per\ mass\ function|^{|mass\ functions|}$), whereas for two sets A, B such that $A, B \subset \Theta$ and $|A| = |B|$, the number of combinations when computing their belief may differ.

Here as well we can conclude that the belief's computations is efficient for the data provided. The maximum time of computation even for a large set and a large Θ is normal and one would chose this implementation to compute belief for this dataset.

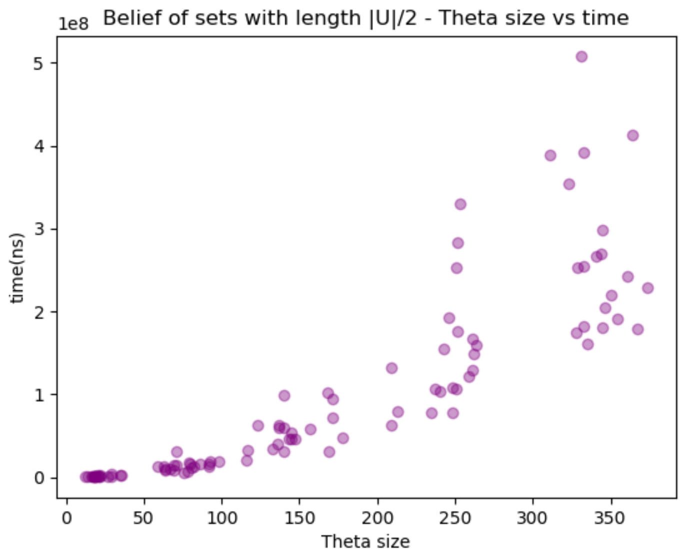


Figure 6.7: Belief of $S \in 2^{\Theta}$, $|S| = |\Theta|/2$: Plot - time(ns) and Θ size

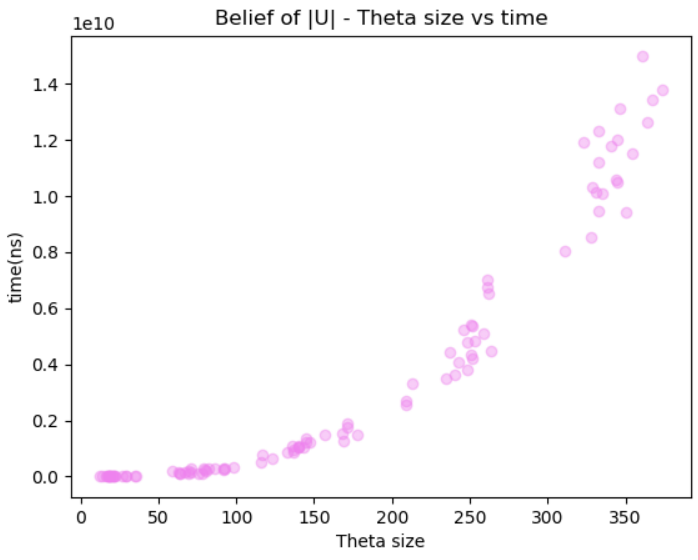


Figure 6.8: Belief of Θ : Plot - time(ns) and Θ size

7. COMPARING ECLIPSE'S AND IBELIEF'S IMPLEMENTATIONS

Both *ECLiPSe Prolog*'s and *ibelief*'s implementations provide functions to perform mass combinations as well as compute belief. In *ECLiPSe* the code is developed with libraries that use constraints, whereas in *ibelief* belief and plausibility functions use FMT's. We are going to explore what implementation performs better and under which circumstances.

7.1 Test cases

Our first thought was to compare the two implementations by their performance in the application described in Chapter 5. Noticeably, we had no problem using *ECLiPSe*'s implementation in order to compute the combined mass and belief. However, when using *ibelief*'s implementation we faced some obstacles. These obstacles were: creating the mass functions in the correct format so that they could be used as an input in *ibelief*'s combination rules.

Let's elaborate on that. The function `DST` needs an input matrix of dimensions $2^{|\Theta|} \times |mass\ functions|$ that represents the mass to be combined. Each column represents a mass function and every cell represents the belief assignment of a set that belongs to the Universe's powerset. These belief assignments are not random, but correspond to sets sorted first by size and then by lexicographical order. So, after projecting the feature-based mass functions to item sets in order to create mass functions that correspond to the same Universe, we needed to map these item sets to the Universe's powerset. This operation not only has a high computational cost but is against the logic of DST's application presented in [33]. This application prevents us from computing the whole powerset of Θ which exempts us from computations of exponential complexity. So, mapping our item sets to their position in the powerset introduces again this high complexity that we were able to avoid when computing the projected mass functions. For this reason, we decided to measure only *Prolog*'s performance in the application of DST and compare the two implementations using random test cases, that will be described implicitly later on.

7.2 Code's structure

We did analyze the libraries used to produce the results, but in order for the reader to understand the components of the program and their connection through data, the Data Flow Diagram 7.1 is presented.

In our system, the external entity User provides to `comparison_mult_exec.sh` the $|U|$, the number of mass functions and the number of focal points that will be used throughout the multiple execution of the processes `ibelief_results.py`, `prolog_results_Dempster.py`, `prolog_results_bel.py` and `example_gen_ibelief_comparison.cpp`. The process `example_gen_ibelief_`

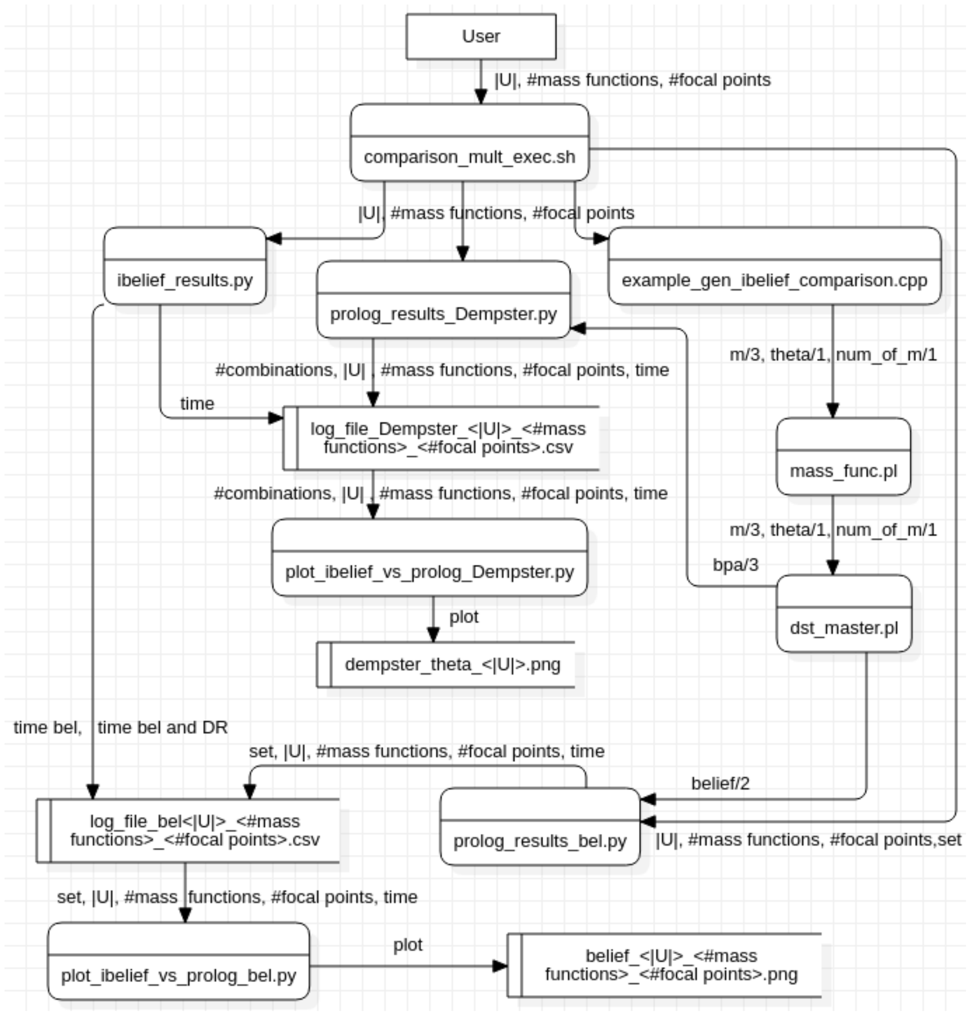

User
|U|, #mass functions, #focal points
comparison_mult_exec.sh
|U|, #mass functions, #focal points
ibelief_results.py
prolog_results_Dempster.py
example_gen_ibelief_comparison.cpp
#combinations, |U|, #mass functions, #focal points, time
time
log_file_Dempster_<|U|>_<#mass functions>_<#focal points>.csv
m/3, theta/1, num_of_m/1
mass_func.pl
#combinations, |U|, #mass functions, #focal points, time
plot_ibelief_vs_prolog_Dempster.py
m/3, theta/1, num_of_m/1
bpa/3
dst_master.pl
plot
dempster_theta_<|U|>.png
time bel, time bel and DR
set, |U|, #mass functions, #focal points, time
belief/2
log_file_bel<|U|>_<#mass functions>_<#focal points>.csv
prolog_results_bel.py
|U|, #mass functions, #focal points,set
set, |U|, #mass functions, #focal points, time
plot_ibelief_vs_prolog_bel.py
plot
belief_<|U|>_<#mass functions>_<#focal points>.png

Figure 7.1: Data Flow Diagram of comparison of the two implementations

`comparison.cpp` is the one creating the test cases for the *ECLiPSe* implementation and is an alteration of a similar process implemented in [16]. The process `ibelief_results.py` computes the mass joint and belief using the *ibelief* module, while `prolog_results_Dempster.py` and `prolog_results_bel.py` compute the mass joint and the belief of a set, respectively, using *Prolog*. Then, the results for the computation of belief for every specific data combination ($|U|$, number of mass functions and number of focal points) are stored in a unique `.csv` file and the same happens with Dempster's rule computations. These data stores are then used by the two *Python* scripts `plot_ibelief_vs_prolog_Dempster.py` and `plot_ibelief_vs_prolog_bel.py` to produce the corresponding plots.

7.3 Results

In the section, at first, we will present the results regarding the computation of the combined mass function and, then, the results regarding the computation of the belief function.

7.3.1 Computing the mass joint

In order to measure and discuss the performance of the two implementations several test cases were created. These cases, we experimented with Θ size, number of mass functions and number of focal points per mass function. Specifically, the test cases were combinations of those three measures where $|\Theta| \in \{10, 15, 20, 25\}$, number of mass functions $\in \{2, 3, \cdots 10\}$ and number of focal points per mass function $\in \{4, 6, \cdots, 14\}$.

Note that the test cases of these implementations *are not the same for the both approaches*. Since we are randomly generating them, the only thing we provide and assure these cases have in common is the Θ, the number of mass functions and the number of focal points, which means that the test cases may differ on the focal point's placement as well as their values. For these test cases, the comparison was made regarding the amount of combinations, which is $|focal\ points\ per\ mass\ function|^{|mass\ functions|}$, but not regarding the intersecting combinations, as they differed for each simultaneous input of each implementation. Like we described earlier, it is computationally heavy in *ibelief*'s implementation to find the index of a subset in the combined mass' matrix. Thus, the comparison won't be made regarding the intersecting combinations.

The Tables 7.1, 7.2, 7.3 and 7.4 show the results for a fixed Θ size and the Figures 7.2, 7.3, 7.4 and 7.5 the relation between time in nanoseconds and combinations. Note that the term 'inf' in the columns of the tables that showcase time denotes that for the corresponding data the implementation of Dempster's rule respectively did not compute the mass joint due to the large amount of memory needed (in case of both *Prolog* and *ibelief*) or it took a substantially large amount of time to compute the results (in case of *Prolog*).

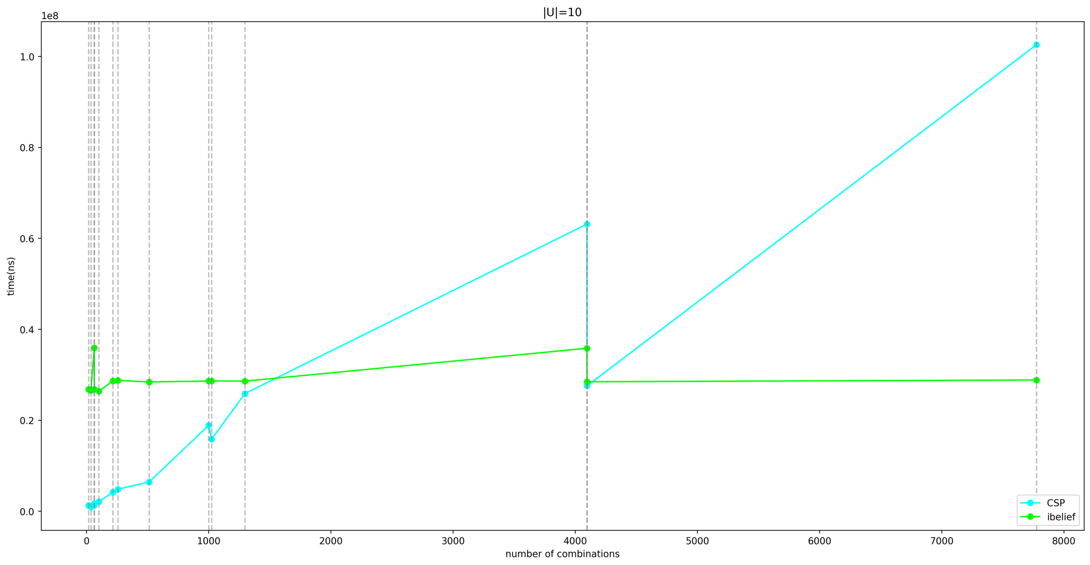


Figure 7.2: Dempster's rule : Plot - $|\Theta| = 10$, time(ns) and number of combinations

Before elaborating on the results from the tables we should mention an observation we

Table 7.1: Dempster's rule : Results for $|\Theta| = 10$, number of mass functions $\in \{2, 3, \cdots, 10\}$ and number of focal points per mass function $\in \{4, 6, \cdots, 14\}$

# combinations	# focal points per mass	# mass functions	prolog time(ns)	ibelief time(ns)
16.0	4.0	2.0	1292437.0	26797933.0
36.0	6.0	2.0	909922.0	26544086.0
64.0	4.0	3.0	1668409.0	35934945.0
64.0	8.0	2.0	1294463.0	26767646.0
100.0	10.0	2.0	2087335.0	26350989.0
216.0	6.0	3.0	4156883.0	28608844.0
256.0	4.0	4.0	4840496.0	28757001.0
512.0	8.0	3.0	6412341.0	28411180.0
1000.0	10.0	3.0	18847147.0	28606681.0
1024.0	4.0	5.0	15807758.0	28636158.0
1296.0	6.0	4.0	25838151.0	28587846.0
4096.0	4.0	6.0	63107997.0	35815056.0
4096.0	8.0	4.0	27544518.0	28449786.0
7776.0	6.0	5.0	102547800.0	28838079.0
10000.0	10.0	4.0	107223202.0	29042213.0
32768.0	8.0	5.0	258346754.0	28710539.0
46656.0	6.0	6.0	526539769.0	30750743.0
100000.0	10.0	5.0	1051353986.0	28748558.0
262144.0	8.0	6.0	1503998713.0	30880489.0
1000000.0	10.0	6.0	5388744823.0	30788290.0
100000000.0	10.0	8.0	inf	35690257.0
429981696.0	12.0	8.0	inf	31242430.0
1000000000.0	10.0	9.0	inf	32295844.0
1475789056.0	14.0	8.0	inf	31414183.0
5159780352.0	12.0	9.0	inf	32528683.0
10000000000.0	10.0	10.0	inf	32621344.0
20661046784.0	14.0	9.0	inf	32225067.0
61917364224.0	12.0	10.0	inf	32408753.0
289254654976.0	14.0	10.0	inf	32169609.0

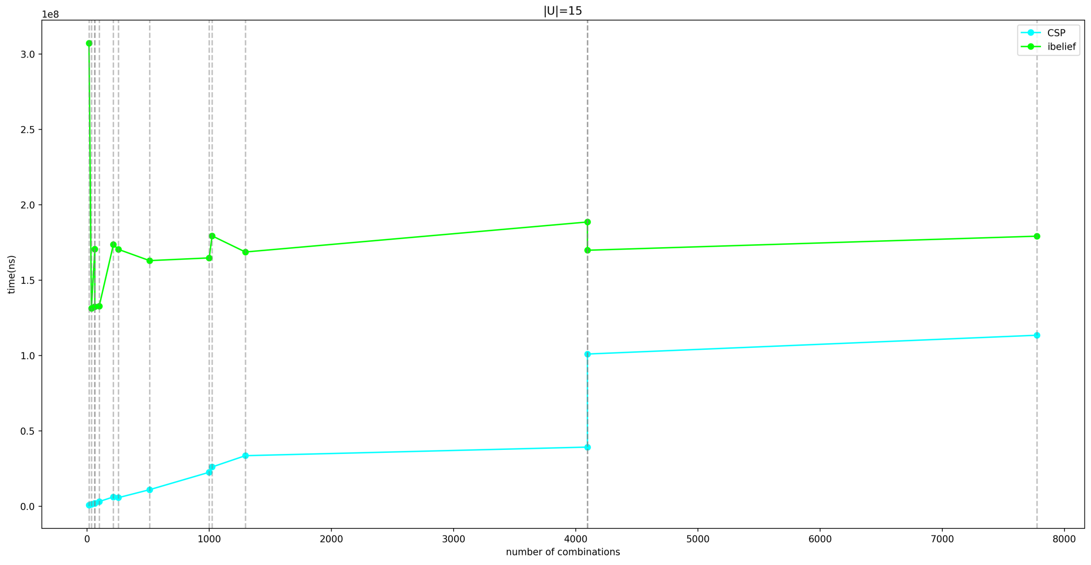


Figure 7.3: Dempster's rule : Plot - $|\Theta| = 15$, time(ns) and number of combinations

made when running the test cases regarding the computation of the combined mass through *ibelief*'s implementation of Dempster's rule. For some tests, if the number of mass functions is small, e.g. 2,3 or 4 when $|\Theta| = 10$, up to 6 when $|\Theta| = 20$, then the matrix of the mass joint contains some negative values. The amount of negative val-

Table 7.2: Dempster's rule : Results for $|\Theta| = 15$, number of mass functions $\in \{2, 3, \cdots, 10\}$ and number of focal points per mass function $\in \{4, 6, \cdots, 14\}$

# combinations	# focal points per mass	# mass functions	prolog time(ns)	ibelief time(ns)
16.0	4.0	2.0	650307.0	307086341.0
36.0	6.0	2.0	1193963.0	131125723.0
64.0	4.0	3.0	1835818.0	170421648.0
64.0	8.0	2.0	1791198.0	132225407.0
100.0	10.0	2.0	2990997.0	132703972.0
216.0	6.0	3.0	6064328.0	173506835.0
256.0	4.0	4.0	5565213.0	170321756.0
512.0	8.0	3.0	10860113.0	162833402.0
1000.0	10.0	3.0	22421459.0	164636218.0
1024.0	4.0	5.0	25940905.0	179248906.0
1296.0	6.0	4.0	33427735.0	168556146.0
4096.0	4.0	6.0	39123172.0	188456839.0
4096.0	8.0	4.0	100861624.0	169714678.0
7776.0	6.0	5.0	113333022.0	179050190.0
10000.0	10.0	4.0	151421327.0	170404507.0
32768.0	8.0	5.0	362481948.0	178947500.0
46656.0	6.0	6.0	722672648.0	184741121.0
100000.0	10.0	5.0	1714424770.0	174360808.0
262144.0	8.0	6.0	2588536081.0	184670414.0
1000000.0	10.0	6.0	12690947558.0	188298058.0
100000000.0	10.0	8.0	inf	359178186.0
429981696.0	12.0	8.0	inf	208862673.0
1000000000.0	10.0	9.0	inf	223772214.0
1475789056.0	14.0	8.0	inf	203652795.0
5159780352.0	12.0	9.0	inf	238656625.0
10000000000.0	10.0	10.0	inf	516761254.0
20661046784.0	14.0	9.0	inf	235277289.0
61917364224.0	12.0	10.0	inf	234746884.0
289254654976.0	14.0	10.0	inf	282396420.0

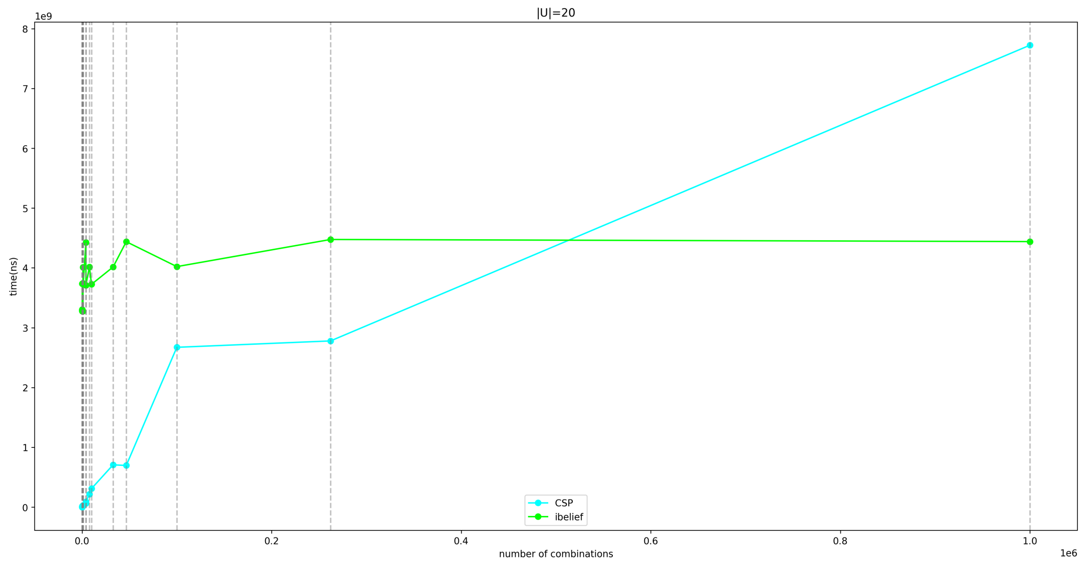


Figure 7.4: Dempster's rule : Plot - $|\Theta| = 20$, time(ns) and number of combinations

ues seem to go up when the number of focal points is quite larger than the number of mass functions. For example, this happens for `nbFocalElement=50, ThetaSize=20, nbMass=5, nbFocalElement=50, ThetaSize=10, nbMass=3` etc. Nonetheless, the summation of the matrix values is correct ($\simeq$ 1). Since we don't have access to the code that implements Demp-

Table 7.3: Dempster's rule : Results for $|\Theta| = 20$, number of mass functions $\in \{2, 3, \cdots, 10\}$ and number of focal points per mass function $\in \{4, 6, \cdots, 14\}$

# combinations	# focal points per mass	# mass functions	prolog time(ns)	ibelief time(ns)
16.0	4.0	2.0	515774.0	3288028486.0
36.0	6.0	2.0	1348252.0	3290169542.0
64.0	4.0	3.0	2296625.0	3278366156.0
64.0	8.0	2.0	2081308.0	3290209694.0
100.0	10.0	2.0	3240224.0	3302266385.0
216.0	6.0	3.0	5892695.0	3295881538.0
256.0	4.0	4.0	11816206.0	3736328412.0
512.0	8.0	3.0	17876675.0	3277489914.0
1000.0	10.0	3.0	24976686.0	3277787446.0
1024.0	4.0	5.0	29155612.0	4011749082.0
1296.0	6.0	4.0	32614797.0	3739609855.0
4096.0	4.0	6.0	62301190.0	4424250092.0
4096.0	8.0	4.0	90881997.0	3710901751.0
7776.0	6.0	5.0	218700943.0	4017582090.0
10000.0	10.0	4.0	309355675.0	3725569776.0
32768.0	8.0	5.0	707325395.0	4014971199.0
46656.0	6.0	6.0	696320020.0	4438223590.0
100000.0	10.0	5.0	2672132768.0	4021488021.0
262144.0	8.0	6.0	2778822925.0	4475155964.0
1000000.0	10.0	6.0	7723927897.0	4439892459.0
100000000.0	10.0	8.0	inf	5301937915.0
214358881.0	11.0	8.0	inf	5199852179.0
429981696.0	12.0	8.0	inf	5220471099.0
815730721.0	13.0	8.0	inf	5194784154.0
1000000000.0	10.0	9.0	inf	5504159994.0
1475789056.0	14.0	8.0	inf	5235751600.0
2357947691.0	11.0	9.0	inf	5754941650.0
5159780352.0	12.0	9.0	inf	5532124191.0
10000000000.0	10.0	10.0	inf	5896095161.0
10604499373.0	13.0	9.0	inf	5541434779.0
20661046784.0	14.0	9.0	inf	5506292544.0
25937424601.0	11.0	10.0	inf	5851495663.0
61917364224.0	12.0	10.0	inf	5875468916.0
137858491849.0	13.0	10.0	inf	5837184550.0
289254654976.0	14.0	10.0	inf	5863993507.0

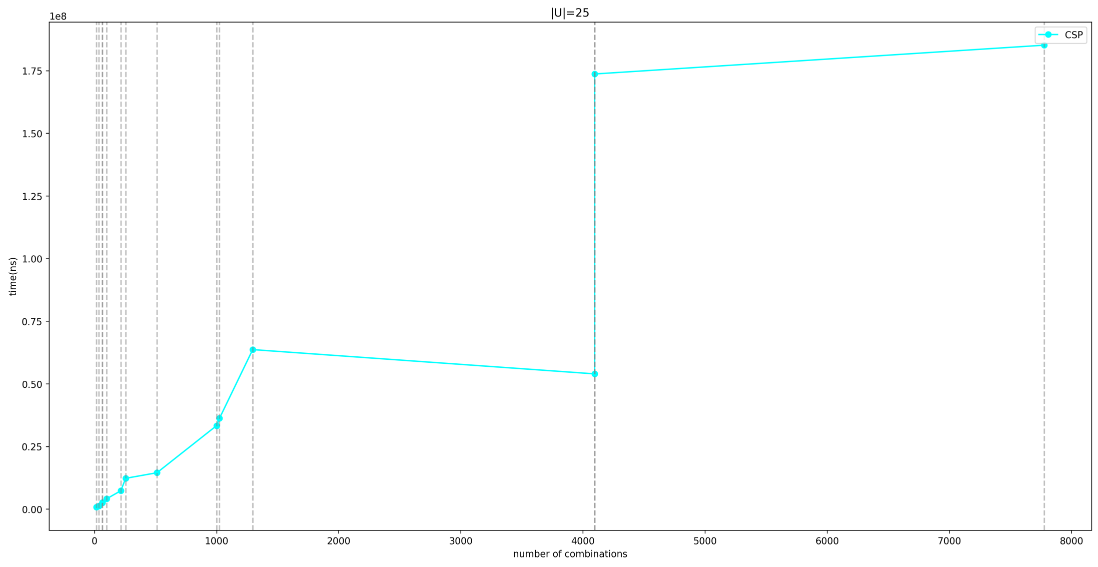


Figure 7.5: Dempster's rule : Plot - $|\Theta| = 25$, time(ns) and number of combinations

ster's rule in *ibelief*, we cannot spot the origin of this behavior, but only make assumptions. One assumption is that for a small amount of mass functions and a large amount of focal points the computations cause overflow. Another one is that the normalization constant K becomes larger than one and thus the computation $1 - K$ is < 0. But, again, these are

Table 7.4: Dempster's rule : Results for $|\Theta| = 25$, number of mass functions $\in \{2, 3, \cdots, 10\}$ and number of focal points per mass function $\in \{4, 6, \cdots, 14\}$

# combinations	# focal points per mass	# mass functions	prolog time(ns)	ibelief time(ns)
16.0	4.0	2.0	786986.0	inf
36.0	6.0	2.0	1235151.0	inf
64.0	4.0	3.0	2625847.0	inf
64.0	8.0	2.0	2453673.0	inf
100.0	10.0	2.0	4097944.0	inf
216.0	6.0	3.0	7310990.0	inf
256.0	4.0	4.0	12271366.0	inf
512.0	8.0	3.0	14501299.0	inf
1000.0	10.0	3.0	33286909.0	inf
1024.0	4.0	5.0	36227683.0	inf
1296.0	6.0	4.0	63649170.0	inf
4096.0	4.0	6.0	53961381.0	inf
4096.0	8.0	4.0	173663698.0	inf
7776.0	6.0	5.0	185158950.0	inf
10000.0	10.0	4.0	321964699.0	inf
32768.0	8.0	5.0	996128004.0	inf
46656.0	6.0	6.0	747874301.0	inf
100000.0	10.0	5.0	2603244386.0	inf
262144.0	8.0	6.0	4042473276.0	inf
1000000.0	10.0	6.0	26213449461.0	inf
100000000.0	10.0	8.0	inf	inf
429981696.0	12.0	8.0	inf	inf
1000000000.0	10.0	9.0	inf	inf
1475789056.0	14.0	8.0	inf	inf
5159780352.0	12.0	9.0	inf	inf
10000000000.0	10.0	10.0	inf	inf
20661046784.0	14.0	9.0	inf	inf
61917364224.0	12.0	10.0	inf	inf
289254654976.0	14.0	10.0	inf	inf

only assumptions. Also, bear in mind that we are not applying the rule directly from *R*, but we do it through *Python*, which means that we cannot be sure if the cause of the negative values lies within the implementation itself, the arithmetic of *Python* or any other reason.

While studying the Tables we observe that the *ECLiPSe* implementation of Dempster's Rule was unable to finish within an acceptable amount of time (or in general) when the number of combinations became $\geq 100,000,000$, or else when the number of mass functions became larger than 7 and the focal points per mass function larger than 10. We must point out that, these numbers are the same regardless of the size of Θ. It also seems that the computational time rises with a higher rate when the number of focal points grows. For example in Table 7.4 for number of mass functions equal to 6 the time it took *Prolog* to compute the mass joint for 8 focal points was 4042473276 ns and for 10 focal points 26213449461 ns that is about 6,5 times more(!). This verifies us the fact that this implementation is sensitive to the number of combinations, as $\#total\ combinations = (\#focal\ points)^{(\#mass\ functions)}$ and $(\#focal\ points)^{(\#mass\ functions)} \ll (\#focal\ points + 1)^{(\#mass\ functions)}$.

On the other hand, *ibelief*'s implementation seems to be consistent in the amount of time it takes for it to compute Demster's rule for a fixed $|\Theta|$. For example, in Table 7.1 *ibelief*'s computational time ranges from 26350989 to 35934945 ns regardless of the mass functions or the focal points. The big time difference is between different $|\Theta|$'s. If you take a look at the Tables 7.2 and 7.3 for $|\Theta|$=15, 8 focal points and 5 mass functions the computational time was 178947500 ns, whereas for $|\Theta|$=20, 8 focal points and 5 mass functions it was 4014971199 ns, which is approximately 22 times more(!). The intuition that this implementation is sensitive to the Θ size is confirmed by the fact that for $|\Theta|$=25 *ibelief* could not produce results due to the excessive amount of memory needed to allocate the mass matrices. Remember that in this implementation each mass function is of size $2^{|\Theta|} \times 1$, that makes it necessary to allocate memory for $2^{|\Theta|} \times |mass\ functions|$ and as Θ grows larger that becomes unrealistic.

To sum up, for a small amount of combinations, regardless the Universe size, *ECLiPSe*'s implementation is faster. For many possible combinations, though, *ibelief*'s implementation is significantly better, but only for small $|\Theta|$'s. For $|\Theta| \geq 25$ only *Prolog* is able to produce the combined mass function.

7.3.2 Computing belief

It has been interesting to study the behavior of the two different implementations of Dempster's rule. However, a very important part of DST is measuring belief. Thus, in this section we shall study the behavior of the two different implementations of calculating the belief function.

Before continuing, some differences between the two implementations should be highlighted:

1. *ibelief*'s implementation of belief function `mtobel` uses FMT's, while Kaltsounidis's implementation constraints.
2. *ibelief*'s `mtobel` needs an input argument that is the combined mass, which means that prior to calling `mtobel` a combination rule should be performed, otherwise the computation cannot take place. For *ECLiPSe*'s implementation that is not needed.
3. `mtobel` requires as an argument the combined mass and returns a vector where the belief values are computed for every subset of the powerset of Θ. That means, that the computational time of belief in *ibelief*'s function is the same $\forall s \in 2^{|\Theta|}$. In *ECLiPSe*'s implementation, on the other hand, the predicate `belief(+A,-V)` is used to compute the belief of a specific set A and, thus, the computational time of belief may vary between different $A \in 2^{|\Theta|}$.

Like before, in order to measure the performance of the two implementations several, similar but not identical, test cases were created where we experimented with Θ size, number of mass functions and number of focal points per mass function. Since the test cases are not exactly the same, the results of each implementations cannot be compared to one another. Specifically, the test cases were combinations of those three measures where

$|\Theta| \in \{10, 15, 20, 25\}$, number of mass functions $\in \{2, 3, \cdots 10\}$ and number of focal points per mass function $\in \{4, 6, \cdots, 14\}$. Additionally, different sets with different sizes(lengths) were used to measure the performance of the computation of their belief.

In the Tables 7.5, 7.7, 7.9 and 7.11 one can see for a fixed Θ size the time taken by CSP implementation to compute belief for some of the test cases. Respectively, in the Tables 7.6, 7.8, 7.10 and 7.12 can be seen for a fixed Θ size the time taken by *ibelief* implementation to compute belief for the similar test cases. The difference between the columns *'ibelief time(ns) bel'* and *'ibelief time(ns) bel and rule'* is that the first showcases the time it took `mtobel` to compute the belief, whereas the second showcases the time it took Dempster's rule to compute the joint mass plus the time it took `mtobel` to compute the belief, as -like we mentioned previously- in *ibelief*'s implementation it is required to perform a combination rule before computing the belief. Again, if any time column has the symbol 'inf' in its cell, it means that for these data, the implementation could not produce a result in an acceptable time frame, or could not produce a result at all due to failure of allocating the required memory.

The Figures 7.6, 7.9, 7.11, 7.13 plot the relation between the length of the set of which we computed the belief and the time required for the computation in the CSP implementation. From these, 7.6, 7.9 and 7.11 include four(4) sub-figures each. These sub-figure's have a fixed $|\Theta|$ and $|mass\ functions|$ but every one has different $|focal\ points\ per\ mass\ function|$ $\in \{4, 6, 8, 10, 14\}$. Also, the scale in the time columns may vary between each sub-figure.

Similarly, the Figures 7.7, 7.8, 7.10, 7.12 plot the relation between the length of the set of which we computed the belief and the time required for the computation in the *ibelief* implementation. From these, 7.7, 7.10 and 7.12 include four(4) sub-figures each. These sub-figure's have a fixed $|\Theta|$ and $|mass\ functio$
$ns|$ but every one has different $|focal\ points\ per\ mass\ function| \in \{4, 6, 8, 10, 14\}$. Again, the scale in the time columns may vary between each sub-figure.

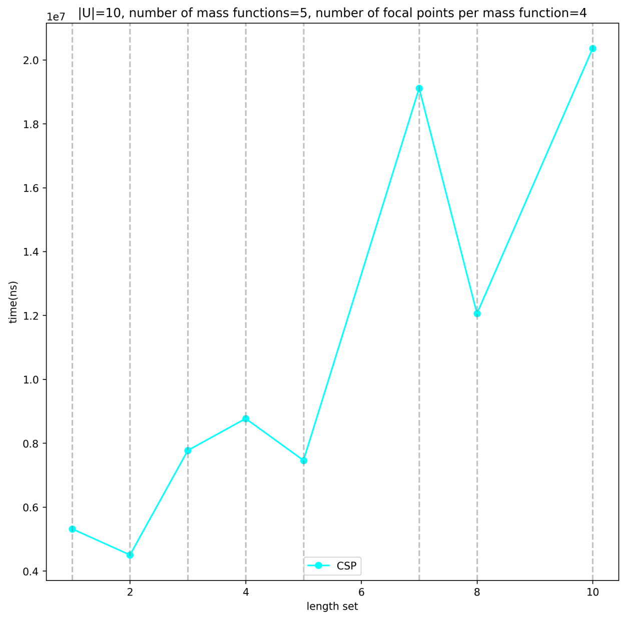
You may have noticed that we did not study the relation between the computational time and the number of combinations, as in *ibelief* the number of combinations is difficult to compute, for reasons described in the beginning of this chapter.

Table 7.5: Belief : Results for $|\Theta| = 10$, number of mass functions $\in \{2, 6, 10\}$ and number of focal points per mass function $\in \{6, 10, 14\}$ and different set-lengths, CSP Implementation

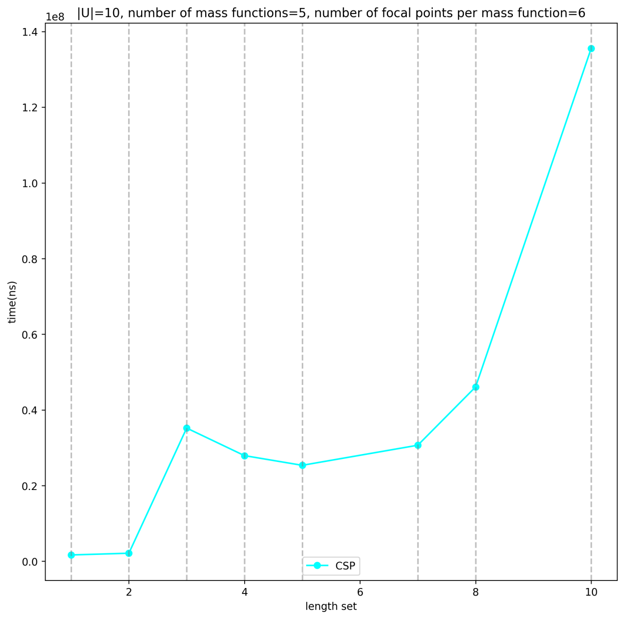
set	# focal points per mass	# mass functions	prolog time(ns)
-	10.0	10.0	inf
-	14.0	10.0	inf
[2]	10.0	2.0	188560.0
[2 3]	10.0	2.0	315721.0
[1 7 9]	10.0	2.0	440499.0
[4 5 9 10]	10.0	2.0	659600.0
[2 3 6 9 10]	10.0	2.0	687533.0
[1 2 4 5 6 7 8]	10.0	2.0	937524.0
[1 2 4 5 6 7 8 9]	10.0	2.0	1136888.0
[1 2 3 4 5 6 7 8 9 10]	10.0	2.0	2152068.0
[2]	6.0	2.0	137853.0
[2 3]	6.0	2.0	211251.0
[1 7 9]	6.0	2.0	316605.0
[4 5 9 10]	6.0	2.0	236484.0
[2 3 6 9 10]	6.0	2.0	426193.0
[1 2 4 5 6 7 8]	6.0	2.0	430492.0
[1 2 4 5 6 7 8 9]	6.0	2.0	468090.0
[1 2 3 4 5 6 7 8 9 10]	6.0	2.0	738826.0
[2]	10.0	6.0	204197620.0
[2 3]	10.0	6.0	939505504.0
[1 7 9]	10.0	6.0	1218814249.0
[4 5 9 10]	10.0	6.0	1007546797.0
[2 3 6 9 10]	10.0	6.0	2593043430.0
[1 2 4 5 6 7 8]	10.0	6.0	4122068060.0
[1 2 4 5 6 7 8 9]	10.0	6.0	4329988717.0
[1 2 3 4 5 6 7 8 9 10]	10.0	6.0	5883572280.0
[2]	6.0	6.0	22131541.0
[2 3]	6.0	6.0	46487865.0
[1 7 9]	6.0	6.0	134642355.0
[4 5 9 10]	6.0	6.0	181678904.0
[2 3 6 9 10]	6.0	6.0	168693919.0
[1 2 4 5 6 7 8]	6.0	6.0	329077877.0
[1 2 4 5 6 7 8 9]	6.0	6.0	243166054.0
[1 2 3 4 5 6 7 8 9 10]	6.0	6.0	374595878.0

Table 7.6: Belief : Results for $|\Theta| = 10$, number of mass functions $\in \{2, 6, 10\}$ and number of focal points per mass function $\in \{6, 10, 14\}$ and different set-lengths, ibelief Implementation

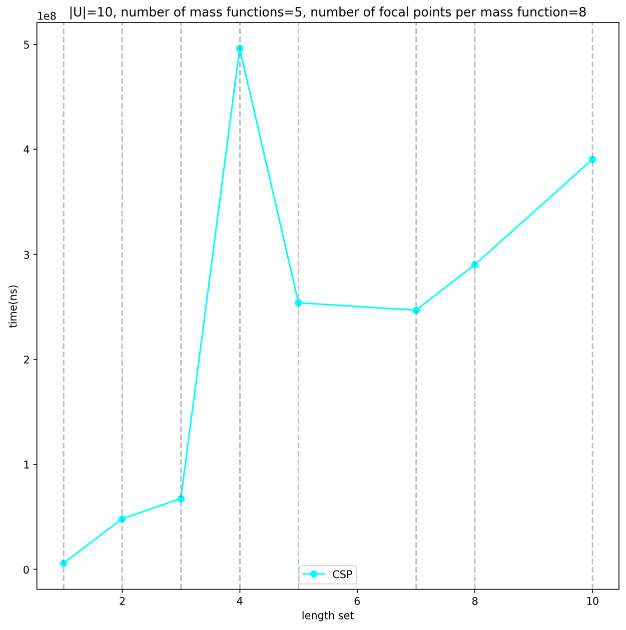
set	# focal points per mass	# mass functions	ibelief time(ns) bel	ibelief time(ns) bel and rule
-	10.0	10.0	229645.0	33329556.0
-	14.0	10.0	225728.0	32538103.0
[2]	10.0	2.0	1903993.0	29633414.0
[2 3]	10.0	2.0	1903993.0	29633414.0
[1 7 9]	10.0	2.0	1903993.0	29633414.0
[4 5 9 10]	10.0	2.0	1903993.0	29633414.0
[2 3 6 9 10]	10.0	2.0	1903993.0	29633414.0
[1 2 4 5 6 7 8]	10.0	2.0	1903993.0	29633414.0
[1 2 4 5 6 7 8 9]	10.0	2.0	1903993.0	29633414.0
[1 2 3 4 5 6 7 8 9 10]	10.0	2.0	1903993.0	29633414.0
[2]	6.0	2.0	1931024.0	28357844.0
[2 3]	6.0	2.0	1931024.0	28357844.0
[1 7 9]	6.0	2.0	1931024.0	28357844.0
[4 5 9 10]	6.0	2.0	1931024.0	28357844.0
[2 3 6 9 10]	6.0	2.0	1931024.0	28357844.0
[1 2 4 5 6 7 8]	6.0	2.0	1931024.0	28357844.0
[1 2 4 5 6 7 8 9]	6.0	2.0	1931024.0	28357844.0
[1 2 3 4 5 6 7 8 9 10]	6.0	2.0	1931024.0	28357844.0
[2]	10.0	6.0	291041.0	31343920.0
[2 3]	10.0	6.0	291041.0	31343920.0
[1 7 9]	10.0	6.0	291041.0	31343920.0
[4 5 9 10]	10.0	6.0	291041.0	31343920.0
[2 3 6 9 10]	10.0	6.0	291041.0	31343920.0
[1 2 4 5 6 7 8]	10.0	6.0	291041.0	31343920.0
[1 2 4 5 6 7 8 9]	10.0	6.0	291041.0	31343920.0
[1 2 3 4 5 6 7 8 9 10]	10.0	6.0	291041.0	31343920.0
[2]	6.0	6.0	311265.0	54626327.0
[2 3]	6.0	6.0	311265.0	54626327.0
[1 7 9]	6.0	6.0	311265.0	54626327.0
[4 5 9 10]	6.0	6.0	311265.0	54626327.0
[2 3 6 9 10]	6.0	6.0	311265.0	54626327.0
[1 2 4 5 6 7 8]	6.0	6.0	311265.0	54626327.0
[1 2 4 5 6 7 8 9]	6.0	6.0	311265.0	54626327.0
[1 2 3 4 5 6 7 8 9 10]	6.0	6.0	311265.0	54626327.0



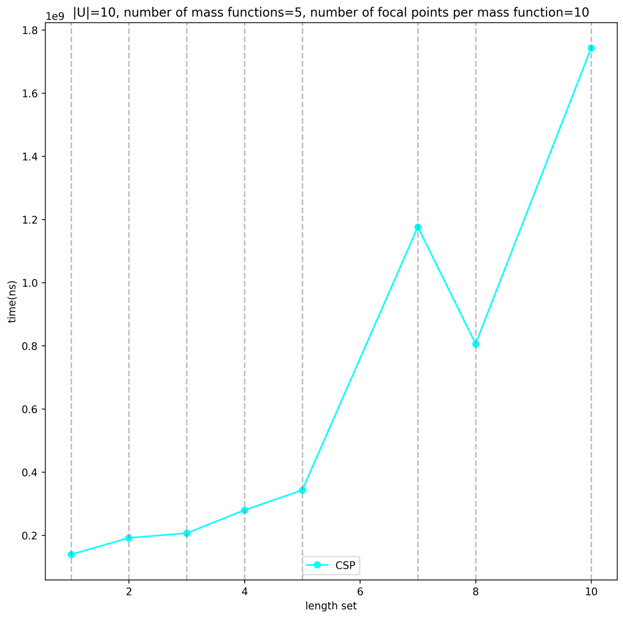
(a) number of focal points=4



(b) number of focal points=6

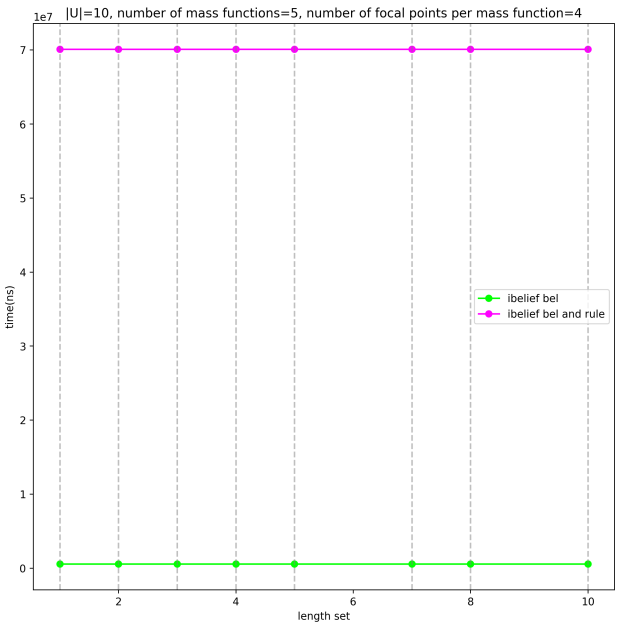


(c) number of focal points=8

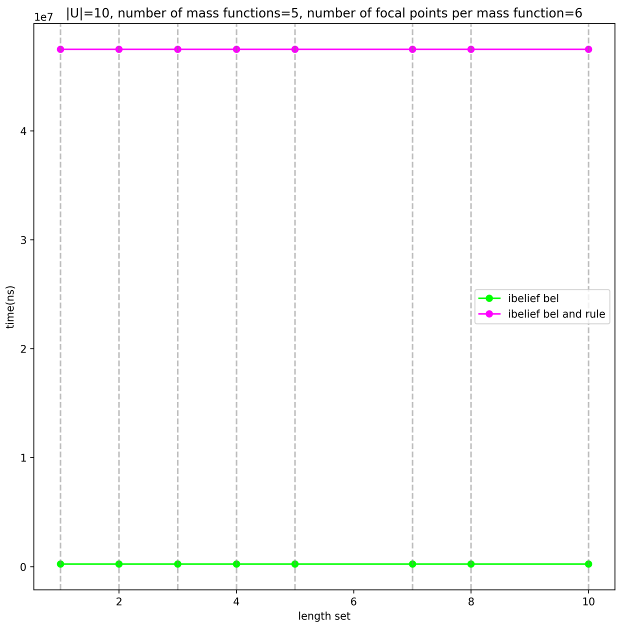


(d) number of focal points=10

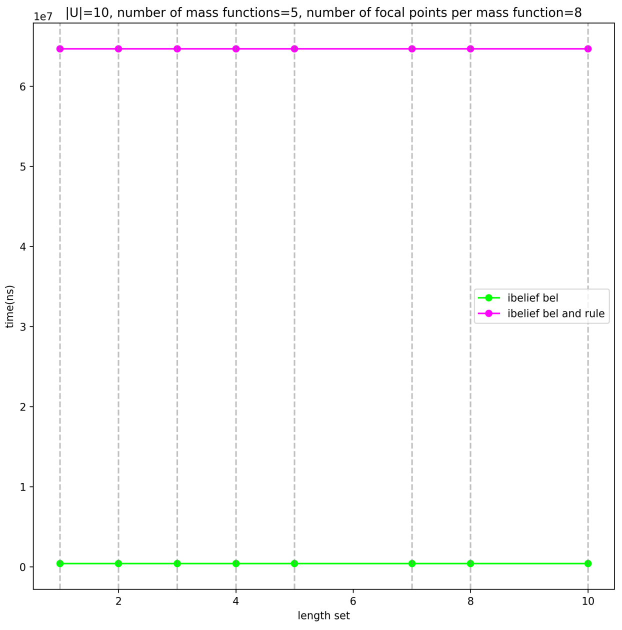
Figure 7.6: Belief : Plot - $|\Theta| = 10$ and number of mass functions=5, time(ns) and set-length, CSP Implementation



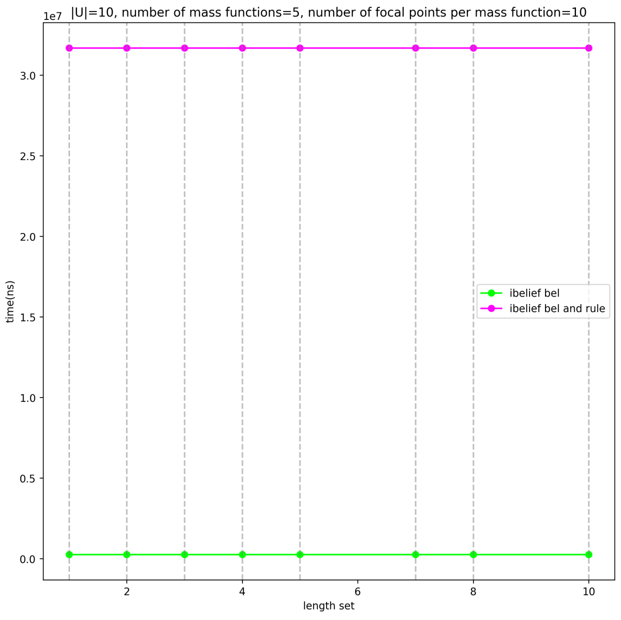
(a) number of focal points=4



(b) number of focal points=6



(c) number of focal points=8



(d) number of focal points=10

Figure 7.7: Belief : Plot - $|\Theta| = 10$ and number of mass functions=5, time(ns) and set-length, ibelief Implementation

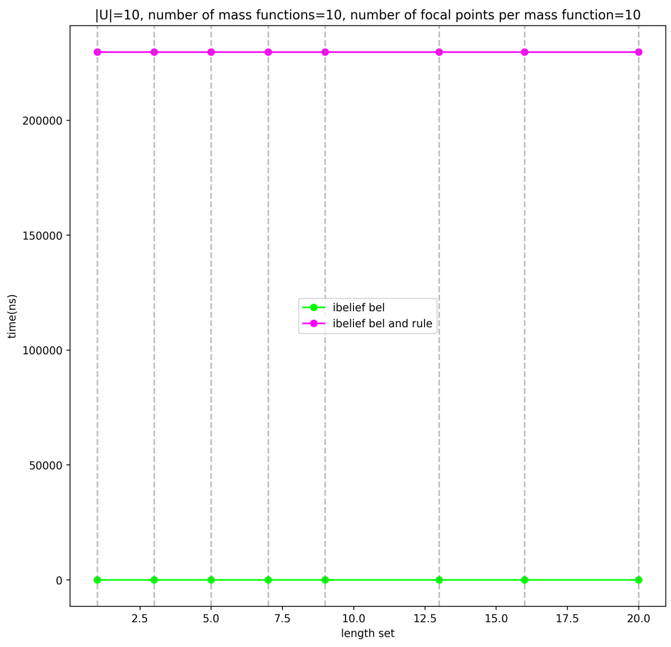


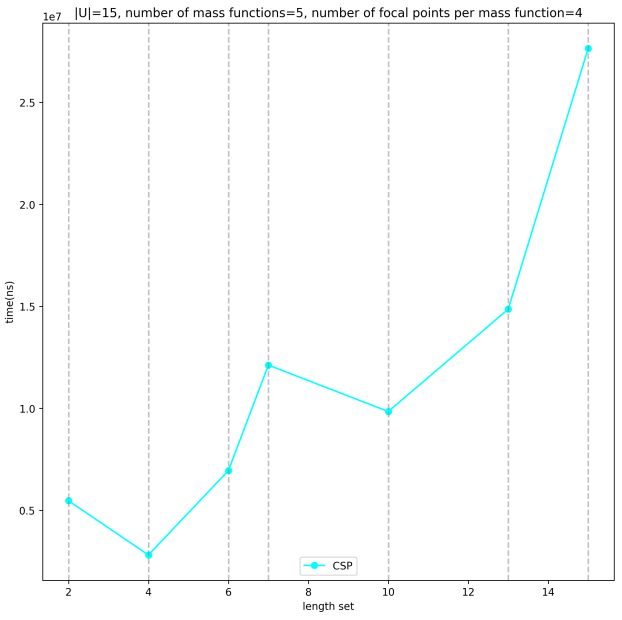
Figure 7.8: Belief : Results for $|\Theta| = 10$, number of mass functions=10 and number of focal points per mass function=10 and different set-lengths

Table 7.7: Belief : Results for $|\Theta| = 15$, number of mass functions $\in \{2, 6, 10\}$ and number of focal points per mass function $\in \{6, 10, 14\}$ and different set-lengths, CSP Implementation

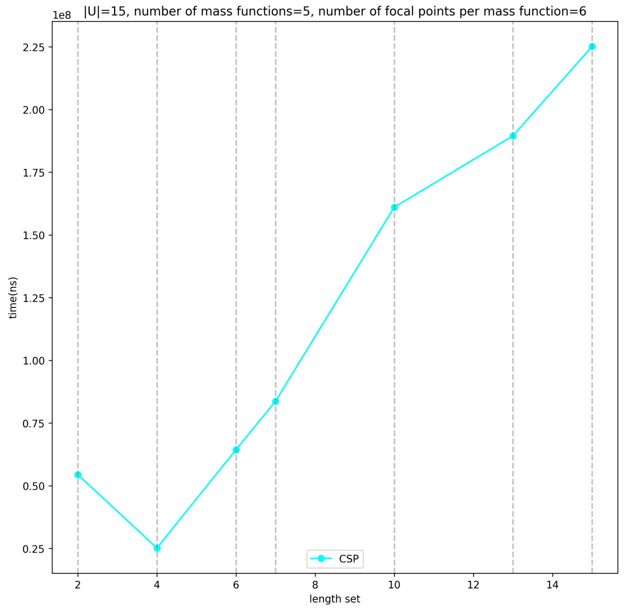
set	# focal points per mass	# mass functions	prolog time(ns)
-	10.0	10.0	inf
-	14.0	10.0	inf
[2 3]	10.0	2.0	435965.0
[1 7 9 11]	10.0	2.0	556271.0
[4 5 9 10 13 15]	10.0	2.0	580377.0
[2 3 6 9 10 11 15]	10.0	2.0	585609.0
[1 2 4 5 6 7 8 9 13 14]	10.0	2.0	1068912.0
[1 2 4 5 6 7 8 9 11 12 13 14 15]	10.0	2.0	1376826.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]	10.0	2.0	2600528.0
[2 3]	6.0	2.0	226289.0
[1 7 9 11]	6.0	2.0	385285.0
[4 5 9 10 13 15]	6.0	2.0	431471.0
[2 3 6 9 10 11 15]	6.0	2.0	392125.0
[1 2 4 5 6 7 8 9 13 14]	6.0	2.0	1957206.0
[1 2 4 5 6 7 8 9 11 12 13 14 15]	6.0	2.0	3665968.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]	6.0	2.0	1148991.0
[2 3]	10.0	6.0	2448670750.0
[1 7 9 11]	10.0	6.0	4251734577.0
[4 5 9 10 13 15]	10.0	6.0	3755364414.0
[2 3 6 9 10 11 15]	10.0	6.0	7128690003.0
[1 2 4 5 6 7 8 9 13 14]	10.0	6.0	7041457108.0
[1 2 4 5 6 7 8 9 11 12 13 14 15]	10.0	6.0	10077762547.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]	10.0	6.0	15007304845.0
[2 3]	6.0	6.0	25777044.0
[1 7 9 11]	6.0	6.0	99032047.0
[4 5 9 10 13 15]	6.0	6.0	127449988.0
[2 3 6 9 10 11 15]	6.0	6.0	135277907.0
[1 2 4 5 6 7 8 9 13 14]	6.0	6.0	325724331.0
[1 2 4 5 6 7 8 9 11 12 13 14 15]	6.0	6.0	304477618.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]	6.0	6.0	337611761.0

Table 7.8: Belief : Results for $|\Theta| = 15$, number of mass functions $\in \{2, 6, 10\}$ and number of focal points per mass function $\in \{6, 10, 14\}$ and different set-lengths, ibelief Implementation

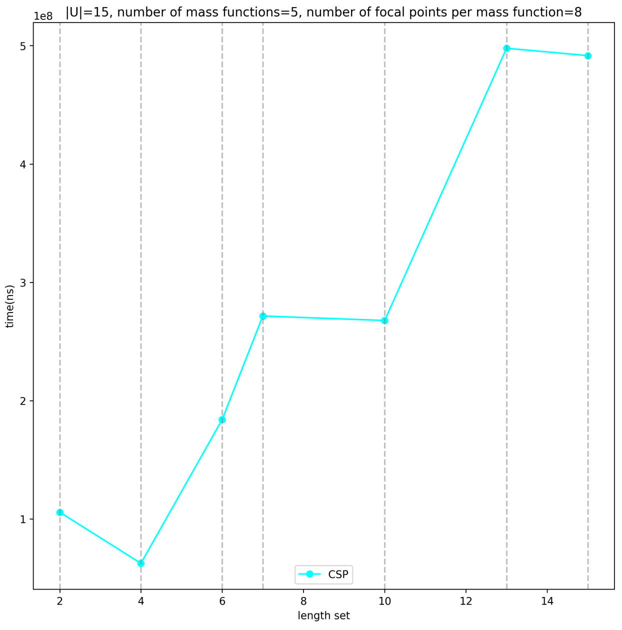
set	# focal points per mass	# mass functions	ibelief time(ns) bel	ibelief time(ns) bel and rule
-	10.0	10.0	5867578.0	232530875.0
-	14.0	10.0	7371342.0	237183145.0
[2 3]	10.0	2.0	6765085.0	159171871.0
[1 7 9 11]	10.0	2.0	6765085.0	159171871.0
[4 5 9 10 13 15]	10.0	2.0	6765085.0	159171871.0
[2 3 6 9 10 11 15]	10.0	2.0	6765085.0	159171871.0
[1 2 4 5 6 7 8 9 13 14]	10.0	2.0	6765085.0	159171871.0
[1 2 4 5 6 7 8 9 11 12 13 14 15]	10.0	2.0	6765085.0	159171871.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]	10.0	2.0	6765085.0	159171871.0
[2 3]	6.0	2.0	8388635.0	141476347.0
[1 7 9 11]	6.0	2.0	8388635.0	141476347.0
[4 5 9 10 13 15]	6.0	2.0	8388635.0	141476347.0
[2 3 6 9 10 11 15]	6.0	2.0	8388635.0	141476347.0
[1 2 4 5 6 7 8 9 13 14]	6.0	2.0	8388635.0	141476347.0
[1 2 4 5 6 7 8 9 11 12 13 14 15]	6.0	2.0	8388635.0	141476347.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]	6.0	2.0	8388635.0	141476347.0
[2 3]	10.0	6.0	5626713.0	200412473.0
[1 7 9 11]	10.0	6.0	5626713.0	200412473.0
[4 5 9 10 13 15]	10.0	6.0	5626713.0	200412473.0
[2 3 6 9 10 11 15]	10.0	6.0	5626713.0	200412473.0
[1 2 4 5 6 7 8 9 13 14]	10.0	6.0	5626713.0	200412473.0
[1 2 4 5 6 7 8 9 11 12 13 14 15]	10.0	6.0	5626713.0	200412473.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]	10.0	6.0	5626713.0	200412473.0
[2 3]	6.0	6.0	4906463.0	222229092.0
[1 7 9 11]	6.0	6.0	4906463.0	222229092.0
[4 5 9 10 13 15]	6.0	6.0	4906463.0	222229092.0
[2 3 6 9 10 11 15]	6.0	6.0	4906463.0	222229092.0
[1 2 4 5 6 7 8 9 13 14]	6.0	6.0	4906463.0	222229092.0
[1 2 4 5 6 7 8 9 11 12 13 14 15]	6.0	6.0	4906463.0	222229092.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]	6.0	6.0	4906463.0	222229092.0



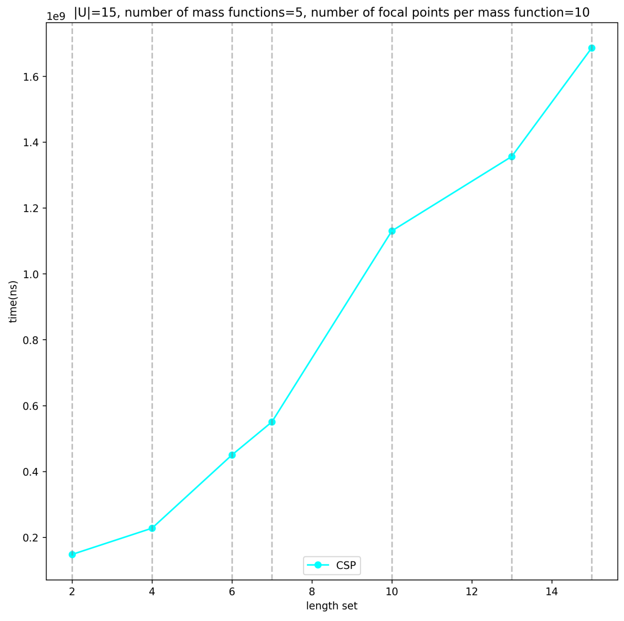
(a) number of focal points=4



(b) number of focal points=6

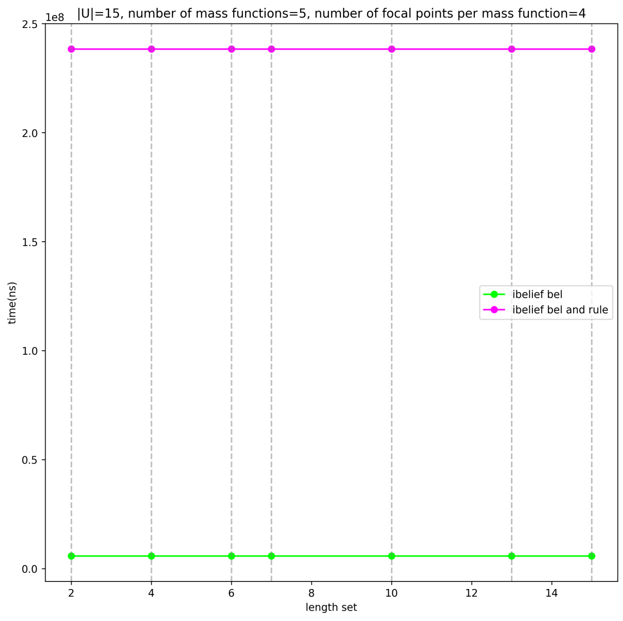


(c) number of focal points=8

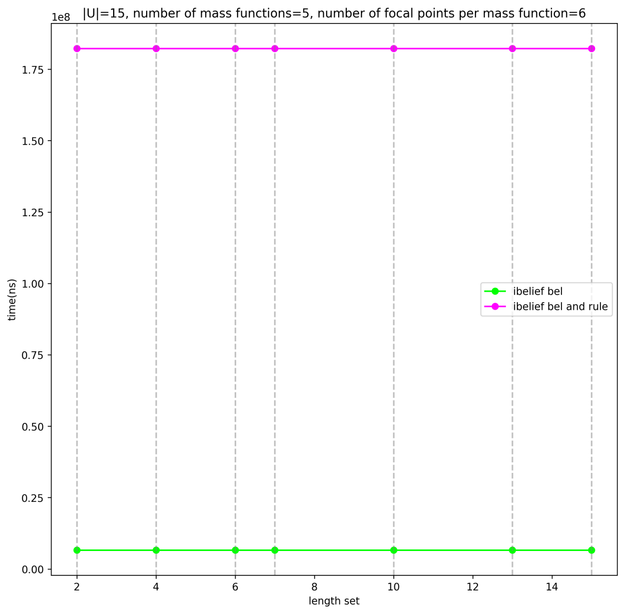


(d) number of focal points=10

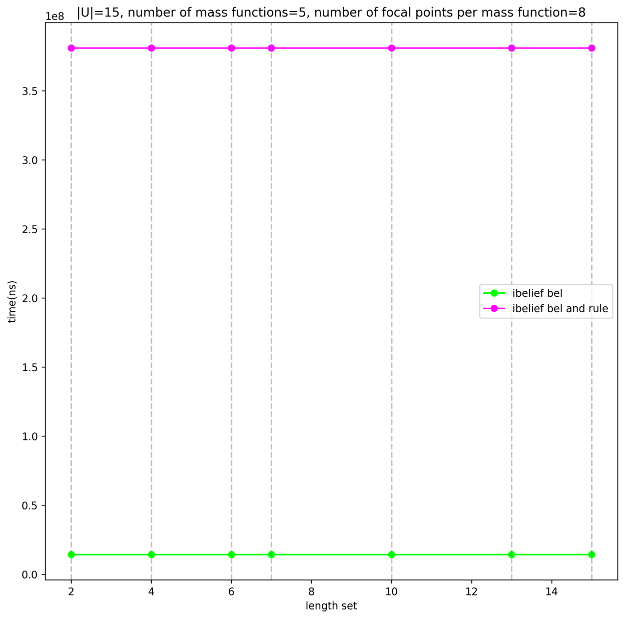
Figure 7.9: Belief : Plot - $|\Theta| = 15$ and number of mass functions=5, time(ns) and set-length, CSP Implementation



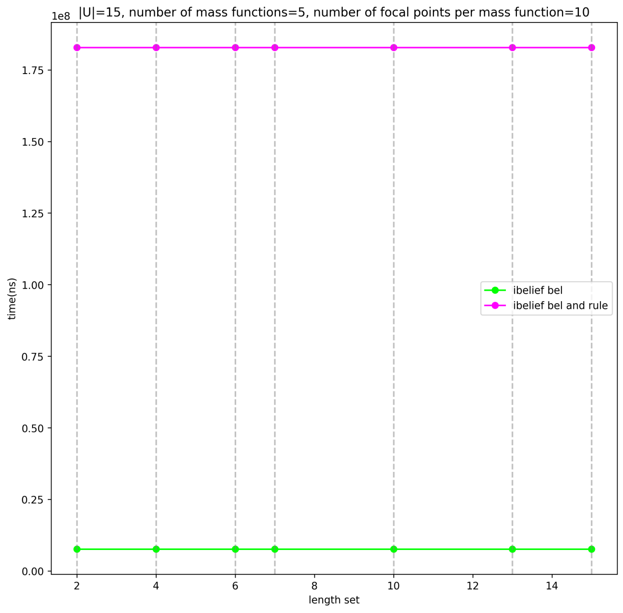
(a) number of focal points=4



(b) number of focal points=6



(c) number of focal points=8



(d) number of focal points=10

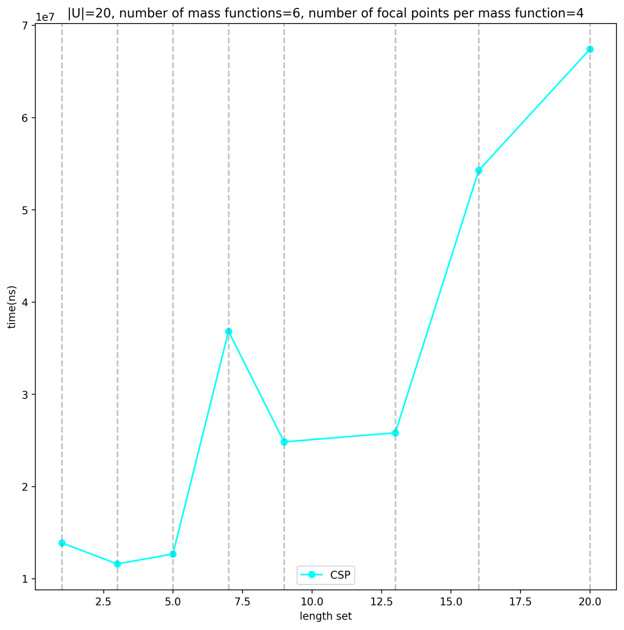
Figure 7.10: Belief : Plot - $|\Theta| = 15$ and number of mass functions=5, time(ns) and set-length, ibelief Implementation

Table 7.9: Belief : Results for $|\Theta| = 20$, number of mass functions $\in \{2, 6, 10\}$ and number of focal points per mass function $\in \{6, 10, 14\}$ and different set-lengths, CSP Implementation

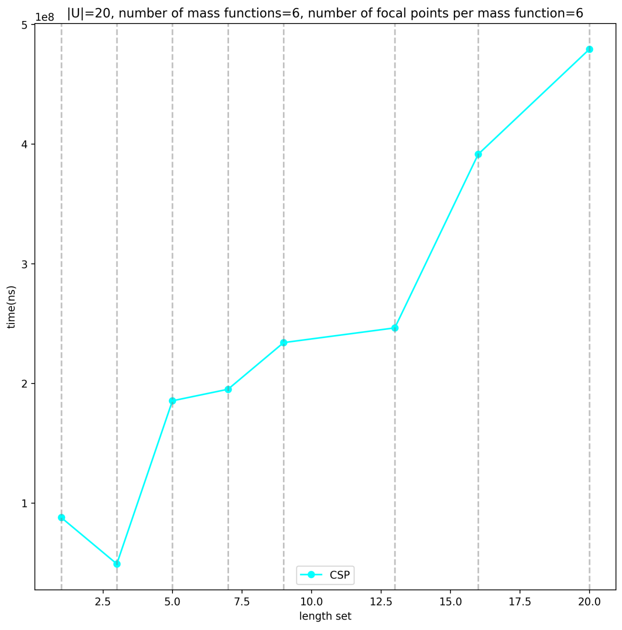
set	# focal points per mass	# mass functions	prolog time(ns)
-	10.0	10.0	inf
-	14.0	10.0	inf
[11]	10.0	2.0	339762.0
[2 3 5]	10.0	2.0	453052.0
[1 7 9 12 14]	10.0	2.0	589446.0
[4 5 9 10 11 13 15]	10.0	2.0	978985.0
[2 3 6 9 10 12 16 17 18]	10.0	2.0	997195.0
[1 2 4 5 6 7 8 9 10 12 16 17 19]	10.0	2.0	1471476.0
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18]	10.0	2.0	1835805.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]	10.0	2.0	3176732.0
[11]	6.0	2.0	196277.0
[2 3 5]	6.0	2.0	251878.0
[1 7 9 12 14]	6.0	2.0	256056.0
[4 5 9 10 11 13 15]	6.0	2.0	499904.0
[2 3 6 9 10 12 16 17 18]	6.0	2.0	375263.0
[1 2 4 5 6 7 8 9 10 12 16 17 19]	6.0	2.0	356064.0
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18]	6.0	2.0	634572.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]	6.0	2.0	1163379.0
[11]	10.0	6.0	352850196.0
[2 3 5]	10.0	6.0	2463948525.0
[1 7 9 12 14]	10.0	6.0	4200157353.0
[4 5 9 10 11 13 15]	10.0	6.0	4731250464.0
[2 3 6 9 10 12 16 17 18]	10.0	6.0	5969690349.0
[1 2 4 5 6 7 8 9 10 12 16 17 19]	10.0	6.0	7784238489.0
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18]	10.0	6.0	12498186673.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]	10.0	6.0	17157224439.0
[11]	6.0	6.0	87833959.0
[2 3 5]	6.0	6.0	49053790.0
[1 7 9 12 14]	6.0	6.0	185377768.0
[4 5 9 10 11 13 15]	6.0	6.0	195026731.0
[2 3 6 9 10 12 16 17 18]	6.0	6.0	234010228.0
[1 2 4 5 6 7 8 9 10 12 16 17 19]	6.0	6.0	246320784.0
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18]	6.0	6.0	391411848.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]	6.0	6.0	479217444.0

Table 7.10: Belief : Results for $|\Theta| = 20$, number of mass functions $\in \{2, 6, 10\}$ and number of focal points per mass function $\in \{6, 10, 14\}$ and different set-lengths, ibelief Implementation

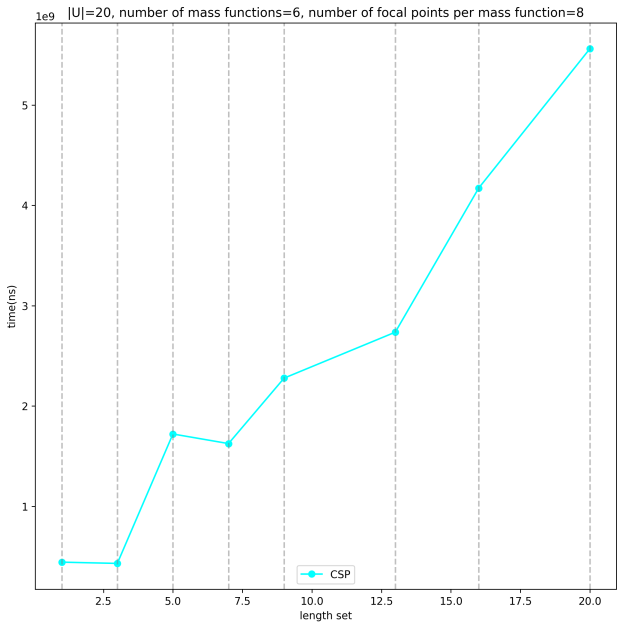
set	# focal points per mass	# mass functions	ibelief time(ns) bel	ibelief time(ns) bel and rule
-	10.0	10.0	350259919.0	6171708669.0
-	14.0	10.0	351484739.0	6196369091.0
[11]	10.0	2.0	365276629.0	3671799192.0
[2 3 5]	10.0	2.0	365276629.0	3671799192.0
[1 7 9 12 14]	10.0	2.0	365276629.0	3671799192.0
[4 5 9 10 11 13 15]	10.0	2.0	365276629.0	3671799192.0
[2 3 6 9 10 12 16 17 18]	10.0	2.0	365276629.0	3671799192.0
[1 2 4 5 6 7 8 9 10 12 16 17 19]	10.0	2.0	365276629.0	3671799192.0
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18]	10.0	2.0	365276629.0	3671799192.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]	10.0	2.0	365276629.0	3671799192.0
[11]	6.0	2.0	363140204.0	3646258854.0
[2 3 5]	6.0	2.0	363140204.0	3646258854.0
[1 7 9 12 14]	6.0	2.0	363140204.0	3646258854.0
[4 5 9 10 11 13 15]	6.0	2.0	363140204.0	3646258854.0
[2 3 6 9 10 12 16 17 18]	6.0	2.0	363140204.0	3646258854.0
[1 2 4 5 6 7 8 9 10 12 16 17 19]	6.0	2.0	363140204.0	3646258854.0
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18]	6.0	2.0	363140204.0	3646258854.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]	6.0	2.0	363140204.0	3646258854.0
[11]	10.0	6.0	375933276.0	4860138713.0
[2 3 5]	10.0	6.0	375933276.0	4860138713.0
[1 7 9 12 14]	10.0	6.0	375933276.0	4860138713.0
[4 5 9 10 11 13 15]	10.0	6.0	375933276.0	4860138713.0
[2 3 6 9 10 12 16 17 18]	10.0	6.0	375933276.0	4860138713.0
[1 2 4 5 6 7 8 9 10 12 16 17 19]	10.0	6.0	375933276.0	4860138713.0
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18]	10.0	6.0	375933276.0	4860138713.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]	10.0	6.0	375933276.0	4860138713.0
[11]	6.0	6.0	383512103.0	4801744301.0
[2 3 5]	6.0	6.0	383512103.0	4801744301.0
[1 7 9 12 14]	6.0	6.0	383512103.0	4801744301.0
[4 5 9 10 11 13 15]	6.0	6.0	383512103.0	4801744301.0
[2 3 6 9 10 12 16 17 18]	6.0	6.0	383512103.0	4801744301.0
[1 2 4 5 6 7 8 9 10 12 16 17 19]	6.0	6.0	383512103.0	4801744301.0
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18]	6.0	6.0	383512103.0	4801744301.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]	6.0	6.0	383512103.0	4801744301.0



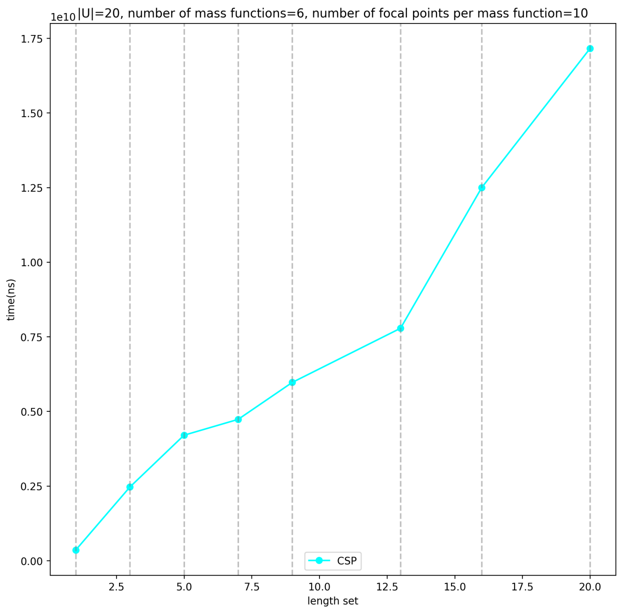
(a) number of focal points=4



(b) number of focal points=6

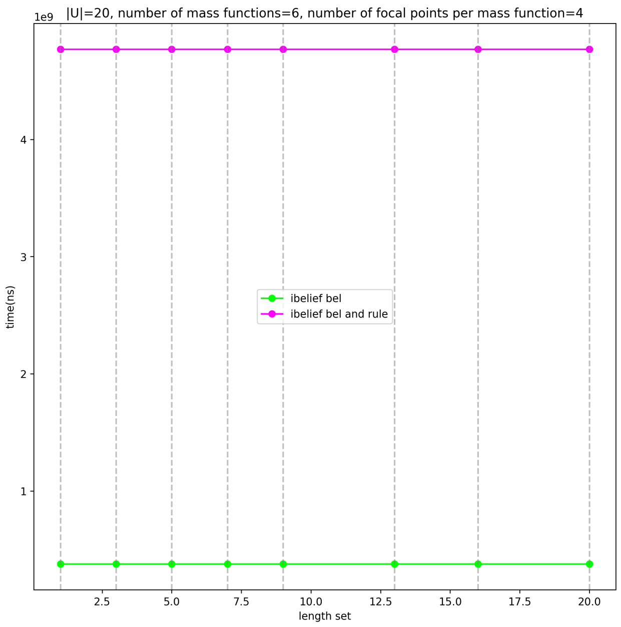


(c) number of focal points=8

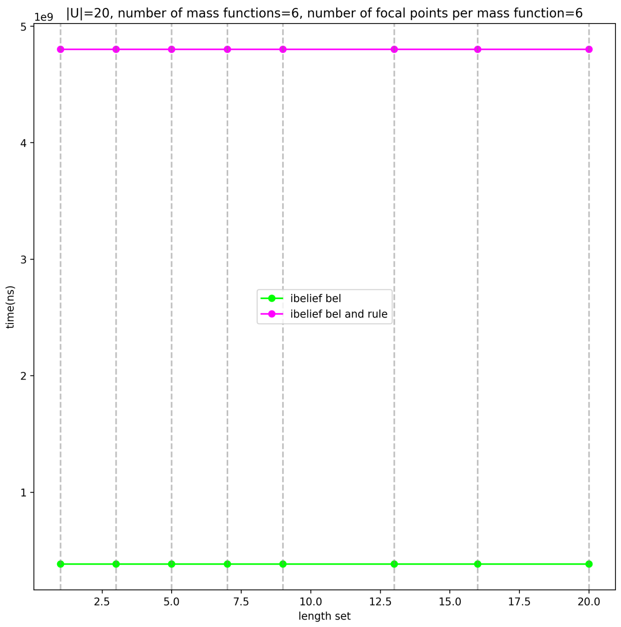


(d) number of focal points=10

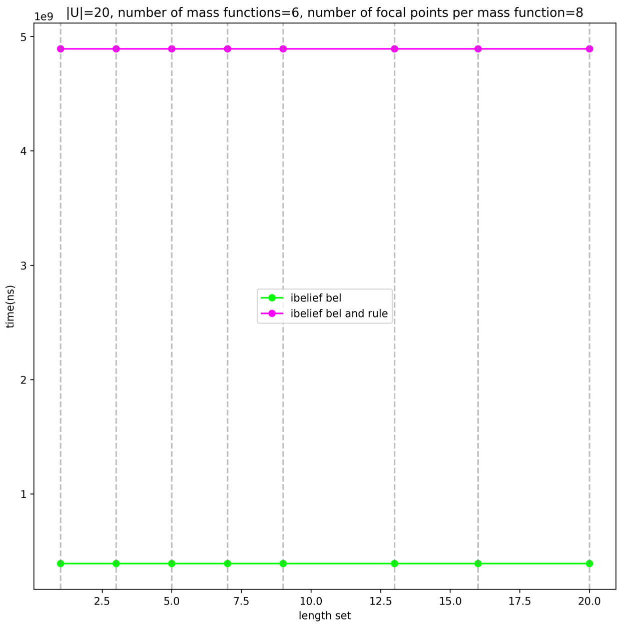
Figure 7.11: Belief : Plot - $|\Theta| = 20$ and number of mass functions=6, time(ns) and set-length, CSP Implementation



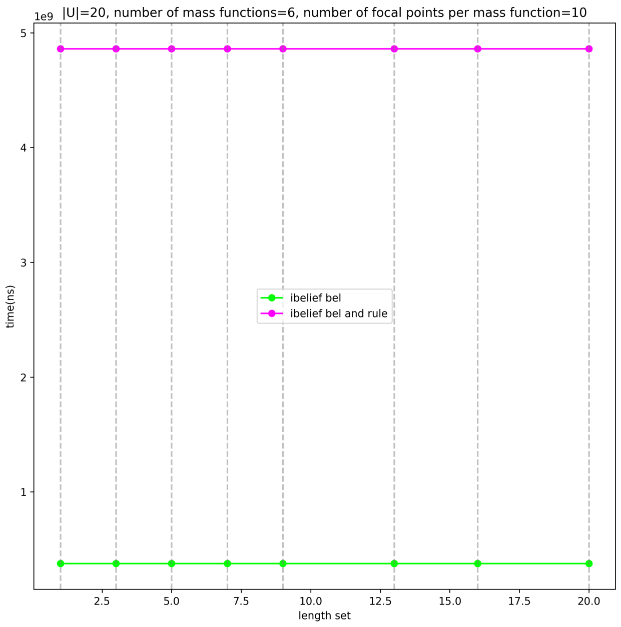
(a) number of focal points=4



(b) number of focal points=6



(c) number of focal points=8



(d) number of focal points=10

Figure 7.12: Belief : Plot - $|\Theta| = 20$ and number of mass functions=6, time(ns) and set-length, ibelief Implementation

Table 7.11: Belief : Results for $|\Theta| = 25$, number of mass functions $\in \{2, 6, 10\}$ and number of focal points per mass function $\in \{6, 10, 14\}$ and different set-lengths, CSP Implementation

set	# focal points per mass	# mass functions	prolog time(ns)
-	10.0	10.0	inf
-	14.0	10.0	inf
[11]	10.0	2.0	818320.0
[2 3 5]	10.0	2.0	475385.0
[1 7 9 12 14 16 17 18]	10.0	2.0	505819.0
[4 5 9 10 11 13 15 16 17 18]	10.0	2.0	1050615.0
[2 3 6 9 10 12 16 17 18 19 20 21]	10.0	2.0	596812.0
[1 2 4 5 6 7 8 9 10 12 16 17 19 22 23 24]	10.0	2.0	2721990.0
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18 20 21 22 23 24]	10.0	2.0	1975667.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]	10.0	2.0	4164803.0
[11]	6.0	2.0	272901.0
[2 3 5]	6.0	2.0	331663.0
[1 7 9 12 14 16 17 18]	6.0	2.0	304766.0
[4 5 9 10 11 13 15 16 17 18]	6.0	2.0	552304.0
[2 3 6 9 10 12 16 17 18 19 20 21]	6.0	2.0	356236.0
[1 2 4 5 6 7 8 9 10 12 16 17 19 22 23 24]	6.0	2.0	510568.0
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18 20 21 22 23 24]	6.0	2.0	913778.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]	6.0	2.0	1498549.0
[11]	10.0	6.0	530231193.0
[2 3 5]	10.0	6.0	1739800923.0
[1 7 9 12 14 16 17 18]	10.0	6.0	4542183258.0
[4 5 9 10 11 13 15 16 17 18]	10.0	6.0	4972265929.0
[2 3 6 9 10 12 16 17 18 19 20 21]	10.0	6.0	8126537570.0
[1 2 4 5 6 7 8 9 10 12 16 17 19 22 23 24]	10.0	6.0	8751664178.0
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18 20 21 22 23 24]	10.0	6.0	12149414893.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]	10.0	6.0	15831163790.0
[11]	6.0	6.0	22312188.0
[2 3 5]	6.0	6.0	77384597.0
[1 7 9 12 14 16 17 18]	6.0	6.0	474763428.0
[4 5 9 10 11 13 15 16 17 18]	6.0	6.0	383199830.0
[2 3 6 9 10 12 16 17 18 19 20 21]	6.0	6.0	480587642.0
[1 2 4 5 6 7 8 9 10 12 16 17 19 22 23 24]	6.0	6.0	688360883.0
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18 20 21 22 23 24]	6.0	6.0	705003298.0
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]	6.0	6.0	1145708957.0

Table 7.12: Belief : Results for $|\Theta| = 25$, number of mass functions $\in \{2, 6, 10\}$ and number of focal points per mass function $\in \{6, 10, 14\}$ and different set-lengths, ibelief Implementation

set	# focal points per mass	# mass functions	ibelief time(ns) bel	ibelief time(ns) bel and rule
-	10.0	10.0	inf	inf
-	14.0	10.0	inf	inf
[11]	10.0	2.0	inf	inf
[2 3 5]	10.0	2.0	inf	inf
[1 7 9 12 14 16 17 18]	10.0	2.0	inf	inf
[4 5 9 10 11 13 15 16 17 18]	10.0	2.0	inf	inf
[2 3 6 9 10 12 16 17 18 19 20 21]	10.0	2.0	inf	inf
[1 2 4 5 6 7 8 9 10 12 16 17 19 22 23 24]	10.0	2.0	inf	inf
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18 20 21 22 23 24]	10.0	2.0	inf	inf
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]	10.0	2.0	inf	inf
[11]	6.0	2.0	inf	inf
[2 3 5]	6.0	2.0	inf	inf
[1 7 9 12 14 16 17 18]	6.0	2.0	inf	inf
[4 5 9 10 11 13 15 16 17 18]	6.0	2.0	inf	inf
[2 3 6 9 10 12 16 17 18 19 20 21]	6.0	2.0	inf	inf
[1 2 4 5 6 7 8 9 10 12 16 17 19 22 23 24]	6.0	2.0	inf	inf
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18 20 21 22 23 24]	6.0	2.0	inf	inf
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]	6.0	2.0	inf	inf
[11]	10.0	6.0	inf	inf
[2 3 5]	10.0	6.0	inf	inf
[1 7 9 12 14 16 17 18]	10.0	6.0	inf	inf
[4 5 9 10 11 13 15 16 17 18]	10.0	6.0	inf	inf
[2 3 6 9 10 12 16 17 18 19 20 21]	10.0	6.0	inf	inf
[1 2 4 5 6 7 8 9 10 12 16 17 19 22 23 24]	10.0	6.0	inf	inf
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18 20 21 22 23 24]	10.0	6.0	inf	inf
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]	10.0	6.0	inf	inf
[11]	6.0	6.0	inf	inf
[2 3 5]	6.0	6.0	inf	inf
[1 7 9 12 14 16 17 18]	6.0	6.0	inf	inf
[4 5 9 10 11 13 15 16 17 18]	6.0	6.0	inf	inf
[2 3 6 9 10 12 16 17 18 19 20 21]	6.0	6.0	inf	inf
[1 2 4 5 6 7 8 9 10 12 16 17 19 22 23 24]	6.0	6.0	inf	inf
[1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 18 20 21 22 23 24]	6.0	6.0	inf	inf
[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]	6.0	6.0	inf	inf

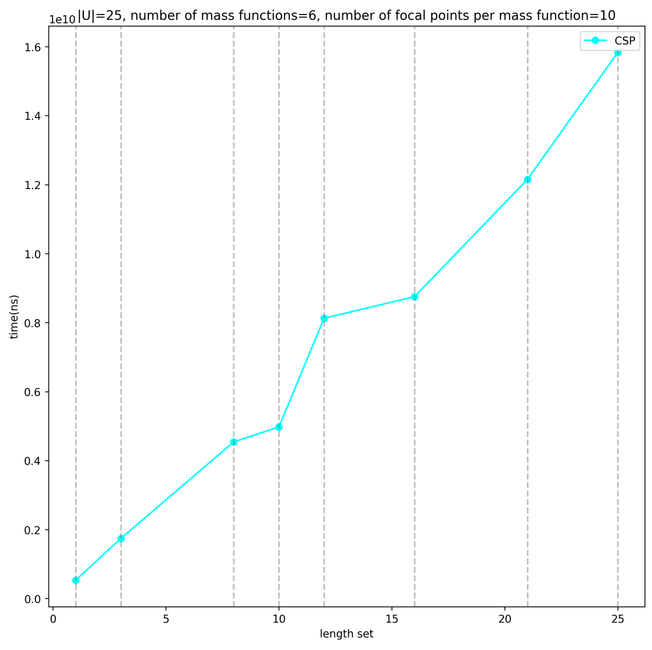


Figure 7.13: Belief : Plot - $|\Theta| = 25$ and number of mass functions=6, time(ns) and set-length

No differently than the Dempster's rule performance, when computing belief, *ECLiPSe*'s implementation does not perform well when the number of possible combinations becomes large. This is expected, as both predicates' functionality `belief/2` and `bpa/3` are based on the same predicates as stated when introducing the predicates of this implementation. What cannot be seen from Tables 7.5, 7.7, 7.9 and 7.11 is that `belief(+A,-V)` may produce results for a number of mass functions larger than 6 and number of focal points larger than 10 when `A`'s length is less than 6, but the computation is so slow, that it is not worth mentioning. For that reason, Figure 7.8 showcases only the performance of *ibelief*'s implementation. Similar figures could be plotted for other Θ sizes, but they would not have been different. The performance of *ibelief* in test cases that *ECLiPSe* cannot handle can be seen through the previously mentioned tables.

In the other cases, where *Prolog* does produce results, we notice that for fixed $|\Theta|$ sizes, number of mass functions and focal points the computational time of belief does not necessary grow when asking for the belief of a set that contains more elements. The Figures 7.6, 7.9, 7.11, 7.13 are a proof of that, as we can see that the line is not strictly ascending. Despite this fact, the line is still ascending, which means that, generally, for larger sets the computation of their belief with this method is more time consuming. Also, as the $|\Theta|$ size enlarges, the computation of the belief of a specific set takes more time, but not significantly. For example, in Table 7.5 for the set $\{2, 3\}$ and 2 mass functions with 10 focal points each, *Prolog* computed the belief in 315721 ns. For the same data, in Table 7.7 the belief is computed in 435965 ns, that is just 1.3 times more.

We already explained that *ibelief*'s implementation computes the belief for every subset of the Universe's powerset and for that reason, the computational time depends solely on the dataset and not on the set whose belief we want to compute. Here as well, the factor that affects the performance of `mtobel` is the size of the Universe and neither the number of mass functions, nor the focal points per mass function. This fact can be observed in both columns that refer to *ibelief*'s time in Tables 7.6, 7.8, 7.10 and 7.12. Since computing the mass joint is essential to compute the belief in this implementation, it is self-explanatory why this implementation does not produce results due to the inability of allocating the needed memory for large $|\Theta|$, as seen in Table 7.12 and Figure 7.13.

As we mentioned, there cannot be a comparison between the performance of the two implementations, as the test cases may differ, but some general remarks can be made. For example, for a smaller amount of focal points, and thus combinations, *ECLiPSe* produces results quick, but when the number of focal points is large, the computational time ascends. Also, for smaller sets *Prolog* computes their belief quite fast. On the other hand, *ibelief* computes quickly the belief for small Universes, but does not perform well at all for larger ones and the main problem for that is that it is not able to allocate the needed memory for the mass matrix.

Previously, we wrote that `mtobel` produces a vector of size $1 \times 2^{|\Theta|}$ where the belief is stored $\forall s \in 2^{|\Theta|}$. In that vector, the belief value for each s is stored regarding the sets size and elements, which means that the sets are sorted first by size and then lexicographically. So what happens if we want to compute the belief for a set $s = \{1, 2\}$ through *ibelief*? Let's suppose that we have $\Theta = \{1, 2, 3\}$. First of all, we compute the mass joint, then call the function `mtobel` that returns the vector v and then we have to find the index of s in v. We compute the index of s (which is 5) and go to $v[5]$ in order to find the belief. So, after computing the belief, there is an overhead of finding the index of the set we are interested in, in order to retrieve it's belief. For large Θ's this overhead is not minor and for sure makes *ibelief*'s implementation not user friendly.

8. CONCLUSIONS AND FUTURE WORK

Dempster's rule of combination and the computation of belief are the two backbones of Dempster-Shafer theory but are very costly in terms of computational time. Kaltsounidis in his thesis developed an implementation based on CLP in order to try and reduce this computational cost. Through several randomly generated test cases he observed a reduced run time. Continuing his work we observed the behavior of his implementation in an application of DST and specifically using a real-life dataset. The problem we faced was that the pre-processing of the dataset used in order to compute the mass functions as defined by the application took long time as the fraction of the dataset we used increased. Thus, we could not measure the performance of the implementation for Θ sizes larger that 400 and number of focal points larger that 160. For the fraction of the dataset we used, though, the *ECLiPSe Prolog* implementation computed the mass joint and the belief within an acceptable time range, with the most costly computation taking 17.5 seconds.

Our next step was to compare Kaltsounidis' implementation to the *ibelief*'s one. Unfortunately, that could not be done in the dataset of application because of the way FMT's implementation takes its input. So, the comparison was made with artificial test cases. It should be noted, also, that we could not compare the two implementations regarding the computation of belief, as we did not ensure identical test cases. The resolution is that *ibelief*'s implementation generally produces correct results faster for Θ sizes smaller than 25, when computing Dempster's rule. For larger Θ's it cannot produce results at all. Also, the CLP implementation may produce the results in less time if the total number of combinations is small, when talking about the combination rule. For a high amount of focal points and mass functions this implementation is unable to compute the mass joint.

For the belief function computation, CLP could not produce results for a high number of combinations, but had a steady computational time within different Θ sizes for fixed number of mass functions and number of focal points. As for *ibelief*'s implementation, it was unable to produce results for large Θ sizes but had quite similar computational times for different number of mass functions and number of focal points within the same Θ. So, both implementations have their pros and cons and one should take into consideration the structure of the dataset before choosing one of the two implementations.

Decisively, *ibelief*'s implementation is not very user-friendly. First of all, the input of the combination rule requires the mass for each subset of Universe's powerset for every mass function, which is computationally heavy for real-life applications. Another problem that can be observed when using this implementation, is that to combine the mass functions and produce the mass joint, a matrix of size $2^{|\Theta|} \times |mass\ functions|$ has to be allocated. When the Θ becomes large, that matrix cannot be allocated and the implementation cannot produce results. If that matrix was not to be allocated and another data structure was to be used, probably such problem was not to be faced and *ibelief* could produce results. In that case, we could measure the performance of FMT's in test cases with large Universes and not deal with the limitations of the data structure itself.

Generally, it would be interesting to observe the behavior of Kaltsounidis’ implementation on different datasets, with more mass functions and reduced pre-processing time. Also, a comparison of this implementation with another on a real-life application would help us come to the conclusion if the implementation is applicable and if Dempster’s rule can be used as a combination of degrees of belief. Additionally, a compelling continuation of this Thesis would be to ensure identical test cases in both CSP’s and *ibelief*’s implementations in order to compute the belief and compare their performance. Lastly, we mentioned that, usually, time approximation methods are used in order to combine mass functions and compute belief. An interesting work would be to compare these methods against each other and against CLP’s implementation and FMT’s implementation.

ABBREVIATIONS - ACRONYMS

DST	Dempster-Shafer theory
BPA	Basic Probability Assignment
TBM	Transferable Belief Model
FMT	Fast Möbius Transform
CSP	Constraint Satisfaction Problem
CP	Constraint Programming
CLP	Constraint Logic Programming
CRAN	The Comprehensive R Archive Network
RS	Recommender Systems
DFD	Data Flow Diagram
AI	Artificial Intelligence
ML	Machine Learning

ANNEX I

PyCLP Package Installation

Installation source: https://sourceforge.net/p/pyclp/

In this appendix the installation process of the *PyCLP* package will be discussed as it had some complexities. This discussion will hopefully be useful for anyone who will be trying to install this package in the future.

The *PyCLP* package is an interface to *ECLiPSe* Constraint Programming System. This package was installed in an *Ubuntu 18.04.5 LTS* machine. Personally, I found it easier to try and install the package via command line, but that could be done as well in a *PyCharm* project.

Let's run through the pre-installation requirements: A *Python 2.x or 3.x version should be installed, as well as the Cython package and, of course, the ECLiPSe Constraint Programming System, but the 6.1 version. Although the requirements mention that the library is supported through a Python 3.x version, I was able to install it only when Python 2.7 was used as with other version's the Cython's module could not be found.*

Because Python 2.7 was used, rpy2 2.8.6 was installed.

If one has multiple Python versions installed in their machine, they can choose with `sudo update-alternatives --config python` *which one will be used.*

Another problem that appeared was that with Python 3.6 and 3.7 the compilation process with Cython (both on gcc and msvc) had given the error:

```
src\pyclp\pyclp.pyx:753:40: Cannot convert Python object to 'pword'
```

This error was fixed by manually replacing the following block of code in the file pyclp.pyx,

```
if len(tail) == 0:
    tail_pword=pyclp.ec_nil()
else:
    tail_pword=PList(tail)
```

with the block,

```
if len(tail) == 0:
    tail_pword=pyclp.ec_nil()
else:
    tail_pword=PList(tail).get_pword()
```

However, even though the installation was only possible when Python 2.7 was used, but when the program was run, then it worked perfectly well with Python 3.10.

We must highlight again that the needed ECLiPSe version is 6.1 and not the latest, 7, otherwise a hand-full of errors will occur.

It should be noted, also that in `setup.py` I had to replace all relative paths with absolute ones.

The last obstacle before being able to run the project was that the import `from pyclp import *` failed and the following error was produced

```
ImportError: libeclipse.so: cannot open shared object file: No such file or
    directory
```

From the error it was easy to understand that the linker was not able to find the location of the file libecliplse.so. In order to solve this the environment variable LD_LIBRARY_PATH was added in the /etc/environment file since,

```
LD_LIBRARY_PATH is the search path environment variable for the
linux shared library

In Linux, the environment variable LD_LIBRARY_PATH is a colon-
separated (:) set of directories where libraries are
searched for first before the standard set of directories.
```

REFERENCES

[1] Belief function implementation. https://cran.r-project.org/web/packages/ibelief/index.html. Accessed: 2022-01-01.

[2] The eclipse constraint programming system. https://eclipseclp.org/. Accessed: 2022-01-01.

[3] Pandas open source data analysis and manipulation tool. https://pandas.pydata.org/. Accessed: 2022-02-10.

[4] Python interface to the r language (embedded r). https://pypi.org/project/rpy2/. Accessed: 2022-01-01.

[5] Python library to interface eclipse constraint system. https://sourceforge.net/projects/pyclp/. Accessed: 2022-01-01.

[6] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[7] Maxime Chaveroche, Franck Davoine, and Véronique Cherfaoui. Efficient möbius transformations and their applications to dempster-shafer theory: Clarification and implementation, 2021.

[8] Maxime Chaveroche, Franck Davoine, and Véronique Cherfaoui. Focal points and their implications for möbius transforms and dempster-shafer theory. *Information Sciences*, 555:215–235, May 2021.

[9] Alain Colmerauer, Henri Kanoui, Robert Pasero, and Philippe Roussel. Un systeme de communication homme-machine en francais. *Technical Report, Groupe Intelligence Artificielle, Universite d' Aix-Marseille II*, 1973.

[10] A. P. Dempster. Upper and Lower Probabilities Induced by a Multivalued Mapping. *The Annals of Mathematical Statistics*, 38(2):325 – 339, 1967.

[11] Thierry Denœux. Conjunctive and disjunctive combination of belief functions induced by nondistinct bodies of evidence. *Artificial Intelligence*, 172(2–3):234–264, 2008.

[12] Didier Dubois and Henri Prade. A set-theoretic view of belief functions logical operations and approximations by fuzzy sets. *International Journal of General Systems*, 12(3):193–226, 1986.

[13] Jean Gordon and Edward H. Shortliffe. *The Dempster-Shafer Theory of Evidence*, page 272–292. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[14] P.Van Hentenryck. Constraint satisfaction in logic programming. *MIT Press 1989*.

[15] T. Inagaki. Interdependence between safety-control policy and multiple-sensor schemes via dempster-shafer theory. *IEEE Transactions on Reliability*, 40(2):182–188, 1991.

[16] Alexandros N. Kaltsounidis. Dempster-shafer theory computation using constraint programming. 2019.

[17] Robert Kennes. Computational aspects of the mobius transformation of graphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(2):201–223, 1992.

[18] Robert Kennes and Philippe Smets. Computational aspects of the mobius transform. 1990.

[19] Robert Kennes and Philippe Smets. The transferable belief model. *Artificial Intelligence*, 66(2):191–234, 1994.

[20] Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, 13(1):32–32, 1992.

[21] Fabien Lange and Michel Grabisch. The interaction transform for functions on lattices. *Discrete Mathematics*, 309(12):4037–4048, June 2009.

[22] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag Berlin Heidelberg, 1984.

[23] Petros Maragos. Representations for morphological image operators and analogies with linear operators. *Advances in Imaging and Electron Physics*, 177:45–187, 2013.

[24] Brian Mayoh. Constraint programming and artificial intelligence. In Brian Mayoh, Enn Tyugu, and Jaan Penjam, editors, *Constraint Programming*, pages 17–50, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[25] Thomas Reineking. Belief functions: Theory and algorithms, 2014.

[26] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to Recommender Systems Handbook*, pages 1–35. Springer US, Boston, MA, 2011.

[27] Gian-Carlo Rota. On the foundations of combinatorial theory i. theory of möbius functions. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 2:340–368, 1964.

[28] P. Roussel. Manuel de reference et d' utilisation. *Technical Report, Universite d' Aix- Marseille II*, 1975.

[29] Kari Sentz and Scott Ferson. Combination of evidence in dempster-shafer theory. 4 2002.

[30] Glenn Shafer. A mathematical theory of evidence. *Princeton, Princeton University Press*, 1976.

[31] Glenn Shafer. Perspectives on the theory and practice of belief functions. *International Journal of Approximate Reasoning*, 4(5):323–362, 1990.

[32] Philippe Smets. Belief functions: The disjunctive rule of combination and the generalized bayesian theorem. *International Journal of Approximate Reasoning*, 9(1):1–35, 1993.

[33] Luigi Troiano, Luis J. Rodríguez-Muñiz, and Irene Díaz. Discovering user preferences using dempster–shafer theory. *Fuzzy Sets and Systems*, 278:98–117, 2015.

[34] Ronald R. Yager. On the dempster-shafer framework and new combination rules. *Information Sciences*, 41(2):93–137, 1987.

[35] John Yen. Generalizing the dempster-shafer theory to fuzzy sets. 1990.

[36] Edward Yourdon and Larry L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice Hall, 1978.

[37] Kuang Zhou, Arnaud Martin, and Quan Pan. Evidence combination for a large number of sources. *2017 20th International Conference on Information Fusion (Fusion)*, pages 1–8, 2017.